معرفی آر.یو.پی

(چارچوب فرایند تولید سیستم‌های نرم‌افزاری)

محمد بدری

آکادمی نرم‌افزار

www.software-academy.com

Overview of the RUP

A Brief Guide to the Rational Unified Process (Software Development Process Framework)

Mohammad Badri
badry@ce.sharif.edu

Software Academy
www.software-academy.com

تقدیم

به پدر و مادر عزیزم، که همواره راهنما و مشوق من در زندگی بوده‌اند؛

به همسر مهربانم، که بدون یاری و همراهی‌اش هرگز قادر به نوشتن این کتاب نمی‌بودم؛

و به تمامی آنانی که برای پیشرفت و سعادت بشریت تلاش می‌کنند.

فهرســت مطالب

به نام خداوند جان و خرد
کزین برتر اندیشه بر نگذرد

پیشگفتار

به نام یگانه خالق هستی بخش، به نام او که جان را فکرت آموخت. نوشتن، همواره یکی از پیچیده‌ترین فعالیت‌های فکری بشر بوده است. یک نوشته‌ی خوب می‌تواند اثری ماندگار و ابزار ارتباطی مناسبی برای انتقال اندیشه‌ها و تجارب باشد. در عین حال، یک نوشته‌ی بَد هم می‌تواند تأثیرات مخربی بر ذهن‌ها و دیدگاه‌ها داشته باشد. کتابی که پیشِ رو دارید، یک کتاب فنی و مهندسی است. نوشتنِ کتاب‌های فنی و مهندسی به لحاظ اهمیت و جایگاه آنها خصوصاً از جنبه‌ی کاربردی‌بودن‌شان، پیچیدگی‌ها و ظرافت‌های خاص خود را دارد. چنین کتاب‌هایی باید مستند، دقیق، و تا حدی به دور از اِعمالِ نظرهای شخصی نوشته شوند؛ البته نظرهای شخصیِ نویسنده در صورتی‌که بیانگر یک تجربه و یا دانشِ قابل اثبات باشد، می‌تواند بر غنای مطالب بیافزاید.

ایده‌ی اصلی نوشتن این کتاب به اواسط سال ۱۳۸۲ برمی‌گردد. در طی چند سال تجربه و همکاری به عنوان مشاور و ناظر در پروژه‌های مختلف و نیز انجام مطالعات و پژوهش‌هایی در دانشگاه، همیشه با این سؤالاتی مواجه بودم از جمله اینکه چرا درصد زیادی از پروژه‌های در صنایعی مانند نرم‌افزار شکست می‌خورند؟ چرا تولید فراورده‌های با فناوری برتر (های‌تِک[1]) مانند سیستم‌های نرم‌افزاری، مشکل و پیچیده است؟ چرا نگهداری فراورده‌ی بدست آمده بسیار سخت و در بسیاری از مواردِ غیر اقتصادی است؟ چرا

[1] -High-Tech

فراورده‌ی تولید شده با خواسته‌های کاربران تطابق ندارند؟ چرا کیفیت فراورده‌ها مطلوب نیست؟ چرا برخی از صنایع از جمله صنعتِ نرم‌افزار در کشور ما، توان رقابت و نوآوری را ندارند؟ چرا این‌همه دوباره‌کاری داریم؟ چرا کار تیمی آنگونه که باید تحقق پیدا نمی‌کند؟ چرا با وجودی که وارد شدن به عرصه‌ی تولید در صنعتی مانند نرم‌افزار تا حد زیادی آسان است، اما بسیاری از شرکت‌ها پس از مدت کوتاهی از بین می‌روند؟

اینها و سؤالات بسیار دیگری، من و همکارانم را بر آن داشت تا به ریشه‌یابی مشکلات و یافتن راهکارهای ممکن بپردازیم. امروز، می‌توانیم پاسخ صریح و بلامنازعی به عنوان ریشه‌ی کلیدی بروز مشکلات یاد شده در یک جمله ارائه دهیم: علت به وجود آمدن مشکلات یاد شده این است که برای حل مسائل امروزی و ارائه‌ی راه‌حل‌های مطلوب، پاسخ‌های امروزی و مناسبی نداریم؛ در واقع، همیشه سعی نموده‌ایم از استراتژی‌ها و راهکاری منسوخ گذشته استفاده نماییم و کمتر به دنبال یافتن و یادگیری پاسخ‌های امروزی (یعنی پاسخ‌های مهندسی) بوده‌ایم. البته، باید اذعان داشت که به دنبال پاسخ‌های امروزی بوده‌ایم ولی متأسفانه به‌روز بودن را تنها در فناوری جستجو کرده‌ایم؛ چیزی که نتیجه‌ی مستقیمِ آن جز مصرف‌کننده‌ی همیشگی بودن نیست.

بنابراین، به طور طبیعی سؤالات بعدی این است که آیا الگویی برای موفقیت در پروژه‌های نرم‌افزاری (و نیز در سایر پروژه‌ها) وجود دارد؟ آیا راهکارهایی برای موفقیت و پرهیز از خطا و اشتباهات گذشته وجود دارد؟ آیا باید خود تجربه کنیم تا یاد بگیریم؟ آیا راهی برای بهره‌گیری از تجربه‌ی دیگران وجود ندارد؟

واقعیت این است که امروز امکانِ دسترسی به تجارب دیگران، به لطف اینکه کسانی این دستاوردها را ثبت و سازماندهی نموده‌اند، وجود دارد و بسیار هم ساده است. پاسخ به پرسش‌های اخیر را باید در توجه به فرایندِ تولید جستجو کرد، نه در فناوری و ابزار. در واقع، راه حل را باید در افراد، تعامل‌شان، فعالیت‌هایشان، توانمندترکردن‌شان، و یادگیری بهترشان جستجو نمود.

متأسفانه، وضعیت بسیاری از پروژه‌های صنعتی از جمله پروژه‌های نرم‌افزاری در کشور ما چندان مطلوب نیست. بررسی‌ها و پژوهش‌های انجام شده[1] نشان می‌دهد که یکی از مهم‌ترین دلایل عدمِ موفقیت پروژه‌های نرم‌افزاری، عدمِ بکارگیری اصولِ و تکنیک‌های نوین مهندسی در این پروژه‌ها می‌باشد. نداشتن یک دیدِ فرایندنگر، عدم توجه کافی به نیازها و خواسته‌های واقعی مشتری، توجه ناکافی به نتیجه‌ی نهایی، و نداشتن اطلاع کافی از تجربیات و راهکارهای موفق، از جمله دلایل مهم و ریشه‌های به وجود آمدن مشکل می‌باشند؛ بطوری‌که متأسفانه، تولیدِ نرم‌افزار بیشتر به یک کار آماتوری تبدیل شده است تا یک کارِ حرفه‌ای.

با یافتن مسیر دست‌یابی به پاسخ، بر آن شدیم انواع فرایندهای تولید را بررسی نموده، دیدگاه، کاربرد، نقاط ضعف و قوت، تاریخچه، ویژگی‌های کلیدی، اصول بنیادی، و هزینه‌های بکارگیری و یادگیری آنها را مقایسه نماییم. در این میان، توجه اصلی خود را بر صنعتِ نرم‌افزار متمرکز نمودیم. هر چند که اصول و مفاهیم مشترکی در بسیاری از صنایع وجود دارد، اما با توجه ویژگی‌ها و پیچیدگی‌های خاص این صنعت و نیز وابستگی روز افزون سایر صنایع به آن، فرایند تولید نرم‌افزار، بسیار پویا و تأثیرپذیر از ریسک[2] است؛ چیزی که در میان بسیاری از صنایع دیگر (به غیر از صنایع حوزه‌ی فناوری‌های برتر) کمتر به چشم می‌خورد.

یکی از کارهای بسیار ارزنده و تأثیرگذار در زمینه‌ی فرایند تولید مربوط است به کاری که توسط شرکت نرم‌افزاری رَشنال[3] در طول بیش از دو دهه‌ی گذشته انجام شده است. متخصصان و کارشناسان این شرکت با بررسی هزاران پروژه‌ی نرم‌افزاری مختلف در سرتاسر دنیا، الگوهای شکست و نیز راهکارهای موفقیت را استخراج و در قالب یک الگوی مناسب برای تعریفِ فرایند تولید و تحت عنوانِ آر.یو.پی[4]، ثبت و سازماندهی نموده‌اند.

[1] – پژوهش‌هایِ یادشده، در طی چند سال اخیر و توسط گروهی از مهندسین و کارشناسان نرم‌افزار در آکادمیِ نرم‌افزار (Software Academy) انجام شده است.
[2] - Risk
[3] - Rational
[4] - RUP : Rational Unified Process

این کتاب به بررسی چرایی[1]، فلسفه، و چیستی[2] **آر.یو.پی**[3] اختصاص دارد. آر.یو.پی، گنجینه‌ای است ارزشمند از راهکارها و تجارب موفق در مهندسی و تولیدِ نرم‌افزار؛ آر.یو.پی، قالب و چارچوبی است برای تعریف فرایندهای مهندسی و تولیدِ سیستم‌های پیچیده‌ای مانند نرم‌افزار. این چاچوبِ فرایند به وسیله‌ی شرکتی به نامِ **رَشنال**[4] در طول بیش از دو دهه تحقیق و بررسی ایجاد گردیده است و اکنون راهبری آن را شرکت **آی.بی.اِم**[5] عهده‌دار است.

آر.یو.پی رویکردی است منظم و دارای دیسیپلین[6]، برای تخصیص مسؤولیت‌ها و مدیریت آنها در یک سازمان یا تیمِ تولیدکننده‌ی سیستم‌های نرم‌افزاری. البته، آر.یو.پی الگویی را در اختیار مهندسین و مدیران قرار می‌دهد که قابل تعمیم و گسترش به طیف وسیعی از پروژه‌ها، حتی پروژه‌های تولید فراورده‌های غیر نرم‌افزاری می‌باشد. هدف این فرایند عبارتست از: تولیدِ یک فراورده دارای کیفیت مطلوب، در یک چارچوب زمانی و هزینه‌ای قابل پیش‌بینی، که برآورده‌کننده‌ی نیازهای کاربران نهایی‌اش باشد. در واقع، این کتاب درباره‌ی راهکارهای موفق در مهندسی و تولید سیستم‌های امروزی است.

با توجه به گستردگی موضوعِ فرایند تولید و بالطبع امکان وجودِ برداشت‌ها و انتظارهای مختلف از آن، لازم است در اینجا به چند نکته توجه داشته باشیم. کتابی که پیش رو دارید، کتابی درباره‌ی استاندارد مدل‌سازی یو.اِم.اِل[7] نیست، در این کتاب به معرفی ابزارها هم نخواهیم پرداخت، متدولوژی خاصی را هم بررسی نمی‌نماییم. معرفی و تشریح مباحثِ مدیریت پروژه، مدیریت نیازمندی‌ها، تحلیل و طراحی، اصول برنامه‌نویسی، تست و مانند آن، هر یک به کتاب‌ها و منابع خاص خود نیاز دارند. ما در این کتاب با مفهومی به نام فرایند تولید[8] آشنا می‌شویم که در عین حال در برگیرنده‌ی تمام این مفاهیم می‌باشد. بنابراین، این کتاب تنها به معرفی اصول و چیستیِ این فرایند می‌پردازد. بررسی کامل‌تر هر یک از مفاهیم ذکر شده، در خودِ آر.یو.پی و نیز منابع و مراجع مرتبط قابل پیگیری می‌باشد.

---

[1] - Why
[2] - What
[3] - RUP : Rational Unified Process
[4] - Rational
[5] - IBM
[6] - Discipline
[7] - UML : Unified Modeling Language
[8] - Development Process

بحث‌های بسیار مهمی مانندِ چگونگی سفارشی‌سازی[۱] و پیکربندی[۲] آر.یو.پی، نکاتی از بکارگیری آر.یو.پی در پروژه‌های مختلف، مفاهیم مرتبط با بهبودِ فرایند[۳]، تشریحِ چگونگی تحققِ رویکردِ تکرارشونده[۴]، چگونگی طراحی و تثبیت معماری[۵]، چگونگی بکارگیری تکنیک مدل‌سازی مواردِکاربرد[۶]، کنترل کیفیت، و به طور کلی جزئیات تحقق راهکارهای موفق، موضوعات مفصلی هستند که در حیطه‌ی مطالب این کتاب نمی‌گنجند. این موضوعات را در کتاب‌های دیگری بررسی خواهیم نمود.

چگونگی بهره‌گیری از کتاب

در این کتاب سعی شده است که تا حد امکان، برخی مطالب و نکاتِ قابل توجه را که عموماً درک آنها با مشکل مواجه بوده و یا به خوبی مفهوم‌شان جا نیافتاده، برجسته‌تر نموده و بر آنها تاکید داشته باشیم. برای حفظ جامعیّت و پرهیز از اِطناب کلام، بسیاری از مطالب این کتاب، به طور خلاصه بیان شده است و از آنجایی که یکی از اهداف مهم در تدوین این کتاب، بکارگیری آن به منظور آموزش بوده است، از خواننده‌ی کتاب خواسته شده به منظور برانگیزاندن روحیه‌ی پرسش‌گری، مطالب تکمیلی را از روی منبع یا منابع دیگری از جمله خودِ آر.یو.پی، پی‌گیری نماید. در بیشتر فصل‌ها، سؤالاتی برای تحقیق و مطالعه‌ی بیشتر، ذکر شده است. خواننده با خواندن این کتاب باید علاقمند و به نوعی هدایت شود که به سراغ منبع اصلی آر.یو.پی (لوح فشرده‌ی فراورده‌ی آر.یو.پی) برود.

با توجه به اینکه بسیاری از مطالب این کتاب تا حد زیادی فنی و تخصصی می‌باشد، سعی شده واژه‌های لاتین معادل آنها که در بکارگیری منابع اصلی آر.یو.پی، آشنایی با آنها ضروری است، در صفحات مختلف تکرار شوند. برخی از شکل‌ها و نمودارها را نیز به همان شکل اصلی و بدون ترجمه آورده‌ایم.

---

[1] - Customization
[2] - Configuration
[3] - Process Improvement
[4] - Iterative
[5] - Architecture
[6] - Use-Case Modeling

بخش اولِ این کتاب را به موضوعِ **چراییِ**[1] و منشأ آر.یو.پی اختصاص داده‌ایم. بخش‌های بعدی کتاب، به بررسی **چیستیِ**[2] آر.یو.پی اختصاص یافته است. موضوعِ **چگونگیِ**[3] بکارگیری آر.یو.پی بسیار مفصل‌تر از آن بود که در قالبِ همین کتاب آن را مطرح نماییم، لذا در نوشته‌ی دیگری، چگونگی بکارگیری آر.یو.پی را به تفصیل بیان نموده‌ایم.

بخشِ اول، به بیان برخی مقدمات از جمله مفاهیم کلیدی مهندسی نرم‌افزار، آمارهایی از وضعیت موفقیت و عدم موفقیت پروژه‌های نرم‌افزاری، بررسی برخی نشانه‌های شکست (در فصل اول)، و نیز معرفی راهکارها و تجارب موفق (در فصل دوم) اختصاص دارد.

بخشِ دوم، به معرفی تعاریف و مفاهیم کلیدی مرتبط با آر.یو.پی، ساختار و ماهیت آن (فصل سوم)، بررسی فلسفه و روحِ آر.یو.پی، و نیز ویژگی‌های کلیدی آن (فصل چهارم) اختصاص دارد.

بخشِ سوم، دارای پنج فصل است که به ترتیب به معرفی چرخه‌ی تولید[4] فراورده از منظر آر.یو.پی، معرفی اهداف و ویژگی‌های اولین فاز (فازِ آغازین)، معرفی اهداف و ویژگی‌های دومین فاز (فازِ معماری)، معرفی اهداف و ویژگی‌های سومین فاز (فازِ ساخت)، و معرفی اهداف و ویژگی‌های آخرین فاز از چرخه‌ی تولید (یعنی فازِ انتقال) اختصاص یافته است.

بخشِ چهارم، در قالب ده فصل به ترتیب به معرفی ساختار محتوایی (استاتیک) فرایند تولید، دیسیپلین‌های مدیریت پروژه، مدل‌سازی سازمان، نیازمندی‌ها، تحلیل و طراحی، پیاده‌سازی، تست، اِستقرار، محیط، و مدیریت پیکربندی و تغییرات اختصاص پیدا کرده است.

آشنایی با مفهوم و فلسفه‌ی آر.یو.پی را به همه‌ی مهندسین و مدیران به خصوص مهندسین نرم‌افزار توصیه می‌نماییم. حتی اگر از آر.یو.پی در محیط کار خود استفاده نمی‌نمایید، آگاهی داشتن از محتوای این گنجینه‌ی ارزشمند، می‌تواند بسیار راهگشا و سودمند باشد.

---

[1] - Why
[2] - What
[3] - How
[4] - Development Process

در ضمن، مطالب تکمیلی این کتاب را می‌توانید در سایت *http://www.unifiedProcess.info* بیابید. در این سایت علاوه بر مطالب تکمیلی، امکاناتی برای تبادل نظر و تجربیات موفق نیز فراهم خواهد شد. بدین ترتیب، شما نیز قادر خواهید بود تجارب و دیدگاه‌های خود را با دیگر کارشناسان و صاحبنظران به اشتراک بگذارید. باشد که روزی ما نیز در عرصه‌های فنی و مهندسی، صاحب فکر، نظر، و ایده‌های نو برای جهانیان باشیم.

به هر حال، امیدوارم این کتاب ارزش خواندن داشته باشد و نقطه‌ی شروعی باشد که بتوان بر اساس آن در پیشبرد مؤثرتر و کارآمدتر صنعت نرم‌افزار و صنایع مرتبط تلاش نماییم.

محمد بدری
آکادمی نرم‌افزار
خردادماه ۱۳۸۵

بخش اول

فصل اول: مقدمه‌ای بر مهندسی و تولیدِ نرم‌افزار

فصل دوم: راهکارهای موفق در مهندسی نرم‌افزار

فصل اول

مقدمه‌ای بر مهندسی و تولیدِ نرم‌افزار

مهم‌ترین موضوعات مورد بررسی در این فصل، عبارتند از:

- تاریخچه‌ی مختصری از مهندسی نرم‌افزار
- ارائه‌ی تعریف مختصری از مهندسی نرم‌افزار و بررسی تفاوت آن با سایر زمینه‌های مهندسی
- معرفی برخی اصطلاحات و مفاهیم پایه‌ای در دنیای نرم‌افزار
- بررسی جایگاه و اهمیت مهندسی در صنعت نرم‌افزار
- ارائه‌ی آمارهایی از میزانِ عدم موفقیت در پروژه‌های نرم‌افزاری

۱ مقدمه‌ای بر مهندسی و تولیدِ نرم‌افزار

بی‌گمان، نرم‌افزار یکی از پیچیده‌ترین و در عین حال قابل انعطاف‌ترین دستاوردهای بشر می‌باشد. با وجودی‌که بیش از چند دهه از پیدایش نرم‌افزار نمی‌گذرد، این پدیده‌ی شگفت‌آور قرن بیستم، به عنوان یکی از مؤلفه‌های کلیدی فناوری‌هـای نوین اطلاعات و ارتباطات[1]، تاثیر شـگرفی بر کلیه‌ی جوانب زنـدگی بشـر داشته است. امروزه نرم‌افزار، سوخت لازم برای راه‌اندازی و به‌حرکت درآوردن موتورهای اقتصادِ نوین تلقی می‌شود. هیچ سازمان و کسب‌وکار نوینی، نمی‌تواند بدون نرم‌افزار به حرکت و تکامل خود ادامه دهد.

به‌طور کلی فناوری اطلاعات و ارتباطات و به طور خاص نرم‌افزار، گونه‌های جدیدی از ارتباطات و تعاملات را شکل داده است. به کمک این پدیده و دستاوردِ منحصر به فرد، قادر هستیم اطلاعات را به شکل‌هایی که تاکنون حتی تصور هم نمی‌کردیم، ایجاد نموده و الگوهای جدیدی را در میان آنها کشف نماییم. روش‌های درمان بیماری‌ها، روش‌های یادگیری، روش‌های کسب و کار، و به طور خلاصه، کلیه‌ی جوانب زندگی به شدت تحت تأثیر قرار گرفته است. به کمک این فناوری‌ها، بشر توانسته پا را از مرزها و قلمروهای پیشین فراتر نهاده و قدم در دنیای پر رمز و راز هستی نهد. دسترسی به فضای بیکران آسمان‌ها از یک سو و وارد شدن به دنیای اتم‌ها (در مقیاس نانو) از سوی دیگر، نمونه‌های آشنایی از تأثیرات و جلوه‌های بکارگیری فناوری‌های نوین اطلاعات و ارتباطات و به خصوص نرم‌افزار می‌باشد.

در طول چند دهه‌ی اخیر، با کمک رایانه‌ها و نرم‌افزارهای مختلف، حجم دانش بشری چندین برابر شده است. در آینده‌ی بسیار نزدیک، هر یک از ما شاهدِ بکارگیری نرم‌افزار در منزل، خودرو، تلویزیون، ساعتِ مچی، کتاب، و حتی لباس‌های خود خواهیم بود.

[1] - Information and Communication Technologies

اما به واسطه‌ی تغییرات بسیار سریع و غافل‌گیرکننده‌ی فناوری‌های نوین اطلاعاتی و ارتباطی و به‌طور خاصْ نرم‌افزار، و به موازاتِ آن، تغییر نیازها، خواسته‌ها، و انتظارات استفاده‌کنندگان از نرم‌افزار و قابلیت‌های آن، طراحی و تولید نرم‌افزار، بسیار پیچیده می‌باشد. عوامل دیگری مانند رقابت شدید[۱]، کمبود نیروی متخصص و حرفه‌ای، عدمِ دسترسی به دانش و تجربه‌ی موفق دیگران، لزوم تولید سریع، لزوم تولید مقرون به صرفه، نیاز روز افزون به همکاری میان رشته‌های مختلف، و مهم‌تر از همه، عدمِ استفاده‌ی مناسب از اصول و مبانی مهندسی در طراحی تولید نرم‌افزار، این صنعت را با چالش‌های بسیاری روبرو نموده است.

حدود ۵۰ سال پیش، یعنی در اوایل پیدایش نرم‌افزار، استفاده‌کنندگانِ این فراورده‌ی نوین، همان طراحان[۲] و تولیدکنندگان آن بودند. در آن زمان، نرم‌افزار عمدتاً برای محاسبات و حل مسائل ریاضی استفاده می‌شد. وجود زبان‌های سطحِ پایین[۳] و محدودیت‌های سخت‌افزاری (کمبود حافظه و سرعت پردازشِ کم) از دیگر مشخصه‌های دوران اولیه‌ی پیدایش نرم‌افزار است. در آن روزهای اولیه، نرم‌افزار چیزی جدا از سخت‌افزار نبود و حتی برای فروش سخت‌افزار، بطور رایگان در آن تعبیه می‌شد! اما با گسترش دامنه‌ی کاربرد رایانه و به تبعِ آن نرم‌افزار در زمینه‌های مختلف، به مرور شرایطی به وجود آمد که استفاده‌کنندگان و کاربرانِ نرم‌افزار از طراحان و تولیدکنندگان آن جدا شدند؛ سازمان‌ها و شرکت‌هایی به‌وجود آمدند که کارشان صرفاً تولید نرم‌افزار بود. حالا دیگر نرم‌افزار قیمت داشت و اتفاقاً برخلاف روندِ کاهش قیمت در سخت‌افزارها، روز به روز بر قیمت نرم‌افزار افزوده می‌شد. نیازهای جدید استفاده‌کنندگان فراتر از محاسبات (رایانِش) بود. آنها به مدیریت اطلاعات نیاز داشتند. پیدایش زبان‌های سطح بالا و رفع محدودیت‌های سخت‌افزاری، از دیگر مشخصه‌های عصر جدید نرم‌افزار می‌باشد. درست در همین زمان است که اولین شکست‌ها و مشکلات نیز خود را نشان دادند. مشکلات و چالش‌ها به قدری جدی و پر هزینه بود که از آن به بحرانِ نرم‌افزار[۴] یاد می‌شد.

سرانجام برای اولین بار، در سال ۱۹۶۸ و در یک کنفرانس که توسط ناتو[۵] در کشور آلمان برگزار شده بود، بر لزوم مهندسیِ[۱] این دستاورد جدیدِ بشر، یعنی نرم‌افزار، تأکید شد. از آن زمان به بعد، با گسترش

---

[۲] – وارد شدن در صنعت نرم‌افزار و ایجاد یک شرکت نرم‌افزاری، کار آسانی به نظر می‌رسد.

[2] - Designers
[3] - Low-level Languages
[4] - Software Crisis
[5] - NATO

تکنیک‌های مهندسی، ابزارها، و دانش و تجربه، صنعت نرم‌افزار به یکی از صنایع برتر جهانی تبدیل شده است.

در جدول ۱-۱، فازهای تحولِ محاسبات[۲] (رایانِش) در طولِ تاریخ و اثر این تحولات بر نرم‌افزار، نشان داده شده است.

جدول ۱-۱

فازهای تحول و تکامل محاسبات در طول تاریخ

فازها	توصیف	اثر بر نرم‌افزار
رایانه‌های بسیار بزرگ (Mainframe)	تمام محاسبات (رایانِش‌ها) به صورت دسته‌ای (Batch) به وسیله‌ی یک رایانه‌ی مرکزی بسیار بزرگ انجام می‌شد.	از آنجایی که رایانه‌ها تنها در سازمان‌های بسیار بزرگ استفاده می‌شدند. بنابراین تمام نرم‌افزارهای کاربردی برای انجام محاسبات علمی و مهندسی پیچیده یا انجام خدمات پشتیبانی سازمان، استفاده می‌شد.
رایانه‌های اشتراک زمانی (Time-sharing)	چندین کاربر قادر بودند که روی یک رایانه‌ی مرکزی نسبتاً بزرگ و گران‌قیمت، به صورت تعاملی (interactive)، کار کنند.	در تولید نرم‌افزارها علاوه بر نیازهای سازمان، به نیازهای هر یک از استفاده‌کنندگان نیز توجه شد.
رایانه‌های شخصی (Personal)	هر یک از کاربران برای خود یک رایانه در اختیار دارد. بنابراین هر شخص می‌تواند نرم‌افزارهای مورد نیازش را مستقیماً تهیه نموده و روی رایانه‌اش نصب نماید.	بازار بزرگی از مصرف‌کننده‌های محصولات نرم‌افزاری با طیف وسیعی از نیازها و نیز روش‌های مختلف توزیع و فروش ایجاد شد.
رایانه‌های شبکه‌ای (Network Comp.)	همه‌ی رایانه‌ها می‌توانند با هم ارتباط برقرار نمایند. نرم‌افزارها می‌توانند در میان رایانه‌های مختلف، توزیع شوند. برای مثال، در رایانشِ مخدوم – خادم (client-server) که ترکیبی است از رایانه‌های شخصی و اشتراک زمانی، نرم‌افزارهای کاربردی به اشتراک گذاشته شده روی خادم (server) به رایانه‌های شخصی متصل به آن، سرویس می‌دهد.	نرم‌افزارهای کاربردی به طور وسیع در فرآیندهای کسب و کار سازمان‌ها وارد شد. توزیع‌شدگی نرم‌افزار، از ویژگی‌های بارز این تحول می‌باشد. سیستم‌های نرم‌افزاری، بستر ارتباطی جدیدی میان افراد و گروه‌ها و نیز دسترسی توزیع‌شده به اطلاعات فراهم نمود. چالش‌های جدیدی مانند امکان کار سیستم‌های مختلف با هم (interoperability) و نیز امکان انتقال میان بسترهای مختلف (portability) مطرح گردید.

در این کتاب، مهندسی نرم‌افزار را به شکل زیر تعریف می‌نماییم:

مهندسی نرم‌افزار[۳]، شاخه‌ای است از مهندسی، که با بهره‌گیری از **دانشِ علمی**[۴]، به ارائه‌ی راه‌حل‌هایی **مقرون به‌صرفه**[۵]، در قالبِ **دستاوردهای نرم‌افزاری**[۶] و به منظور حل **مسائل و مشکلاتِ عملی**[۱] و **خدمت به جامعه‌ی بشری**، اقدام می‌نماید.

[1] _ Engineering
[2] - Computing
[3] - Software Engineering
[4] - Scientific Knowledge
[5] - Cost-effective
[6] - Software Artifacts

مهندسی نرم‌افزار یک **حرفه**[۲] است نه یک **شغل**[۳]. حرفه گسترده‌تر از شغل است و در آن فرد به یادگیری و گسترش دانسته‌های خود می‌پردازد. شغل از پیش تعریف شده و محدود است؛ بستن پیچ‌های یک دستگاه و نوشتن یک تکه از برنامه برای اتصال به بانک اطلاعاتی. حرفه بر گرداگردِ فرد به‌وجود می‌آید. در حالی که، شغل متعلق به کارفرماست.

یک فردِ حرفه‌ای، کسی است که به جای انجام وظیفه‌های جداگانه، مسئول نتیجه‌گیری از کلِ کار است. بیمار برای معاینه‌ی گلو، گرفتن فشار خون، و یا بازدید قلب به نزد پزشک نمی‌رود. هدف از مراجعه به پزشک، خوب شدن و به دست آوردن سلامتیِ دوباره است. توجه پزشک به نتیجه نهایی و نه فعالیت‌هایی است که در راه رسیدن به آن انجام می‌دهد. در میدان کارهای حرفه‌ای، بالاترین احترام از آنِ کسی است که دانش و توانمندی بیشتری دارد؛ بهترین وکیل دادگستری، بهترین مهندس معمار، بهترین پزشک، و بهترین مهندس نرم‌افزار.

از دید کلّی، سه واژه ویژگی‌های کار حرفه‌ای[۴] را روشن و آشکار می‌کنند: **مشتری**[۵]، **فرایند**[۶]، و **نتیجه**[۷]. مهندسی نرم‌افزار نیز یک کارِ حرفه‌ای به شمار می‌آید. بنابراین، مهندسِ نرم‌افزار فردی است حرفه‌ای، که خود را در برابرِ مشتری مسئول می‌داند و همواره در پی دستیابی به ارزش[۸] موردِ نیاز اوست. رسالت و مأموریت[۹] یک مهندسِ نرم‌افزار، حل مشکل مشتری است. بنابراین بایستی سراسرِ فرایندِ کار را اجرا نماید و به نتیجه‌ی دلخواه مشتری دست یابد.

فعالیت‌های مهندسی، عمدتاً دو گونه‌اند: ۱- طراحی روتین[۱۰] که شامل ارائه‌ی راه‌حل برای مسائلی آشنا و استفاده‌ی مجدد از راه‌حل‌های قبلی می‌باشد و ۲- طراحی نوآورانه[۱۱] که عبارت است از ارائه‌ی راه‌حل‌هایی نو و بدیع برای مسائلی نا آشنا. مهندسی نرم‌افزار، اغلب با فعالیت‌های دسته‌ی دوم، یعنی طراحی نوآورانه سر و کار دارد.

---

[1] - Practical Problems
[2] - Career
[3] - Job
[4] - Professional
[5] - Customer
[6] - Process
[7] - Result
[8] - Value
[9] - Mission
[10] - Routine Design
[11] - Innovative

یک مهندس نرم‌افزار باید علاوه بر آشنایی با برنامه‌نویسی، دارای دانش و تخصص‌هایی نظیر مهارت‌های فنی، مدیریت پروژه، مهارت‌های شناختی[۱]، درک سازمان‌ها در سطح گسترده[۲]، توانایی کار تیمی، یادگیری، و کسب دانش در زمینه‌ی حوزه‌ی فعالیت خود باشد. متأسفانه، دوره‌های آموزشی مهندسی نرم‌افزار، کمتر به این مفاهیم پرداخته‌اند و بنابراین لازم است بازنگری عمیقی در ساختار و محتوای دوره‌های آموزشی مهندسی نرم‌افزار صورت پذیرد.

با توجه به ویژگی‌های خاصِ نرم‌افزار، مهندسیِ آن نیز تفاوت‌هایی با سایر زمینه‌ها و شاخه‌های مهندسی دارد. درک این تفاوت‌ها برای به‌دست آوردن نگرشی مناسب نسبت به مهندسی و تولید نرم‌افزار، ضروری است. برخی از این تفاوت‌ها، عبارتند از:

- مهندسی نرم‌افزار هنوز در حال بکارگیری (و آموزش) به روشی غیرِ اصولی[۳] می‌باشد.
- نسبت به سایر تخصص‌های مهندسی، از سازماندهی کمتری برخوردار می‌باشد.
- نسبت به برخی از رشته‌های مهندسی، استانداردهای کمتری برای طراحی نرم‌افزار وجود دارد.
- بسیار جوان‌تر از دیگر رشته‌های مهندسی است.
- در بسیاری از رشته‌ها و تخصص‌های مهندسی، در مقطعی از زمان، طراحی کامل شده و اصطلاحاً بسته می‌شود؛ بعد از آن، پیمانکار کمتر در طراحی مذکور، دخل و تصرف خواهد داشت. در صورتی‌که در مهندسی نرم‌افزار، بستن طراحی و کامل شدن آن، تا انتهای پروژه به طول می‌انجامد.
- در مهندسی و تولید نرم‌افزار، فرآیندهای استاندارد، کمتر استفاده می‌شود.
- معمولاً، دستاوردهایی مانند ساختمان‌ها و پل‌ها با هزینه‌ی پیش‌بینی شده و سر موعدِ مقرر ساخته می‌شوند؛ در حالی‌که نرم‌افزار به ندرت، سر موقع و با هزینه‌ی از قبل پیش‌بینی شده ایجاد می‌گردد.
- فراورده و دستاورد اصلی مهندسی نرم‌افزار (یعنی فراورده‌ی نرم‌افزاری)، دارای ماهیتی غیرقابل لمس می‌باشد.
- اثرات ناشی از شکست یک پروژه‌ی نرم‌افزاری برای همگان قابل لمس نیست. در مقابل اثرات ناشی از فرو ریختن یک پل یا خرابی یک اتوموبیل را همه می‌توانند درک نمایند.

---

[1] - Cognitive
[2] - Enterprise
[3] - Non-Systematic

این تفاوت‌ها از یک طرف و پیچیده‌تر شدن سیستم‌ها، بزرگ‌تر شدن اندازه‌ی آنها، و افزایش روز افزون اهمیت نرم‌افزار از سوی دیگر، مهندسین نرم‌افزار را با چالش‌ها و معضلات زیادی روبرو نموده است. در واقع، علیرغم اهمیت و نقش کلیدی نرم‌افزار در اقتصادِ نوین و جایگاه منحصر به فرد آن در سایر دستاوردها و مصنوعات بشری (اعم از سخت‌افزار، کسب و کار، و یا سازمان) و نیز تأثیر آن بر سایر صنایع، هر روز خبرهایی از عدم موفقیت در طیف وسیعی از پروژه‌های نرم‌افزاری (پروژه‌هایی که محصول آنها یک سیستم نرم‌افزاری می‌باشد) به گوش می‌رسد.

بر اساس آمارهای معتبری که توسط مؤسساتی مانند آی.دی.سی[1] و اِس‌تَن‌دیش گروپ[2] و در پی بررسی هزاران پروژه‌ی تولید نرم‌افزار در اَبعاد و در زمینه‌های مختلف، تهیه شده است، درصد زیادی از پروژه‌های نرم‌افزاری در دنیا با شکست و عدم موفقیت مواجه می‌شوند. قبل از بررسی برخی از این آمارها، اجازه دهید مفهوم موفقیت و عدم موفقیتِ یک پروژه‌ی نرم‌افزاری را توصیف نماییم.

یک **پروژه‌ی موفق**[3] نرم‌افزاری، پروژه‌ای است که در یک **محدوده‌ی زمانی** از قبل برنامه‌ریزی شده و با **بودجه‌ای از قبل پیش‌بینی شده**، یک **فراورده‌ی نرم‌افزاریِ** دارای **کیفیتِ مطلوب** (یعنی کیفیتی مطابق با خواسته‌ها و نیازهای واقعی کاربران) در آن تولید می‌گردد.

البته برای تکمیل تعریف ارائه شده از یک پروژه‌ی موفق، لازم است برای دو مفهوم اساسی در این تعریف، یعنی فراورده‌ی نرم‌افزاری و نیز مفهومِ کیفیت، تعاریفی ارائه نماییم.

به زبان ساده، یک **فراورده‌ی نرم‌افزاری**[4] عبارتست از یک برنامه‌ی نرم‌افزاری قابل اجرا[5] به علاوه‌ی مجموعه‌ی مستندات و دست‌نامه‌های کاربران آن (البته گاهاً مجموعه‌ی کدهای برنامه نیز در قالب فراورده‌ی نهایی ارائه می‌گردد). توجه داشته باشید که یک فراورده‌ی نرم‌افزاری باید به اصطلاح، قابلیتِ **خودپشتیبانی**[6] داشته باشد؛ بر اساس این قابلیت، همه‌ی کاربران یک سیستمِ نرم‌افزاری، اَعم از کاربران ساده، مدیران

[1] - IDC : International Data Corporation
[2] - Standish Group
[3] - Successful Project
[4] - Software Product
[5] - Executable
[6] - Self-Supportability

سیستم[1]، و یا مسئولین نگهداری و به‌روز رسانی آن، باید قادر باشند بدون نیاز به حضورِ تولیدکنندگان و پدیدآورندگان آن، تمام انتظاراتشان را از سیستم برآورده نمایند.

کیفیت[2]، مفهوم بسیار پیچیده‌ای است. مسلم است که همه کیفیت را می‌پسندند. اما شاید کمتر کسی بتواند تعریف کاملی از آن ارائه نماید. دو تعریف کلّیِ مفهوم کیفیت، عبارتند از: تطابق با نیازمندی‌ها[3] و متناسب بودن برای استفاده[4]. این دو تعریف که تا حد زیادی به هم مرتبط می‌باشند، تفاوت کوچکی نیز دارند؛ متناسب بودن برای استفاده، تأکید بیشتری بر نقشِ نیازمندی‌ها و انتظاراتِ[5] مشتری[6] و کاربر[7] دارد.

شاید از خود بپرسید که حد و اندازه‌های کیفیت چه میزان است و یا چگونه می‌توان کیفیت را سنجید؟ در فصل‌های آتی این کتاب، اشاره خواهیم نمود که دیدگاه موردِ پذیرشِ امروزی از این مفهوم، کیفیتِ مطلوب و کافی[8] است و نه بهترین کیفیت. داشتن معیارهایی برای سنجش کیفیت، شرط اطمینان از موفقیت یک پروژه می‌باشد.

حال می‌توانیم مفهومِ **عدمِ موفقیتِ** یک پروژه‌ی نرم‌افزاری را بررسی نماییم. در شکل ۱-۱، مثلثِ موفقیت نشان داده شده است. بر اساس این شکل، تنها در صورتی یک پروژه، موفق تلقی می‌گردد که در محدوده‌ی زمان و هزینه‌ی پیش‌بینی شده، خاتمه یافته و نتیجه‌ی حاصل از آن دربرگیرنده‌ی تمامِ نیازمندی‌های مورد توافق با مشتری باشد؛ هر شرایطی غیر از این، عدم موفقیت یا **شکستِ**[9] پروژه محسوب می‌شود.

---

[1] - Administrators
[2] - Quality
[3] - Conformance to requirements
[4] - Fitness for use
[5] - Expectations
[6] - Customer
[7] - User
[8] - Good Enough Quality
[9] - Failure

شکل ۱-۱

مثلث موفقیتِ پروژه

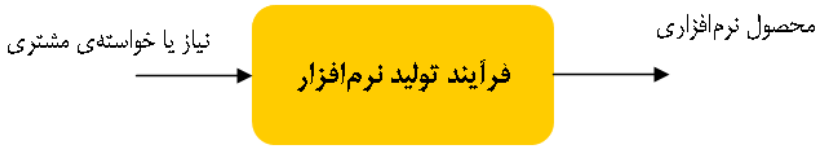
هر پروژه‌ی نرم‌افزاری، چه بزرگ و چه کوچک، چه موفق و چه ناموفق، مراحلی را طی می‌نماید که در طی آن یک خواسته یا نیاز، به فراورده‌ای نرم‌افزاری تبدیل می‌شود. الگو و قالبی[1] که چگونگی طیِ مراحل مختلفِ یک پروژه را تعریف می‌نماید، اصطلاحاً **فرایندِ تولیدِ نرم‌افزار**[2] نامیده می‌شود. در این کتاب، از واژه‌ی فرایند، به طور اختصار، به جای فرایندِ تولید نرم‌افزار، استفاده خواهیم کرد.

شکل ۱-۲، فرایند تولید نرم‌افزار و ورودی و خروجی‌های آن را نشان می‌دهد. بر اساس تعریف خوبی که در فصل سوم این کتاب نیز ارائه شده است، یک فرایند تولید، به ما می‌گوید که برای دستیابی به هدف مطلوب در یک پروژه که همان تولید فراورده‌ای نرم‌افزاری با کیفیت مطلوب می‌باشد، چه کسی، چه کاری را، چه موقع، و چگونه باید انجام دهد. در واقع، بدون داشتن تعریف مشترکی از فرایند، هماهنگی و انجام کار تیمی در یک پروژه‌ی نرم‌افزاری، امکان‌پذیر نخواهد بود.

[1] - Template
[2] - Software Development Process

شکل ۱-۲

فرآیند تولید محصول نرم‌افزاری



باید توجه داشته باشید که امروزه، تولید نرم‌افزار، کاری است تیمی و همانگونه که در فصل چهارمِ این کتاب اشاره خواهیم نمود، در تولید یک فراورده‌ی نرم‌افزاری موفق، صرف‌نظر از اندازه و ابعاد آن، تنها یک تیم شرکت می‌نماید؛ این تیم بدون داشتن یک فرهنگِ کاری مشترک، که همان فرایند تولید می‌باشد، هرگز نمی‌تواند موفق باشد.

در اینجا ممکن است کسی تصور نماید که داشتن یک فرایندِ خوبْ تعریف شده[1]، لزومی ندارد. در واقع این‌طور به نظر می‌آید که با جمع شدن چند نفر برنامه‌نویس، مدیر پروژه، تحلیل‌گر، و طراح در کنار هم و تشکیل یک گروه، می‌توان یک پروژه‌ی نرم‌افزاری را انجام داد و یا ممکن است شما پیش از این، خود به تنهایی یک پروژه را به طور موفق انجام داده باشید و برایتان اهمیت و جایگاه فرایند چندان آشکار نباشد. کافی است از خود بپرسید که چند درصد احتمال دارد دوباره یک پروژه‌ی موفقِ دیگر انجام دهید؟ یا اینکه آیا پروژه‌ای وجود دارد که بتوان به تنهایی انجام داد؟ آن گروه نرم‌افزاری، چقدر در کارشان دوباره‌کاری داشته‌اند؟ آیا نرم‌افزار به موقع تحویل شد؟ کیفیت آن چگونه بود؟ آیا هزینه‌ها را درست پیش‌بینی کرده بودند؟ آیا فراورده‌ی ارائه شده مورد قبول همه‌ی مشتریان و کاربران بود؟ آیا در نگهداری نرم‌افزار مشکلی ندارید؟ اینها و بسیاری سوالات دیگر، باید شما را به تأمل واداشته باشد.

به‌هر حال در این فصل، لازم نیست بیشتر از این، به جزئیات بپردازیم. اما نکته‌ی مهمی که باید به خاطر داشته باشید اینست که هر پروژه‌ای چه موفق و چه ناموفق، دارای فرایند خاص خود می‌باشد. اما هر فرایندی،

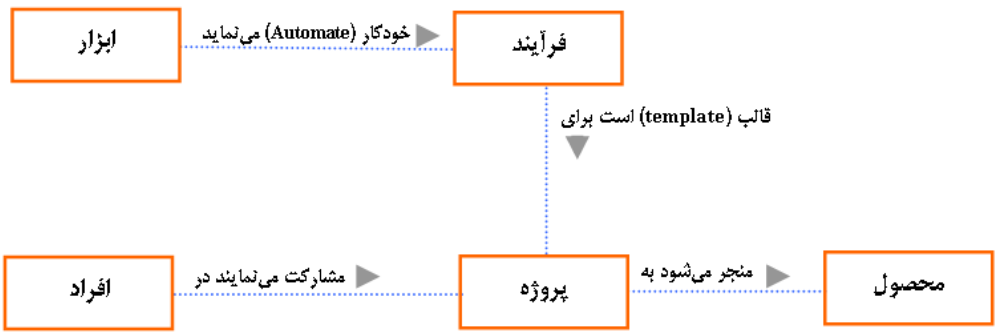
[1] - Well-Defined Process

قالب مناسبی برای موفق شدن فراهم نمی‌کند. زیرا مسلم است که **کیفیتِ فرایند**[۱] بر کیفیت فراورده‌ی خروجی آن تاثیرگذار می‌باشد. یک فرایند مطلوب و با کیفیت، فرایندی است که به کمکِ آن، یک پروژه‌ی تولید نرم‌افزار، قابل پیش‌بینی[۲] و تجاربِ موفقِ کسب شده در طی آن، قابلِ تکرار مجدد[۳] باشد. بنابراین، فرایند باید به خوبی سازمان‌دهی[۴] و مستندسازی شده باشد.

شکل ۱-۳، نشان‌دهنده‌ی ارتباط میان فرایند، پروژه، افراد (تولیدکنندگان و استفاده‌کنندگان)، فراورده و ابزارها نشان داده شده است. توجه داشته باشید که در این مدل، منظور از ابزارها، مجموعه‌ی ابزارهای به اصطلاح کمک به مهندسی نرم‌افزار[۵] یا کِیْسْ تولز[۶] می‌باشد. به عنوان نمونه، می‌توان ابزارهایی مانند ابزارهای مدل‌سازی (نظیر رَشنال رُز[۷])، محیط‌های مجتمعِ تولید[۸] (نظیر محیط دات‌نِتْ، ای.کلیپس[۹])، ابزارهای تست (مانند رَشنال تستْ سویْت[۱۰])، ابزارهای مستندسازی (نظیر رَشنال سودا[۱۱]) و ابزارهای مدیریت پروژه (نظیر اِم.اِس.پی[۱۲]) را نام برد.

شکل ۱-۳

مدل ارتباط میان مفاهیم پروژه، فرایند، فراورده (محصول)، ابزار، و افراد



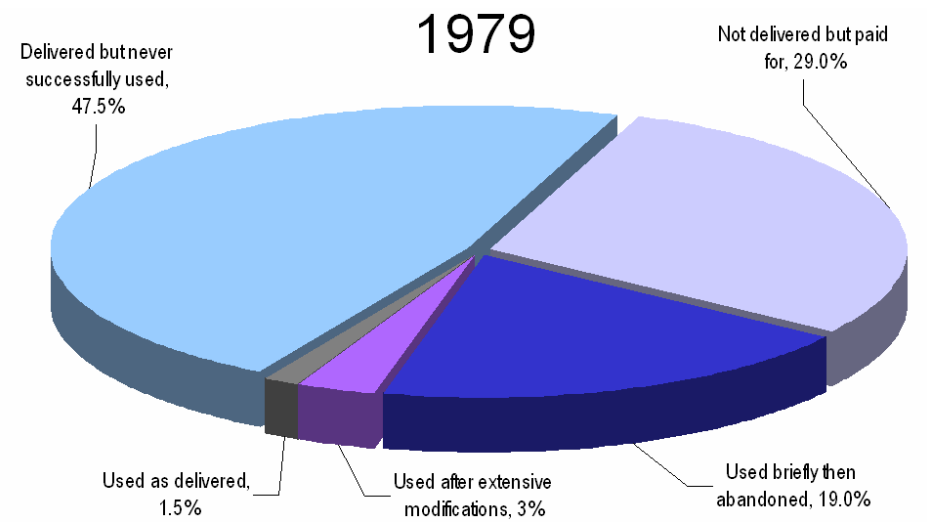
[1] - Process Quality
[2] - Predictable
[3] - Repeatable
[4] - Well-Organized
[5] - Computer Aided Software Engineering (CASE)
[6] - Case-Tools
[7] - Rational Rose
[8] - Integrated Development Environment
[9] - Eclipse
[10] - Rational Test Suite
[11] - Rational SoDA
[12] - MSP (Microsoft Project)

در تعریفِ مهندسی نرم‌افزار به عنوان یک حرفه، اشاره کردیم که یک مهندس نرم‌افزار، در هر شغل و سِمَتی که باشد، باید با کلِّ فرایند تولید آشنا باشد. بنابراین، اگر شما یک برنامه‌نویس، تحلیل‌گر، طراح، و یا مدیر پروژه هستید، برای آنکه بتوانید در حرفه‌ی خود موفق باشید باید با نگرشی جامع به تمام اَبعاد فرایندِ تولید، با مشتری و نیازهایش آشنا باشید و همواره بر دستیابی به نتیجه‌ی مطلوبِ مشتری که یک فراورده‌ی نرم‌افزاری با کیفیت می‌باشد، تمرکز داشته باشید. در این کتاب، سعی نموده‌ایم که یکی از موفق‌ترین فرایندهای تولید نرم‌افزار، یعنی آر.یو.پی، را معرفی نماییم. خواهید دید که بسیاری از مفاهیم و اصول آر.یو.پی، بسیار منطقی و در تطابق با تعریف‌ها و نیازهای امروزی تولید نرم‌افزار می‌باشد. در واقع، آر.یو.پی نه یک فرایند، بلکه چارچوبی است برای تعریف فرایندهای طیف وسیعی از پروژه‌هادر اندازه‌ها، پیچیدگی‌ها، و ملاحظات مختلف.

اکنون فرصت مناسبی است تا نگاهی به برخی از آمارهای مرتبط با موفقیت و شکست در پروژه‌های نرم‌افزاری، داشته باشیم. در شکل‌های ۱–۴ ، ۱–۵ ، و ۱–۶ برخی از این آمارها نشان داده شده است. در فصل آینده، توجه خود را به ریشه‌یابی دلایل عدم موفقیت و نیز راهکارهای دستیابی به موفقیت، معطوف خواهیم نمود.

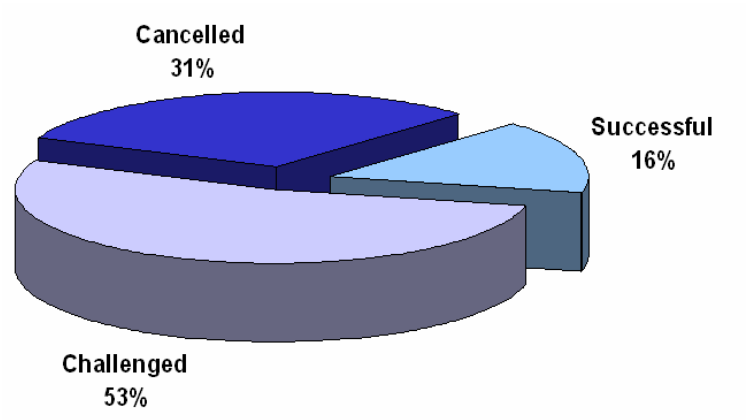
شکل ۱–۴

آمارهای مربوط به وضعیت پروژه‌های نرم‌افزاری در سال ۱۹۷۹ میلادی



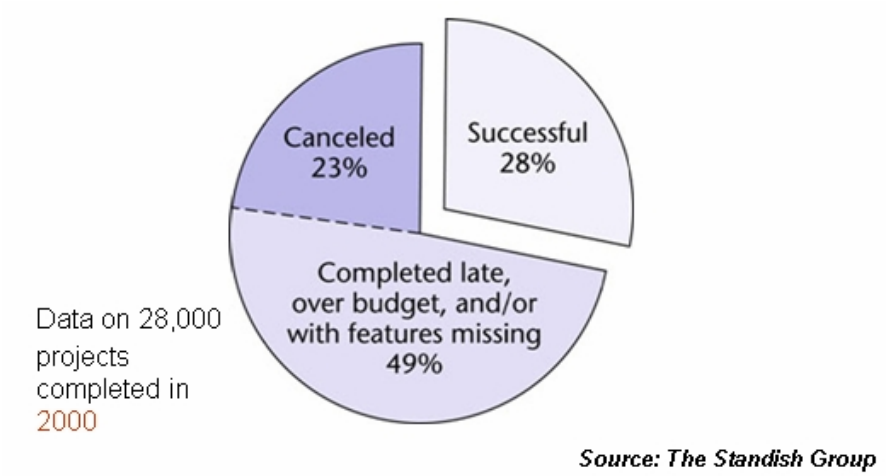
شکل ۱-۵

آمارهای مربوط به وضعیت پروژه‌های نرم‌افزاری در سال ۱۹۹۵ میلادی



شکل ۱-۶

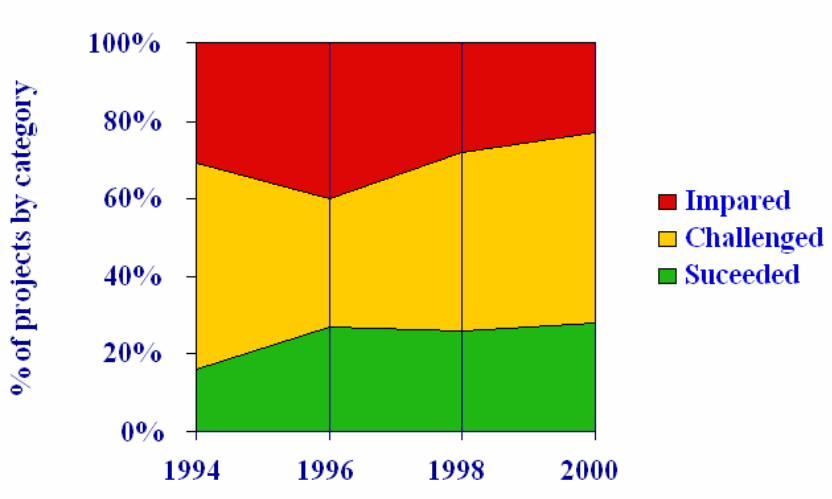
آمارهای مربوط به وضعیت پروژه‌های نرم‌افزاری در سال ۲۰۰۰ میلادی



همانگونه که در نمودار شکل ۱-۷ ، نشان داده شده است، در طول زمان و با بهبود روش‌های مهندسی و نیز تکامل ابزارها، آمارهای مرتبط با شکست و عدم موفقیت، کمتر شده است.

شکل ۱-۷

مقایسه‌ی میان درصدِ پروژه‌های موفق، شکست‌خورده، و دارای چالش در طول سال‌های مختلف



چکیده‌ی فصل

مهم‌ترین مطالبی که در این فصل مورد بررسی قرار گرفت، عبارتند از:

- اهمیت و جایگاه مهم نرم‌افزار در اقتصاد دانایی محور (اقتصاد نوین)،
- تعریف مهندسی نرم‌افزار به صورت یک حرفه و ارتباط آن با مشتری، فرایند، و نتیجه،
- تفاوت مهندسی نرم‌افزار با سایر حرفه‌های مهندسی،
- مفهوم فرایند به عنوان قالب و چارچوب یک پروژه،
- مفهوم فراورده‌ی نرم‌افزاری و ویژگی‌های آن،
- مفهوم کیفیت و ارتباط آن با نیازمندی‌ها،
- بررسی برخی آمارهای نشان‌دهنده‌ی عدم موفقیت.

اکنون آماده هستیم تا در فصل آینده به بررسی نشانه‌ها و دلایل شکست و عدم موفقیت پروژه‌های نرم‌افزاری پرداخته و با راهکارها و تجارب موفق آشنا شویم.

پرسش‌هایی برای مطالعه‌ی بیشتر

۱. با توجه به تعریفی که از شکستِ پروژه‌های نرم‌افزاری ارائه شد، بررسی نمایید که چند درصد از پروژه‌های نرم‌افزاری در کشور با شکست مواجه می‌شوند؟

۲. دلایل شکست پروژه‌های نرام‌افزاری را بررسی نمایید.

۳. در رابطه با مفهوم حرفه‌ی مهندسی نرم‌افزار و مسئولیت‌های آن تحقیق نمایید.

۴. مهندسی نرم‌افزار را با مهندسی ساختمان مقایسه کنید.

۵. چه ارتباطی میان فرایند و کارِ تیمی وجود دارد؟

۶. انواع فرایندهای تولید نرم‌افزار را بررسی و با هم مقایسه نمایید.

۷. درباره‌ی انواع مشاغل مرتبط با حرفه‌ی مهندسی نرم‌افزار تحقیق نمایید.

۸. اصول و قواعد مهندسی نرم‌افزار را بررسی نمایید.

۹. در رابطه با مفهوم متدولوژی و انواع آن در مهندسی و تولید سیستم‌های نرم‌افزاری تحقیق نمایید.

۱۰. در رابطه با آینده‌ی مهندسی نرم‌افزار و خط مشی‌های مرتبط با آن، تحقیق نمایید.

منابع و مراجع

[1]. Ivar Jacobson, Grady Booch, James Rumbaugh, (1999). *The Unified Software Development Process*, Reading, MA: Addison-Wesley.

[2]. Walker Royce, (1998). *Software Project Management: A Unified Framework*. Reading, MA: Addison-Wesley.

[3]. Steve McConnell, (2003). *Professional Software Development: Shorter Schedules, Higher Quality Products, More Successful Projects, Enhanced Careers*, Reading, MA: Addison Wesley.

[4]. Robert L. Glass, (2002). *Facts and Fallacies of Software Engineering*, Reading, MA: Addison Wesley.

[5]. Scott E. Donaldson, Stanley G. Siegel, (2000). *Successful Software Development*, Reading, NJ: Prentice Hall PTR.

[6]. Goertzel, B., and P. Pritchard. (2002). *The Internet economy as a complex system*. Available Online: http://www.goertzel.org/papers/ecommerce.html

[7]. Software Academy. (2006) "Unified Process Knowledge Base," Available at: http://www.unifiedProcess.info/

[8]. Slaughter, S. A., D. E. Harter, and M. S. Krishnan. (1998). *Evaluating the cost of software quality*. Communications of the ACM 41 (8): 67–73.

[9]. Schaller, R. R. (1997). *Moore's law: past, present and future*. IEEE Spectrum 34 (6): 52–59.

[10]. Pressman, R. S. (2000). *Software engineering: A practitioner's approach*. 5th ed. New York: McGraw-Hill.

[11]. Meyers, J. 1993. *A short history of the computer*. Available Online: http://www.softlord.com/comp.

[12]. Boehm, B. W., and K. Sullivan. (2000). *Software economics: A roadmap.* In The future of software engineering, ed. A. Finkelstein. 22d International Conference on Software Engineering.

فصل دوم

راهکارهای موفق در مهندسی نرم‌افزار

مهم‌ترین موضوعات مورد بررسی در این فصل، عبارتند از:

- بررسی نشانه‌های عدم موفقیت در پروژه‌های نرم‌افزاری
- تحلیل دلایل مشکلات و معضلاتِ پروژه‌های نرم‌افزاری
- معرفی راهکارها و تَجارب موفق در دنیای مهندسی و تولیدِ نرم‌افزار
- بررسی ارتباط آر.یو.پی با راهکارها و تجارب موفّقِ مهندسی نرم‌افزار

۲ راهکارهای موفق در مهندسی نرم‌افزار

در انتهای فصلِ اول، آمارهایی اِرائه گردید که حاکی از عدم موفقیتِ درصدِ زیادی از پروژه‌های نرم‌افزاری بود؛ در این فصل، در تلاش برای حلِ مشکلات و اِرائه‌ی راهکارهای موفق، ابتدا برخی از نشانه‌های شکستِ یک پروژه‌ی نرم‌افزاری را بررسی نموده و پس از آن دلایلِ ریشه‌ای رُخ دادن این نشانه‌ها و به تَبعِ آن، دلایل شکستِ پروژه‌های نرم‌افزاری را معرفی خواهیم نمود. سپس، با توجه به تجربه‌ی پروژه‌های موفق، چندین راهکار و الگوی موفق برای تولید فراورده‌های نرم‌افزاری، اِرائه می‌گردد. در اِنتهای فصل، آر.یو.پی را به عنوان فرایند یا به عبارت بهتر، چارچوبِ فرایندی که در آن راهکارهای موفق پیاده‌سازی شده است، معرفی خواهیم نمود.

آگاهی از راهکارهای موفق، بدون داشتن بینش و اطلاع نسبت به عواملِ و ریشه‌های شکست، چندان مفید نخواهد بود. لذا، پیش از بررسی راهکارها و اصولِ موفق در دنیای مهندسی نرم‌افزار، نشانه‌ها و دلایل شکست را باید به دقّت مرور نمود. بر همین اساس، روندِ ارائه‌ی بحث‌ها در این فصل، همانند معالجه‌ی یک بیمار پیش می‌رود. در اینجا، بیماران ما پروژه‌های نرم‌افزاری و بیماری‌شان، شکست و عدم موفقیت است. برای تشخیص و درمانِ مؤثرِ بیماری، ابتدا باید نشانه‌ها و علائمِ بیماری را بررسی نموده و سپس با توجه به این نشانه‌ها، در پی دلایل ریشه‌ای آن باشیم. این کار ممکن است با بررسی نمونه‌های آماری مختلف انجام شود. در نهایت با توجه به تجربه‌ی موفق دیگران، راهکار مناسب یا درمان مناسبی را تجویز می‌نماییم.

بررسی نشانه‌ها و دلایل شکست

پروژه‌های مختلفْ به دلایل متنوعی با شکست مواجه می‌شوند. بررسی‌های انجام شده، نشان می‌دهد که نشانه‌ها و علائم مشترکی میان پروژه‌های ناموفق وجود دارد. برخی از مهم‌ترین نشانه‌های مشترکِ شکست و عدمِ موفقیت میان پروژه‌های ناموفق، عبارتست از:

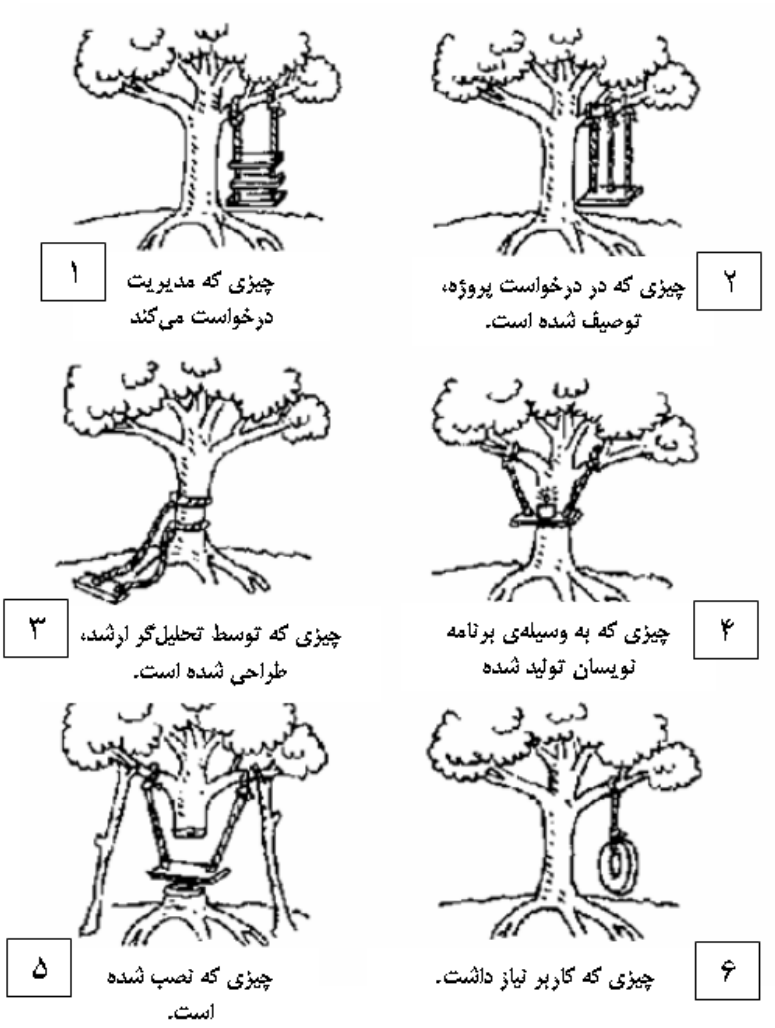
- عدم درک صحیح از نیازهای واقعیِ کاربران
- عدم توانایی در رویارویی با نیازمندی‌های در حال تغییر
- داشتن ماجول‌ها یا پیمانه‌هایی که با هم سازگاری ندارند
- داشتن نرم‌افزاری که نگهداری و گسترش آن سخت است
- کشف دیرهنگامِ نقایص و مشکلات جدی پروژه
- کیفیت پایین و نامطلوبِ فراورده‌ی نرم‌افزاری
- کارایی غیرِ مطلوب نرم‌افزار
- عدمِ توانایی در هماهنگی کارهای تیم
- داشتن یک فرایندِ نامطمئن

و بسیاری نشانه‌های دیگر، مانندِ طول‌کشیدنِ بیش از اندازه‌ی پروژه، افزایش هزینه‌های از قبل پیش‌بینی شده، کنار گذاشته شدنِ زودهنگامِ سیستم‌های نرم‌افزاری، و مانند آن، که به مراتب با آنها برخورد داشته‌ایم.

متأسفانه، صرفاً توجه به این نشانه‌ها و سعی در برطرف کردن آنها نمی‌تواند بیماری را بهبود بخشد؛ همان‌طور که با گذاشتن یک دستمالِ خیس روی بدن کسی که تب دارد، ممکن است کمی از تب او کم شود، ولی بیماری‌اش بهبود نمی‌یابد!

شکل ۲-۱

بدون شرح!



شناسایی دیرهنگامِ نواقص پروژه، تنها نشانه‌ای است از وجود مشکلات بسیار بزرگ‌تری مانندِ سنجش نامناسبِ وضعیتِ پروژه (که عمدتاً ناشی از بررسی بر اساسِ اِظهار نظرهای شخصی است) و نیز عدم شناسایی ناسازگاری‌های میان نیازمندی‌ها، طراحی‌ها، و پیاده‌سازی‌های انجام شده.

شکل ۲-۲

بدون شرح

بررسی پروژه‌های ناموفق، نشان داده است که دلایل ریشه‌ای مشکلات نیز به رغم تنوعِ بسیار زیادِ پروژه‌ها، در میان بسیاری از پروژه‌های ناموفق، مشترک می‌باشد. بر این اساس، مهم‌ترین دلایلِ عدمِ موفقیتِ پروژه‌های نرم‌افزاری عبارتند از:

- نداشتن یک روش اصولی و مشخص برای مدیریتِ نیازمندی‌ها
- وجود ارتباطات مبهم و نادقیق
- معماری‌های شکننده
- پیچیدگی روز افزون[1] و عدم توانایی در مدیریت آن
- عدم شناسایی ناسازگاری میان نیازمندی‌ها، طراحی‌ها، و پیاده‌سازی‌ها
- عدم انجامِ تست به اندازه‌ی کافی
- ارزیابی وضعیت پروژه‌ها با اظهار نظر شخصی، بدون داشتن معیارهای کمّی مناسب
- عدم توانایی در غلبه بر ریسک‌های پروژه
- پخش کنترل نشده‌ی تغییرات
- عدم استفاده از ابزارها و اتوماسیون به اندازه‌ی کافی

[1] - Overwhelming Complexity

دلایلی که اشاره شد، از مهم‌ترین دلایل ریشه‌ای و مشترک میان پروژه‌های ناموفق می‌باشد. مسلماً، شما می‌توانید دلایل و عواملِ بیشتری را ذکر نمایید. البته، بسیاری از دلایل دیگر به نحوی با دلایل ریشه‌ای بیان شده ارتباط دارند.

راهکارهای موفق[1]

بررسی و مشاهده‌ی پروژه‌های موفق، نشان داده است که این پروژه‌ها از راهکارها و استراتژی‌های خاصی برای از میان برداشتن دلایل و عواملِ بروزِ مشکلات استفاده نموده‌اند. نکته‌ی جالب اینکه بسیاری از راهکارهای بکارگرفته شده به وسیله‌ی پروژه‌های موفق، راهکارهای مشترکی می‌باشد؛ یعنی برای موفق شدن هم فرمول‌هایی وجود دارد که با از میان برداشتن و درمان دلایل ریشه‌ای شکست، نه تنها نشانه‌های عدم موفقیت را از بین می‌برند، بلکه ما را قادر می‌سازند که بتوانیم فراورده‌هایی با کیفیت مطلوب‌تر را به روشی قابل پیش‌بینی و قابل تکرار، تولید نماییم.

در اینجا منظورِ ما از راهکارها یا هنجارهای موفق، رویکردها و تجارب اثبات شده‌ی مهندسی در صنعت نرم‌افزار می‌باشد؛ هنگامی‌که این راهکارها را در یک پروژه استفاده نماییم، دلایل و عواملِ ریشه‌ای مشکلاتِ تولید، از بین خواهند رفت. در واقع، یک راهکار یا هنجار موفق عبارتست از مجموعه‌ای از اصول، روش‌ها، و فرایندهای به خوبی سازمان‌دهی و مستندسازی شده که موجبات بالا رفتن کیفیت و بهره‌وری تولید نرم‌افزار را فراهم می‌نمایند.

[1] - Best Practices

بنابراین، راهکارهای موفق نتیجه‌ی تجربه‌ی سازمان‌ها و پروژه‌های موفق می‌باشند. در این فصل، شش راهکارِ موفق را که بسیاری از راهکارهای موفق دیگر را در بر می‌گیرند، معرفی خواهیم نمود. این راهکارها عبارتند از:

۱. تولید و توسعه‌ی نرم‌افزار به روشی تکرارشونده[۱]
۲. مدیریت نیازمندی‌ها[۲] به روشی سیستماتیک و اصولی
۳. بهره‌گیری از معماری‌های مبتنی بر مؤلفه[۳]
۴. مدل‌سازی بصری[۴]
۵. ارزیابی مستمرِ کیفیتِ نرم‌افزار[۵]
۶. کنترلِ تغییراتِ[۶] قابل اِعمال بر سیستم

در فصل‌های آتی، خواهیم دید که تجارب و راهکارهای موفق در قالبِ آر.یو.پی پیاده‌سازی شده‌اند و در واقع از یک منظر، آر.یو.پی گنجینه‌ی ارزشمندی از تجارب و راهکارهای موفق را در اختیار ما قرار می‌دهد. در ادامه‌ی این فصل، هر یک از راهکارهای موفق اشاره شده را تشریح خواهیم نمود.

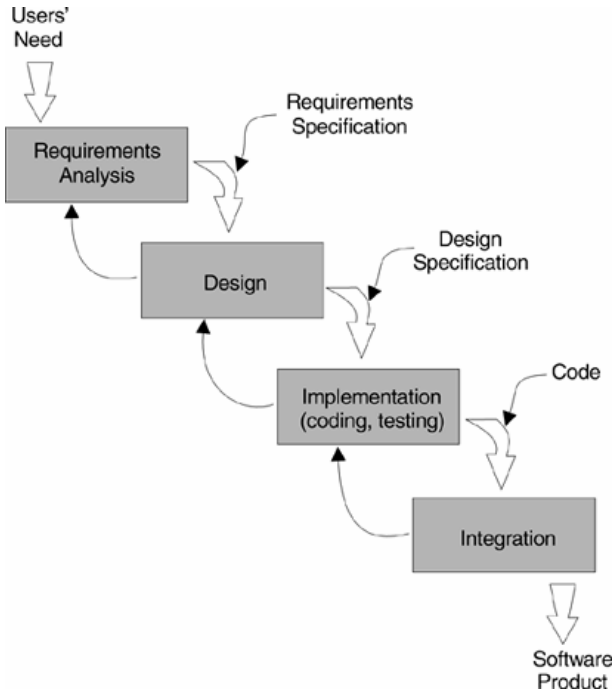
[1] - Iterative Development
[2] - Requirement Management
[3] - Component-based Architectures
[4] - Visual Modeling
[5] - Continuously Verify Quality
[6] - Control Changes

راهکار موفق ۱ – توسعه و تولید با رویکردِ تکرارشونده

رویکردِ سنتی فرایندِ تولیدِ نرم‌افزار، چرخه‌ی توسعه یا تولید با رویکردِ آبشاری[1] می‌باشد. در این رویکرد که مهندسی به صورت متوالی[2] نیز نام دارد، یک حرکت خطی از نیازمندی‌ها به تحلیل، طراحی، برنامه‌نویسی، تستِ اجزاء، تست زیرسیستم‌ها، و در نهایت به سوی تست و یکپارچه‌سازی کاملِ سیستم وجود دارد.

شکل ۲-۳

رویکردِ آبشاری



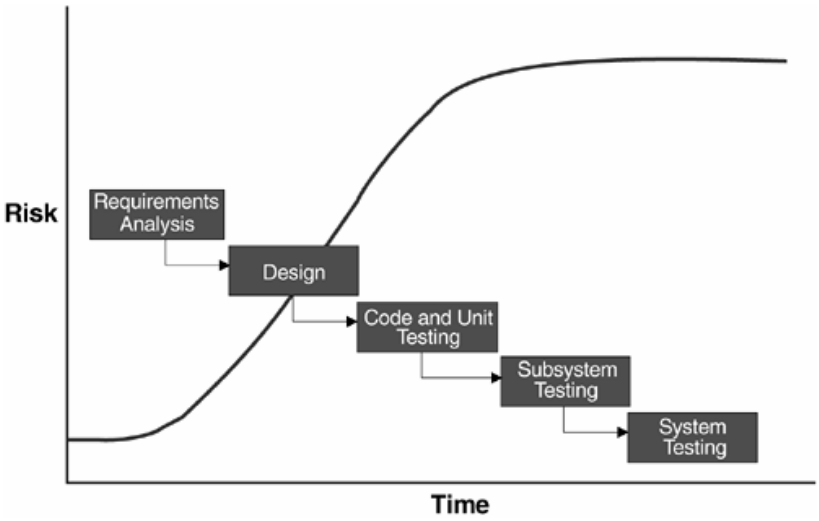
مهم‌ترین مشکلِ رویکرد آبشاری، ضعفِ ذاتی آن در غلبه بر ریسک[3] می‌باشد. در اینجا منظور از ریسک، کلیه‌ی شرایط، عوامل، و نگرانی‌هایی است که ممکن است مانعِ موفقیت شوند. بسیاری از ریسک‌ها تنها با پیاده‌سازی، تست، و یکپارچه‌سازی سیستم آشکار می‌شوند. از آنجایی که در رویکرد آبشاری، پیاده‌سازی، تست، و یکپارچه‌سازی سیستم به انتهای پروژه موکول می‌شود، بنابراین، در صورتِ آشکار شدن یک ریسک،

[1] - Waterfall
[2] - Sequential Engineering
[3] - Risk

فرصت کمی برای مدیریت آن وجود خواهد داشت و اغلب هزینه‌های زیادی برای مقابله با آن باید صرف نمود.

شکل ۲-۴

افزایشِ هزینه‌ی مدیریتِ ریسک در طول زمان



برای مثال، نقصِ یک طراحی ممکن است از نقصِ یک نیازمندی[1] متناظر با آن ناشی شده باشد. این مشکل تنها در زمان پیاده‌سازی و تست آشکار می‌شود، یعنی زمانی که ممکن است تصحیح آن باعث افزایش هزینه‌ها و طولانی‌تر شدن پروژه و یا حتی بسته شدن و شکست کامل آن شود.

ضعفِ رویکرد آبشاری در مواجهه با ریسک‌های عمده در پروژه‌های امروزی، مهم‌ترین عاملِ منسوخ شدنِ این رویکرد در دنیای مهندسی نرم‌افزار است. رویکردِ آبشاری، تنها برای پروژه‌هایی که ریسک‌های آن به خوبی شناخته شده است، مناسب است؛ برای مثال، در بسیاری از پروژه‌های ساختمان‌سازی، ریسک‌ها از قبل مشخص می‌باشد و بنابراین فرایندِ پروژه را می‌توان بر اساسِ رویکردِ آبشاری بنا نهاد. امّا در دنیای نرم‌افزار چنین پروژه‌هایی به ندرت وجود دارد؛ به گونه‌ای که بسیاری از متخصصان در دنیای نرم‌افزار بر این عقیده‌اند که تمامِ پروژه‌هایی که ریسک نداشتند، قبلاً توسط دیگران انجام شده‌اند!

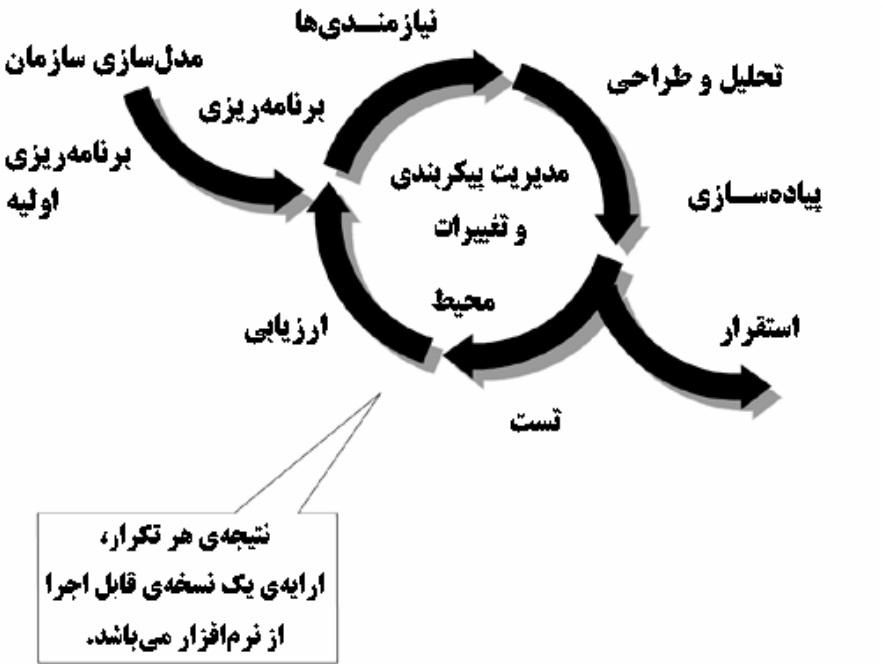
[1] - Requirement

در طول سال‌های گذشته، بسیاری از افراد در دانشگاه‌ها و نیز شرکت‌های پیشروی صنعت نرم‌افزار، تلاش زیادی برای ارائه و معرفی رویکردهای دیگری که جایگزینِ رویکردِ منسوخِ آبشاری باشد، انجام داده‌اند. حاصل این تلاش‌ها، ارائه‌ی ده‌ها رویکرد و فرایندِ دیگر بوده است. از جمله: روش حلزونی[1]، تولیدِ پیش‌الگو[2]، RAD ، SCRUM ، DSDM.

یکی از راهکارها و تجارب موفق، رویکردی است تحتِ عنوان تکرارشونده[3]. این روش که عمدتاً بر اساسِ مدلِ حلزونی[4] ایجاد شده است، توانایی قابل توجهی در مدیریت ریسک دارد. شکل ۲–۵ ، الگوی یک تکرار را که پایه و مبنای رویکرد تکرار شونده می‌باشد، نشان می‌دهد. شکل ۲–۶ ، بیانگر شِمایی کلی از فرایندِ مبتنی بر رویکردِ حلزونی می‌باشد.

شکل ۲–۵

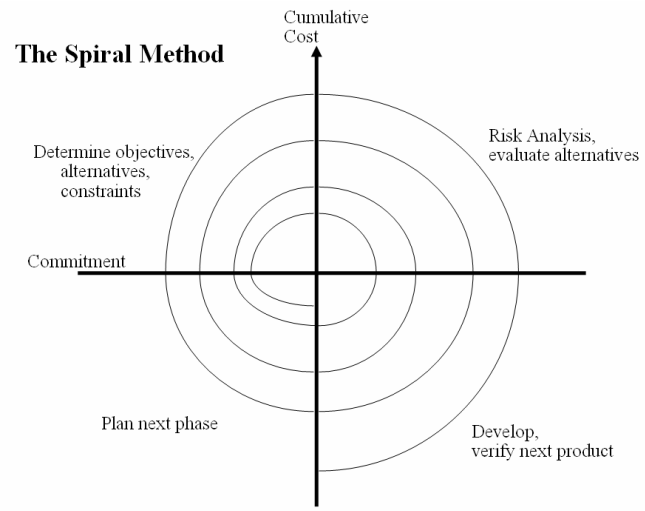
یک تکرار، توالی مجموعه‌ی فعالیت‌های فرایند در یک بازه‌ی زمانی کوتاه



[1] - Spiral
[2] - Prototyping
[3] - Iterative
[4] - Spiral

شکل ۲-۶

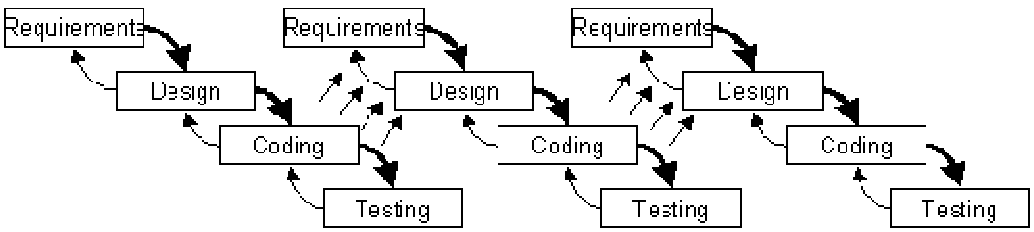
روشِ حلزونی، مبنای رویکردِ تکرارشونده



در واقع، یک تکرار[1] عبارتست از انجام متوالی مجموعه فعالیت‌های لازم در یک بازه‌ی زمانی کوتاه و انجام چندین باره‌ی آن در طول بازه‌ی زمانی یک پروژه. بنابراین، به جای انجام یک بار و به صورت متوالی فعالیت‌ها، چندین بار و در بازه‌های کوچکتری این مجموعه فعالیت‌ها را تکرار می‌نماییم. همان‌گونه که در شکل ۲-۷ نشان داده شده است، در فرایندی مبتنی بر رویکرد تکرارشونده، چندین تکرار از تقریباً تمام مجموعه فعالیت‌های لازم (و البته با تأکید متفاوت نسب به مقاطع زمانی در طول پروژه) وجود دارد.

شکل ۲-۷

شِمای کلی از فرایندِ مبتنی بر رویکردِ تکرارشونده

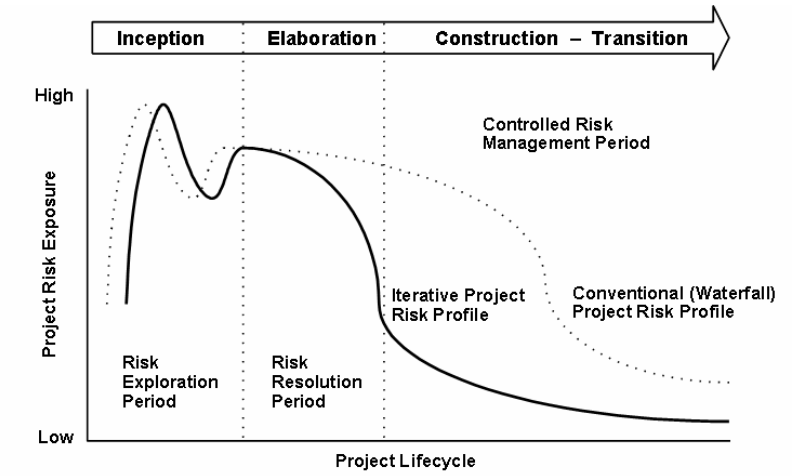


در شکلِ ۲-۸ ، پروفایلِ ریسک در رویکردهای تکرارشونده و آبشاری، مقایسه شده است.

[1] - Iteration

شکل ۲-۸

مقایسه‌ی میان پروفایلِ ریسک در رویکردهای آبشاری و تکرارشونده



رویکردِ تکرارشونده یکی از مهم‌ترین راهکارهای موفق در صنعت نرم‌افزار می‌باشد. اصول و مبنای اصلی فرآیندهای به اصطلاح چابُک[1] نیز بر مبنای همین رویکرد بنا شده است. در ادامه خواهیم دید که این رویکرد، تأثیر بسیار زیادی بر سایر راهکارهای موفق دارد.

تولیدِ نرم‌افزار با کمکِ رویکردِ تکرارشونده، نقشِ مؤثری در از میان برداشتن برخی از دلایل و عواملِ ریشه‌ای مشکلات در پروژه‌های نرم‌افزاری دارد. از جمله‌ی این تأثیرات، می‌توان موارد زیر را بیان نمود:

- امکان دریافتِ بازخوردهای[2] مستمر از کاربران و به تبعِ آن بهبود مدیریتِ نیازمندی‌های[3] و اطمینان از جلب مشارکت مشتری و کاربر در فرایند تولید
- امکانِ شناسایی زودهنگامِ مشکلات و نواقصِ کلیدی در همان تکرارهای ابتدای پروژه فراهم شده و می‌توان با هزینه‌ی کمتری نسبت به رفع آنها اقدام نمود.
- فراهم شدن اِمکان تمرکز بیشترِ تیم بر روی مسائل مهم‌تر و حساس‌تر پروژه که در ارتباط با ریسک‌های عمده‌ی پروژه می‌باشد.

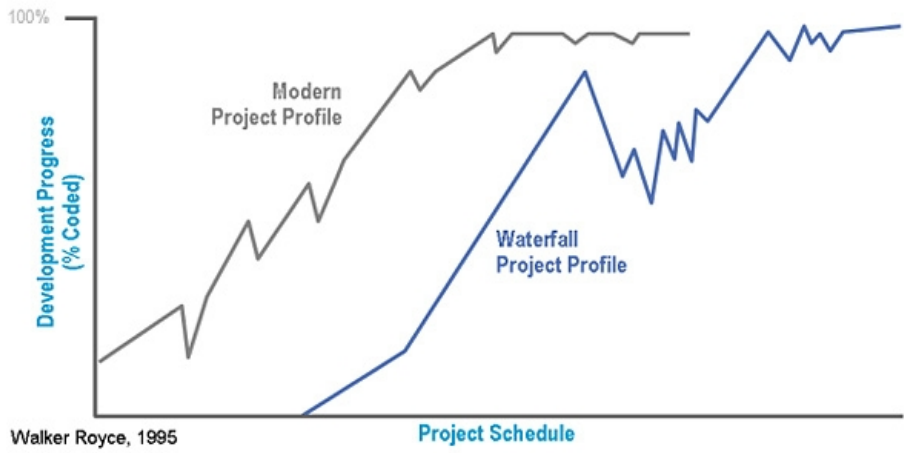
[1] - Agile
[2] - Feedback
[3] - Requirements

- امکان انجام تست‌های مکرّر و مستمر فراهم می‌شود و بنابراین معیارهایی کمّی[1] و قابل سنجشی برای بررسی وضعیت پروژه در اختیار مدیران تیم قرار می‌گیرد.
- ناسازگاری میان نیازمندی‌ها، طراحی‌ها، و پیاده‌سازی‌ها، زودتر قابل کشف می‌باشد.
- بارِ کاری تیم، خصوصاً بارِ کاری مربوط به مسئولین انجام تست در طول فرایند و چرخه‌ی تولید تقسیم می‌شود.
- تیم قادر خواهد بود که از تجارب خود درس گرفته و به طور مستمر به بهبود فرایند اقدام نماید.
- همه‌ی ذینفعان[2] پروژه در طول چرخه‌ی تولیدِ فراورده، در جریانِ وضعیتِ پیشرفت کار قرار می‌گیرند.

شکل ۲-۹ ، مقایسه‌ای است میان پروفایلِ پروژه‌هایی که در گذشته با رویکردِ آبشاری انجام می‌شدند و پروفایلِ پروژه‌های نوین که فرایندِ تولیدشان به صورت تکرارشونده است.

شکل ۲-۹

مقایسه‌ی میان پروفایلِ پروژه‌ها در رویکردهای آبشاری و تکرارشونده



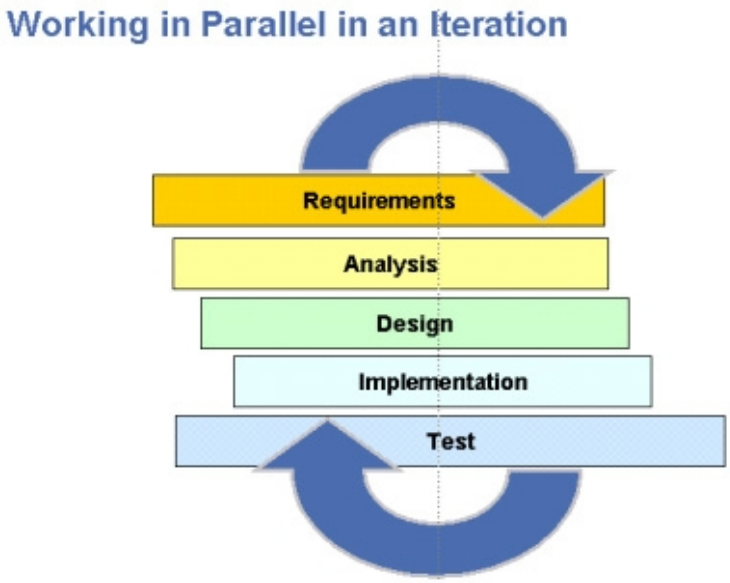
جالب است توجه داشته باشیم که در رویکردِ تکرارشونده[3]، مجموعه‌ی فعالیت‌های فرایندِ تولید، به طور موازی قابل انجام می‌باشند. شکل ۲-۱۰ ، یک تکرار[4] را که در آن مجموعه‌ی فعالیت‌های مختلفِ فرایند، به

[1] - Objective vs. Subjective
[2] - Stakeholders
[3] - Iterative
[4] - Iteration

طور موازی اجرا می‌شوند. نشان می‌دهد. هر یک از این مجموعه فعالیت‌ها را که در رویکردِ آبشاری فاز نامیده می‌شدند، دیسیپلین[1] می‌نامند. در فصل‌های آینده مفهومِ دیسیپلین را تشریح خواهیم نمود.

شکل ۲-۱۰

اجرای موازی دیسیپلین‌ها در یک تکرار



[1] - Discipline

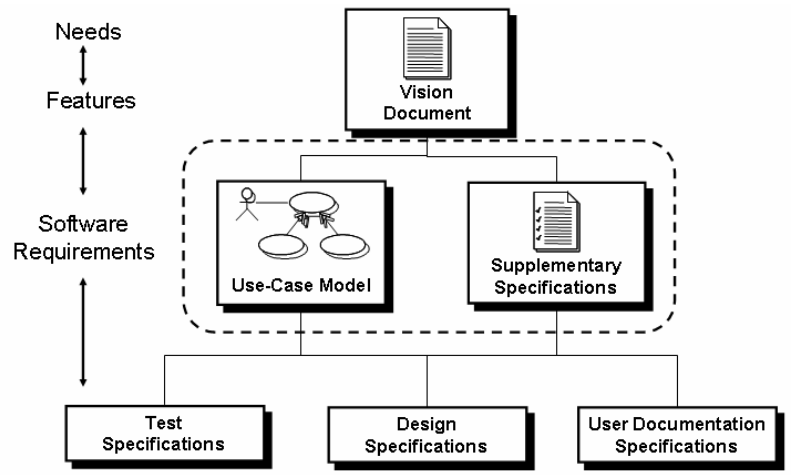
راهکار موفق ۲ – مدیریتِ نیازمندی‌ها[1]

مهم‌ترین مسأله در مدیریتِ نیازمندی‌های یک سیستمِ نرم‌افزاری، این است که نیازمندی‌ها دائماً در حال تغییر بوده و موجودیتی پویا می‌باشند. بنابراین، باید انتظار داشته باشیم که در طولِ عمرِ یک پروژه‌ی نرم‌افزاری، شاهد تغییرات وسیعی در نیازمندی‌ها باشیم. علاوه بر این، شناسایی نیازمندی‌های واقعی سیستم، یک فرایندِ مستمر می‌باشد. به جز در پروژه‌های تولیدِ سیستم‌های بسیار خاص که بارها و بارها نمونه‌ی مشابه‌شان تولید شده است، نیازمندی‌های یک سیستم را نمی‌توان قبل از شروعِ فرایندِ تولید و اقدام به پیاده‌سازی آن، به طور کامل و جامع بیان نمود.

یک نیازمندی[2]، شرایط و یا قابلیتی است که سیستم باید دارا باشد. به طور کلی، نیازمندی‌های نرم‌افزاری به دو دسته‌ی نیازمندی‌های کارکِردی[3] (وظیفه‌مندی) و نیازمندی‌های غیرِ وظیفه‌مندی[4] تقسیم‌بندی می‌شوند. شکل ۲–۱۱ ، بیانگر این تقسیم‌بندی می‌باشد.

شکل ۲– ۱۱

انواعِ نیازمندی‌ها

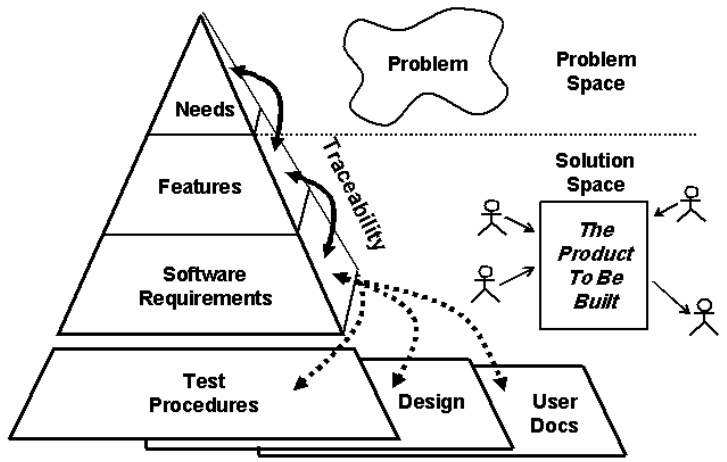


[1] - Requirement Management
[2] - Requirement
[3] - Functional
[4] - Non-Functional

شکل ۲-۱۲ ، مدلی از ارتباط میان نیازهای تعریف شده در دنیای صورت مسأله، ویژگی‌های کلیدی راه‌حل، و نیازمندی‌های نرم‌افزار را نشان می‌دهد.

شکل ۲-۱۲

ارتباطِ میان نیازها، ویژگی‌های مطلوب، و نیازمندی‌ها



مدیریتِ نیازمندی‌ها به عنوان یک روشِ سیستماتیک[1]، برای جمع‌آوری[2]، سازماندهی[3]، و مستندسازی[4] مجموعه‌ی نیازمندی‌های نرم‌افزار، ارزیابی تغییرات و تأثیرات آنها، ردگیری و مستندسازی انتخاب‌های موجود[5]، و تصمیمات مربوطه، مطرح می‌باشد.

مدیریتِ نیازمندی‌ها، راهکارهایی برای رفع دلایل و عواملِ ریشه‌ای مشکلاتِ پروژه‌های نرم‌افزاری فراهم می‌نماید. از جمله، می‌توان به مواردِ ذیل اشاره نمود:

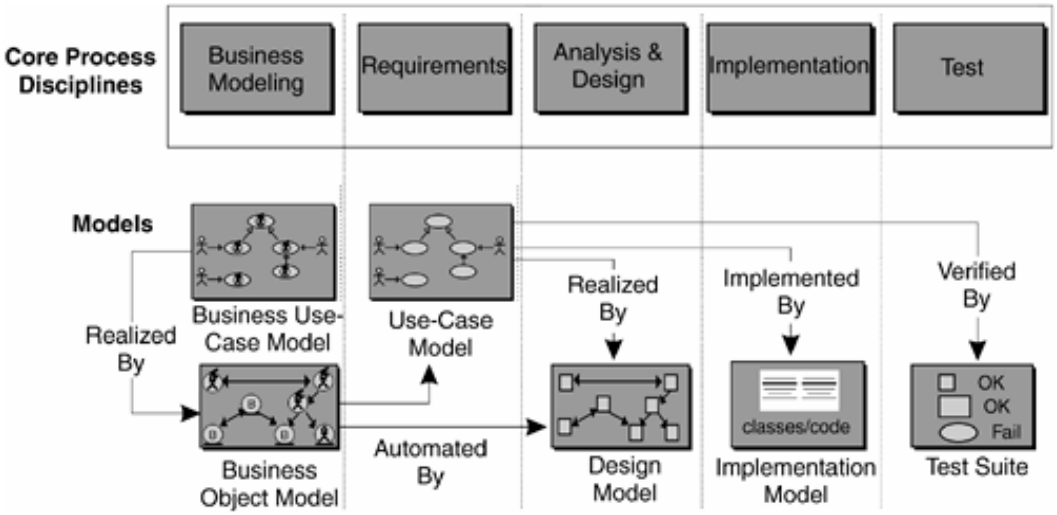
- یک رویکردِ منظم و دارای دیسیپلین برای مدیریتِ نیازمندی‌ها ایجاد می‌شود.
- ارتباطات بر اساس نیازمندی‌های تعریف شده، صورت می‌پذیرد.
- نیازمندی‌ها را می‌توان اولویت‌بندی، فیلتر، و نیز ردگیری نمود.
- ارزیابی مقصودگرا[1] و کمّی کارکردها و کارایی، امکان‌پذیر می‌گردد.

[1] - Systematic
[2] - Eliciting
[3] - Organizing
[4] - Documenting
[5] - Trade-offs

- ناسازگاری‌ها را راحت‌تر می‌توان تشخیص داد.
- با داشتن ابزارهای مناسب، مخزنی از نیازمندی‌ها، ویژگی‌ها، و ارتباط میان آنها در پروژه فراهم می‌شود.

شکل ۲-۱۳

ارتباط میانِ مدلِ نیازمندی‌ها (مواردِ کاربرد) و سایر مدل‌ها در فرایندِ تولید



[1] - Objective

راهکار موفق ۳ - بهره‌گیری از معماری‌های مبتنی بر مؤلفه

توصیفِ ساخت و مستندسازی یک سیستمِ نرم‌افزاری، مستلزمِ داشتنِ منظرهای مختلفی از سیستم و از جنبه‌های مختلف می‌باشد. هر یک از ذینفعان[۱]، اعم از کاربر[۲]، مشتری[۳]، تحلیل‌گر، و مدیر پروژه، نگاهِ خاصی به پروژه دارند. معماری نرم‌افزار، مهم‌ترین چیزی است که به منظور مدیریت منظرها و انتظاراتِ مختلف ذینفعان استفاده می‌شود و به کمکِ آن، توسعه‌ی مبتنی بر رویکردِ تکرارشونده و تکاملِ تدریجی سیستم در طول چرخه‌ی تولید، کنترل می‌شود.

معماری نرم‌افزار، موجودیتی است بیانگرِ تصمیم‌گیری‌های کلیدی درباره‌ی ملاحظات مختلفِ سیستم، از جمله:

- چگونگی سازماندهی اجزای سیستم
- انتخابِ اجزای ساختاری و واسط میانِ این اجزاء
- رفتارِ سیستم در قالبِ همکاری میان اجزاء و عناصرِ ساختاری
- شیوه‌ی معماری[۴] که راهنمای چگونگی سازماندهی می‌باشد

معماری نرم‌افزار، نه تنها به ملاحظات مرتبط با ساختار و رفتار توجه دارد، بلکه با ملاحظاتی نظیرِ کاربرد[۵]، وظیفه‌مندی[۶]، کارایی[۷]، قابلیتِ انعطاف[۸]، استفاده‌ی مجدد[۹]، ملاحظاتِ اقتصادی، محدودیت‌ها و استانداردهای فنی و تکنولوژیکی نیز در ارتباط می‌باشد. ایجادِ معماری‌های قابل انعطاف، اهمیّت بسیاری دارد. چنین معماری امکانِ استفاده‌ی مجدد و مقرون به صرفه از اجزاء را فراهم نموده و موجباتِ تسهیلِ مدیریتِ تغییرات و نگهداری سیستم را فراهم می‌آورد.

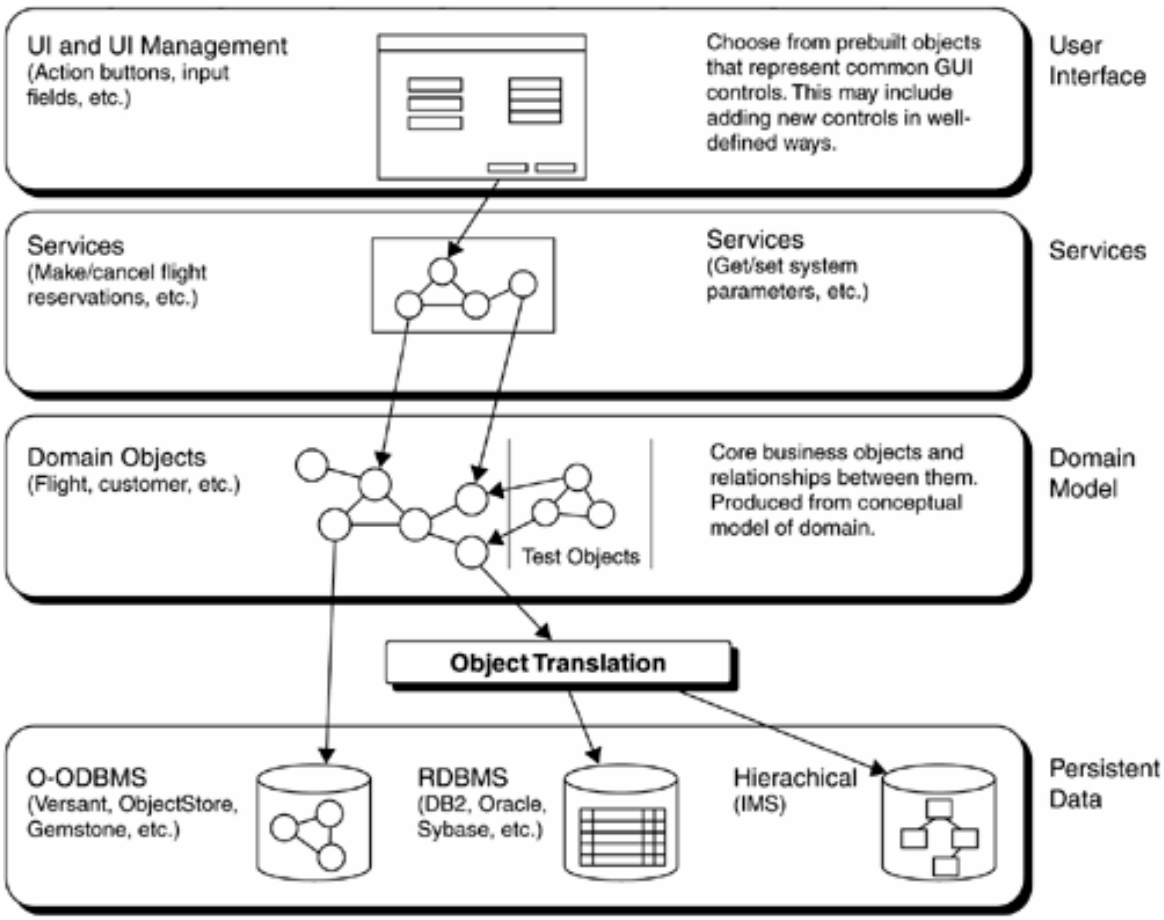
---

[1] - Stakeholders
[2] - User
[3] - Customer
[4] - Style
[5] - Usage
[6] - Functionality
[7] - Performance
[8] - Resilience
[9] - Reuse

تولید مبتنی بر مؤلفه[1]، رویکرد مهمی در توسعه‌ی معماری نرم‌افزار می‌باشد. با استفاده از این رویکرد که یک تجربه و راهکار موفق در دنیای مهندسی می‌باشد، قابلیتِ استفاده‌ی مجدد و نیز سفارشی‌سازی مؤلفه‌ها تسهیل می‌شود. فناوری‌های مختلفی مانند COM ، CORBA ، و EJB ، برای ایجادِ مؤلفه‌های نرم‌افزاری بکار می‌روند.

شکل ۲-۱۴

نمونه‌ای از یک معماری مبتنی بر مؤلفه



بهره‌گیری از معماری مبتنی بر مؤلفه در ترکیب با رویکردِ تکرارشونده[2]، امکانِ تکاملِ مستمرِ معماری سیستم را در طولِ چرخه‌ی تولید[3] فراهم می‌آورد. در هر تکرار[4]، یک نسخه اجرایی از سیستم و زیرساخت‌های

[1] - Component-Based Development
[2] - Iterative Development
[3] - Development Cycle
[4] - Iteration

معماری آن، تولید می‌شود که قابل سنجش، تست، و ارزیابی نسبت به نیازمندی‌های نرم‌افزار می‌باشد. بدین ترتیب، کارشناسانِ تیمِ تولید قادر خواهند بود در کمترین زمانِ ممکن، به ریسک‌های عمده‌ی پروژه غلبه نمایند.

استفاده از معماری‌های مبتنی بر مؤلفه، راهکارهایی برای از بین بردن برخی از دلایل و عواملِ ریشه‌ای مشکلاتِ تولید نرم‌افزار، فراهم می‌نماید. از جمله:

- یک معماری مبتنی بر مؤلفه، قابلیت انعطاف بیشتری دارد.
- معماری مبتنی بر مؤلفه با تفکیک ملاحظات و نگرانی‌های مختلف در میان عناصر و مؤلفه‌های مختلفِ سیستم، مدیریتِ تغییرات را تسهیل می‌نمایند.
- امکانِ استفاده‌ی مجدد و بهره‌گیری از چارچوب‌های استانداردی مانند CORBA ، +COM ، EJB ، و مؤلفه‌های آماده[1] فراهم می‌شود.
- مؤلفه‌ها، مبنایی برای مدیریتِ پیکربندی فراهم می‌نمایند.
- ابزارهای مدل‌سازی بصری به خودکارسازی تولیدِ مبتنی بر مؤلفه‌ی سیستم، کمک می‌کنند.

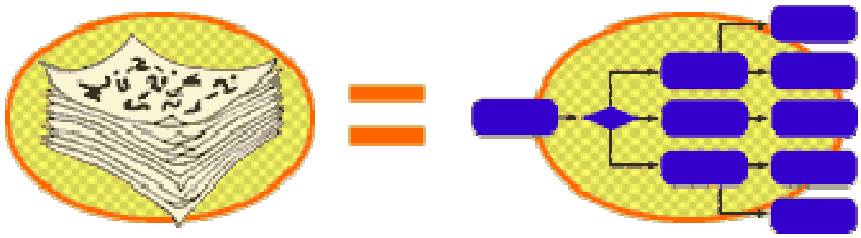
[1] - COTS: Commercial-Off-The-Shelf

راهکار موفق ۴ – مدل‌سازی بصری[1]

بررسی پروژه‌های موفق نشان می‌دهد که یکی از مهم‌ترین دلایل و عواملِ موفقیت پروژه، مدل‌سازی بصری (تصویری) می‌باشد. ارائه‌ی مدل‌های بصری از جنبه‌های مختلفِ سیستم، می‌تواند ما را در درک بهتر سیستم، غلبه بر پیچیدگی، و برقراری ارتباطات مؤثر، یاری داده و موجبات رفعِ بسیاری از دلایل و ریشه‌های بروزِ شکست و عدمِ موفقیت پروژه را فراهم نماید.

شکل ۲-۱۵

مدل‌سازی بصری در مقابلِ مستنداتِ متنی



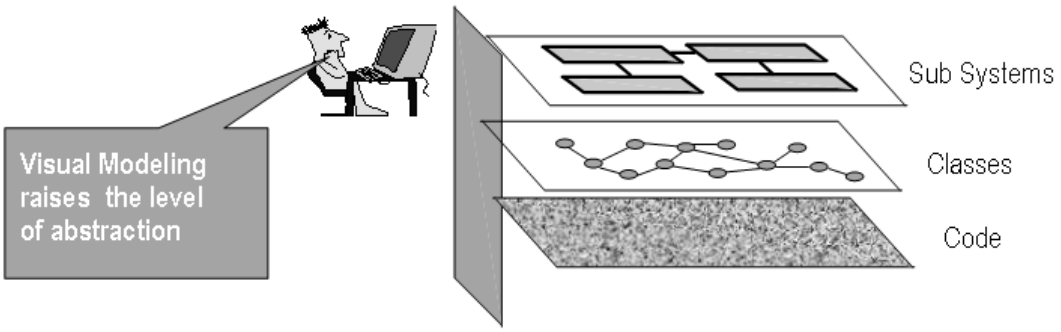
مدل، بازنمایی ساده‌شده‌ای از یک واقعیتِ پیچیده و توصیف‌گرِ سیستم از یک منظر خاص می‌باشد. مدل‌سازی با هدف غلبه بر پیچیدگی، تسهیل ارتباطات، و نیز مستندسازی انجام می‌شود. مدل‌سازی به شیوه‌های مختلفی می‌تواند انجام شود، اما تجربه‌ی پروژه‌های موفق نشان داده است که مدل‌سازی بصری، راهکار مؤثر و موفقی در مدل‌سازی می‌باشد. به کمک مدل‌سازی بصری، مدیریتِ مدل‌های مختلف تسهیل شده و امکانِ کنترلِ میزان جزئیات در سطوح مختلف مدل‌ها فراهم می‌گردد.

حفظِ سازگاری میان دستاوردهای مختلف سیستم، یعنی نیازمندی‌ها، طراحی‌ها، و پیاده‌سازی به کمک مدلسازی بصری امکان‌پذیر است. در واقع، مدل‌سازی بصری، اعضای تیم را قادر می‌سازد که بتوانند پیچیدگی نرم‌افزار را مدیریت نمایند.

[1] - Visual Modeling

شکل ۲-۱۶

مدل‌سازی بصری و سطوحِ مختلفِ تجرید به عنوان یک ابزار و تکنیکِ مهندسی



مدل‌سازی بصری در کنارِ توسعه و تولید مبتنی بر رویکردِ تکرارشونده، امکان اِعمال تغییرات و ارزیابی اثر آنها روی معماری را فراهم می‌نماید. با بهره‌گیری از ابزارهای مناسبِ مدل‌سازی بصری، می‌توان مدل‌سازی و کدهای پیاده‌سازی سیستم را در هر تکرار، با هم همزمان[1] نمود.

مدل‌سازی بصری سیستم‌های نرم‌افزاری، راهکارهایی برای از میان برداشتنِ برخی دلایل ریشه‌ای مشکلات در پروژه‌های نرم‌افزاری فراهم می‌نماید، از جمله:

- کیفیتِ فراورده با یک طراحی خوب شروع می‌شود؛ داشتن یک طراحی مناسب، تنها با مدل‌سازی بصری امکان‌پذیر و مقرون به صرفه می‌باشد.
- طراحی‌های غیر مبهم، ناسازگاری‌های‌شان را زودتر آشکار می‌نمایند.
- جزئیات را می‌توان در صورتِ لزوم، پنهان نمود (مدیریت پیچیدگی).
- تشخیص یک معماری غیر پیمانه‌ای و غیر قابل انعطاف، آسان‌تر خواهد بود.
- مدل‌ها و خصوصاً مدل‌هایی که به زبان استاندارد مدل‌سازی، یعنی یو.اِم.اِل، ایجاد شوند، ارتباطات غیر مبهم را ترویج می‌دهد.

امروزه برای مدل‌سازی بصری، استانداردی تحتِ عنوان یو.اِم.اِل[1] وجود دارد. با کمک این استاندارد ضمنِ بهره‌گیری از مزایای مدل‌سازی بصری، مدل‌های ارائه شده در قالبِ یک زبان استاندارد، توصیف می‌شوند.

[1] - Synchronize

پیش از این، روش‌ها و تکنیک‌های مختلفی برای مدل‌سازی بصری مفاهیم استفاده می‌شد. برخی از این تکنیک‌ها مانند دی.اِف.دی[۲] ، ای.آر.دی[۳] ، اِس.تی.دی[۴] ، فلوچارت[۵]، پِتری‌نِت[۶]، و مانند آنها آشنا می‌باشند. برخی تکنیک‌های دیگر هم به طور محدود و در زمینه‌های خاصی استفاده می‌شد. در طی سال‌های گذشته، تعدّد تکنیک‌ها و روش‌های مدل‌سازی، عملاً خود به مشکلی اساسی تبدیل شده بود.

یو.اِم.اِل زبانی است که توسط کارشناسان دنیای مهندسی سیستم‌های نرم‌افزاری به منظور متحدالشکل[۷] کردن و یکی کردن تکنیک‌های مدل‌سازی ارائه گردید. در این زبان، نقاط قوت تکنیک‌ها و نمادگذاری‌های موفق در مدل‌سازی گردآوری شده است. البته، با توجه به اینکه امروزه دیگر دیدگاهِ ساختیافته[۸]، رویکردی مهندسی نمی‌باشد و رویکرد شیءگرا[۹] به عنوان راهکار مهندسی مطرح شده است، گرامر این زبان عمدتاً مبتنی بر مفاهیم شیءگراست.

از ویژگی‌های مهم زبان یو.اِم.اِل، قابلیت گسترش و کاربری آن در طیف وسیعی از موارد مدل‌سازی اعم از نرم‌افزارهای مختلف، سخت‌افزار، سازمان، پدیده‌های زیستی، ریاضیات، و به طور کلی مدل‌سازی انواع مختلفی از سیستم‌ها می‌باشد. این زبان توسط کنسرسیومی از شرکت‌های بزرگ تحت عنوان اُ.اِم.جی[۱۰] نگهداری و به‌روز می‌شود.

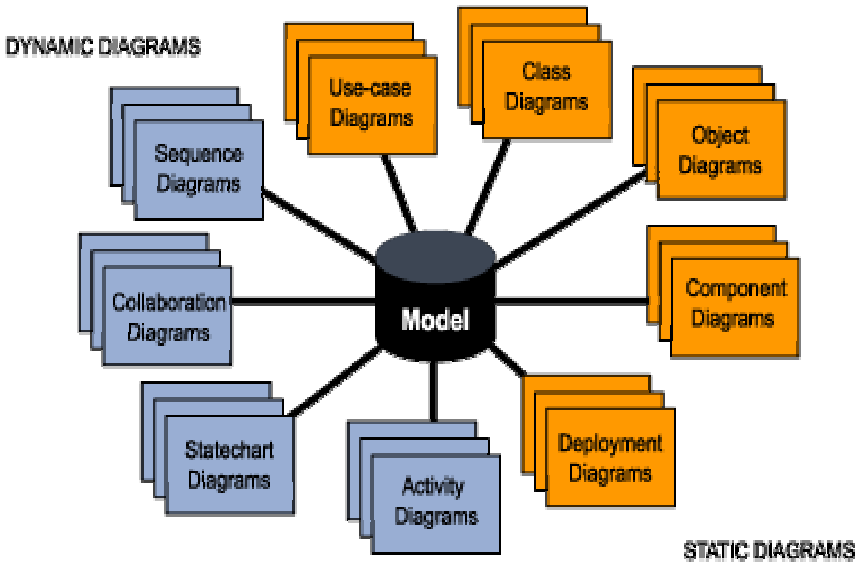
از منظر استفاده کننده، اجزاء کلیدی در این زبان، شامل تعدادی دیاگرام می‌باشد. در نسخه‌ی ۱/۴ از این زبان، نُه دیاگرام مختلف برای مدل‌سازی استفاده می‌شد. در نسخه‌ی ۲ ، تعداد دوازده دیاگرام برای مدل‌سازی حوزه‌ی گسترده‌تری از مفاهیم در دنیای سیستم‌ها ارائه شده است.

---

[1] - UML: Unified Modeling Language
[2] - DFD: Data Flow Diagram
[3] - ERD: Entity Relationship Diagram
[4] - STD: State Transition Diagram
[5] - Flowchart
[6] - Petri-Net
[7] - Unification
[8] - Structured Approach
[9] - Object Orientation
[10] - OMG: Object Management Group

شکل ۲-۱۷

دیاگرام‌های زبان مدل‌سازی یو.اِم.اِل



امروزه، یکی از مهم‌ترین رویکردهای تولید، رویکرد توسعه یا تولید بر مبنای مدل[۱] می‌باشد. دیاگرام‌های یو.اِم.اِل به منظور تصویرسازی[۲]، توصیف[۳]، ایجاد، و مستندسازی[۴] مفاهیم رفتاری و ساختاری در یک مدل بکار می‌روند. در یک فرایند مبتنی بر مدل و مدل‌سازی، مدل‌هایی نظیر مدلِ نیازمندی‌ها[۵]، مدلِ تحلیل[۶]، مدلِ طراحی[۷]، مدلِ پیاده‌سازی[۸]، و مدلِ تست وجود دارد.

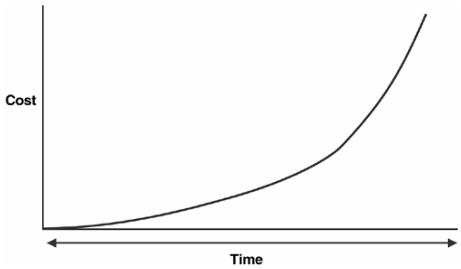
[1] - MDD: Model Driven Development
[2] - Visualization
[3] - Specification
[4] - Documentation
[5] - Requirements Model
[6] - Analysis Model
[7] - Design Model
[8] - Implementation Model

راهکار موفق ۵ – ارزیابی مستمرِ کیفیتِ نرم‌افزار[1]

مطابقِ شکل ۲–۱۸ ، هزینه‌ی بهبود و تصحیح مشکلات و نواقص یک سیستم نرم‌افزاری در طول زمان و با پیشرفتِ پروژه، به صورت نمایی زیاد می‌شود. به‌گونه‌ای که هزینه‌ی رفع یک خطا بعد از اینکه سیستم در محیط مشتری و کاربران نصب و راه‌اندازی گردید، در حدود صد تا هزار برابر بیشتر از هزینه‌ی رفع همین مشکل در اوایل فرایند تولید می‌باشد.

شکل ۲–۱۸

افزایشِ نمایی هزینه‌ی بر طرف نمودن خطاها و نواقص با گذشتِ زمان



بنابراین، لازم است که به صورت مستمر، کیفیتِ سیستم را از جنبه‌های مختلفی مانندِ کارکردها[2]، قابلیتِ اعتماد[3]، کارایی[4] نرم‌افزار، و کارایی سیستم، ارزیابی نماییم. تجربه‌ی پروژه‌های موفق، حاکی است که داشتن یک رویکرد مناسب برای بررسی و مدیریتِ کیفیت فراورده در طولِ چرخه‌ی تولیدِ آن، تأثیر بسیار زیادی بر موفقیت پروژه دارد. شکل ۲–۱۹ ، چگونگی استمرارِ تست را در طولِ تکرارهای مختلفِ فرایندِ تولید نشان می‌دهد.

همان‌گونه که احتمالاً حدث زده‌اید، رویکرد تکرارشونده مهمترین و بهترین راه‌حل را برای تحقق این راهکار موفق، یعنی ارزیابی مستمرِ کیفیت، فراهم می‌آورد. در واقع، ارزیابی مستمرِ کیفیت، مستلزم داشتن رویکردی تکرارشونده است که در طی آن فرصت انجامِ تست‌های مکرّر ایجاد می‌شود.

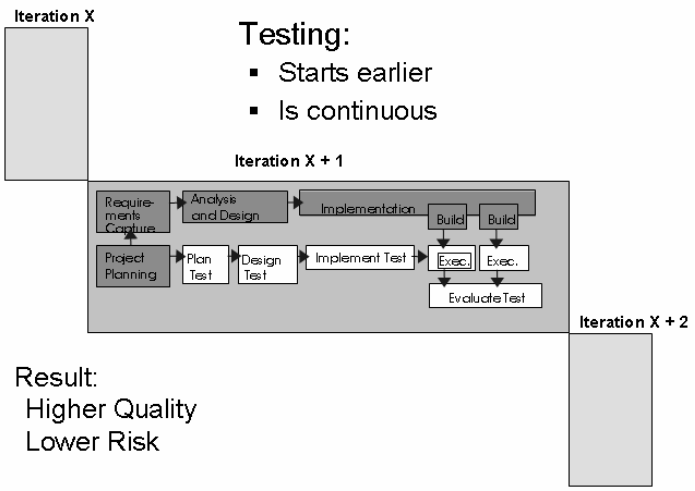
[1] - Continuously Verify Quality
[2] - Functionality
[3] - Reliability
[4] - Performance

ارزیابی مستمر کیفیت، راهکارهایی را برای رفعِ برخی از مشکلات و دلایلِ ریشه‌ای مرتبط با آن‌ها در پروژه‌های نرم‌افزاری فراهم می‌نماید، از جمله:

- امکانِ ارزیابی مقصودگرا[1] و کمّی وضعیتِ پروژه فراهم می‌گردد و بنابراین از اظهارنظرهای شخصی[2] فاصله خواهیم گرفت. در واقع نتایجِ انجام تست‌های مکرّر، معیارها و اعداد و ارقامِ مناسبی از پیشرفت یا عدم پیشرفت پروژه فراهم می‌نمایند.
- ناسازگاری میان نیازمندی‌ها، طراحی‌ها، و پیاده‌سازی‌ها آشکار می‌گردد.
- تست می‌تواند زودتر به نواحی با ریسک بالاتر و پر اهمیت‌تر متمرکز شده و در نتیجه، کیفیت آن نواحی با افزایش چشمگیری مواجه می‌شود.
- نواقص و خطاها، زودتر کشف شده و لذا هزینه‌های رفع آنها کمتر خواهد بود.
- ابزارهای اتوماسیونِ تست، امکانِ تست خودکارِ کارایی، قابلیت اعتماد، و نیز کارکردها و نیز امکانِ انجام تست‌های بیشتر را در طول چرخه‌ی تولید فراهم می‌نمایند.

شکل ۲-۱۹

استمرارِ تست در طولِ تکرارهای مختلفِ فرایندِ تولید



[1] - Objective
[2] - Subjective

راهکار موفق ۶. کنترل تغییرات[1]

یکی از مهم‌ترین معضلات و چالش‌های موجود در تولیدِ سیستم‌های نرم‌افزاری، عبارتست از وجود افراد مختلف در تیم‌های مختلف و احتمالاً در مکان‌های جغرافیایی متفاوت که با هم روی تکرارهای[2] مختلف، نسخه‌های[3] مختلف، فراورده‌ها، و بسترهای[4] مختلفی کار می‌کنند. در چنین شرایطی، اگر کنترلِ منظم و منطقی روی تغییرات و تصمیم‌گیری‌های مختلف وجود نداشته باشد، با نوعی آشوب و سردرگمی مواجه خواهیم شد.

هماهنگی تکرارها و نسخه‌های مختلف در تولید یک فراورده‌ی نرم‌افزاری، مستلزمِ کنترلِ تغییرات و پیکربندی[5] نرم‌افزار می‌باشد. تغییر[6]، مختلف ممکن است از منابع مختلفی ناشی شده باشد و اثرات متفاوتی نیز بر روندِ فعالیت‌ها و مهم‌تر از آن بر فراورده‌ی نهایی داشته باشد. عدم کنترلِ تغییرات، علت شکست بسیاری از پروژه‌ها و خصوصاً پروژه‌های پیچیده و بزرگ می‌باشد. بدون کنترلِ تغییرات، خصوصاً با بزرگ شدن ابعاد و پیچیده‌تر شدن پروژه، امکانِ انجام کار تیمی وجود نداشته و در نتیجه، پروژه با عدمِ موفقیت مواجه خواهد شد.

تجربه‌ی پروژه‌های موفق نشان می‌دهد که داشتن یک روشِ سیستماتیک و اصولی برای کنترلِ تغییرات، اثر بسزایی در موفقیت پروژه‌ها دارد. کنترلِ تغییراتِ نرم‌افزار، راهکارهایی را برای رفع برخی از دلایل و عواملِ ریشه‌ای مشکلاتِ پروژهای نرم‌افزاری، فراهم می‌نماید. از جمله:

- مدیریت نیازمندی‌ها بهتر و اصولی‌تر انجام می‌شود. مدیریت نیازمندی‌های متغیر، بدون کنترلِ تغییرات امکان‌پذیر نمی‌باشد. در واقع، تغییر مفهومی است جدایی ناپذیر از نیامندی‌ها.
- کنترل و مدیریت درخواست‌های تغییر[7]، موجبِ تسهیل ارتباطاتِ شفاف و بدون ابهام می‌شود.

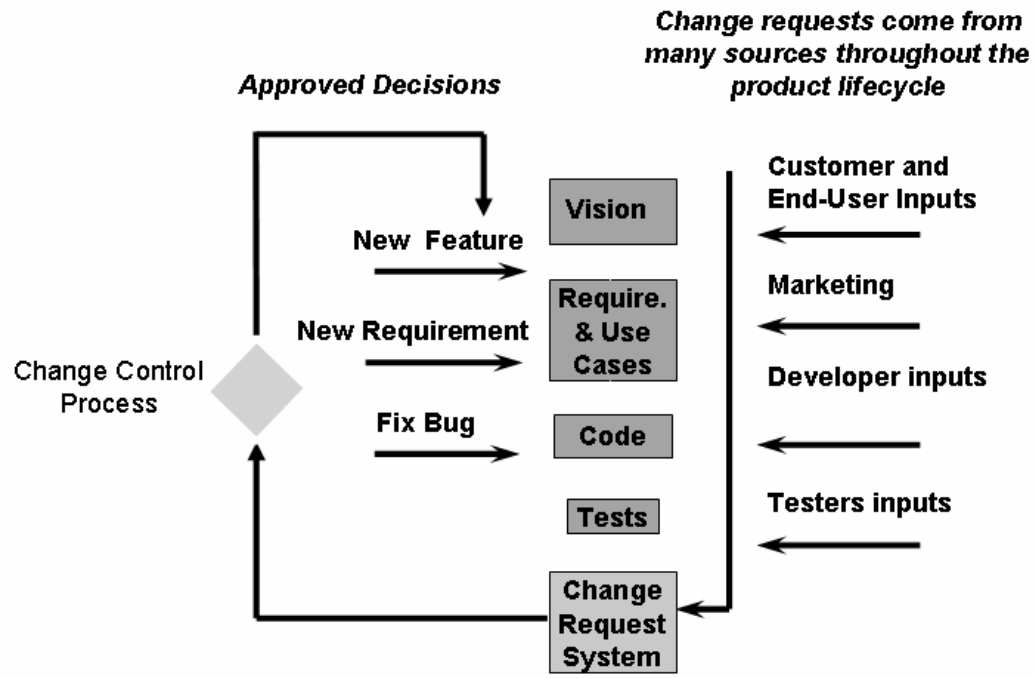
---

[1] - Control Changes
[2] - Iteration
[3] - Release
[4] - Platform
[5] - Configuration
[6] - Change
[7] - Change Request

- با فراهم‌آوری محیط‌های کاری مجزا و اَمن[1] برای هر یک از اعضای تیم، امکان انجام کارهای موازی و همروند، فراهم شده و از تداخلِ کاری که معمولاً در تیم‌های بزرگ اتفاق می‌اُفتد، جلوگیری می‌نماید.
- نرخِ آماری تغییرات و اثرات آنها، معیارهای خوبی برای ارزیابی کمّی و عینی وضعیتِ پروژه فراهم می‌آورد.
- سازگاری میانِ دستاوردهای مختلف، حفظ می‌شود. باید توجه داشت که عامل ناسازگاری، تغییر است و تنها کاری که می‌توان با آن انجام داد، کنترل است.
- تغییرات قبل از پخش‌شدن و تأثیر داده شدن، مورد ارزیابی قرار گرفته و کنترل می‌شوند.

شکل ۲-۲۰

کنترل یکپارچه‌ی تغییرات



[1] - Secure Workspaces

تحقق و پیاده‌سازی راهکارهای موفق در قالب فرایندِ تولید[۱]

یک تیم برای ایجاد یک سیستم و به عبارتی تحقق یک متدولوژی[۲] در قالب یک سیستمِ نرم‌افزاری، نیازمند سه مؤلفه‌ی اصلی می‌باشد. این مؤلفه‌ها، عبارتند از:

- فرایندِ تولید
- ابزار (ابزارهای کمک به مهندسی و تولید نرم‌افزار[۳])
- زبان یا روشی برای نمادگذاری[۴]

مشخص است که بکارگیری ابزارهای مناسب، نقش مهمی در بهبودِ کارایی تولید دارد. زبان یا روشِ نمادگذاری نیز برای انجامِ کارِ تیمی و تسهیلِ ارتباطات، ضروری است. نقشِ زبانِ مشترک در تولیدِ نرم‌افزار و تجربه‌ی موفق مرتبط با آن، یعنی استفاده از یک زبانِ بصری[۵]، در قالبِ چهارمین راهکارِ موفق بررسی شد.

حال باید نقش فرایند را در تولید نرم‌افزار بررسی نماییم. به طور کلی، فرایند تعریف می‌نماید که چه‌کسی[۶]، چه‌کاری[۷] را، چه موقع[۸]، و چگونه[۹] باید انجام دهد تا اینکه دستیابی به یک هدف مشخصِ امکان‌پذیر باشد. هدف مشخص و مقصودِ نهایی در یک فرایند تولیدِ نرم‌افزار[۱۰]، عبارتست از تولیدِ فراورده‌ای نرم‌افزاری دارای کیفیتِ مطلوب[۱۱].

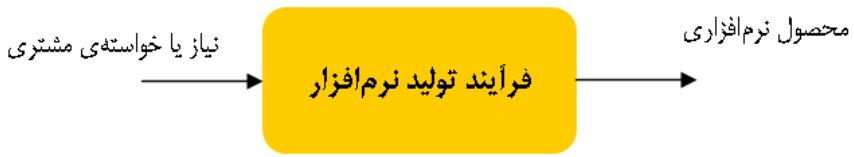
---

[1] - Development Process
[2] - Methodology
[3] - CASE Tools
[4] - Notation
[5] - Visual Language
[6] - Who
[7] - What
[8] - When
[9] - How
[10] - Software Development Process
[11] - Quality Software

در شکل ۲-۲۱ ، فرایند تولیدِ نرم‌افزار و ورودی‌ها و خروجی‌های آن نشان داده شده است. همان‌گونه که ملاحظه می نمایید، ورودی این فرایند، نیاز یا خواسته‌ی مشتری و خروجی آن، یک فراورده‌ی نرم‌افزاری است. البته ممکن است ورودی چنین فرایندی، خواسته‌های مشتری همراه با یک سیستم موجود (مثلاً نسخه‌ی یک از یک سیستم) باشد و خروجی مطلوبِ آن، یک سیستم بهبود یافته (مانند نسخه‌ی دوم از آن سیستم) باشد.

شکل ۲-۲۱

نمای کلی فرایندِ تولیدِ نرم‌افزار



مهم‌ترین نقش‌های یک فرایندِ تولید، عبارتند از:

- ارائه‌ی راهنمایی‌ها و توصیه‌هایی برای انجام کارهای تیمی به منظورِ تولیدِ بهینه‌ی یک فراورده‌ی نرم‌افزاری که دارای کیفیت مطلوب باشد.
- کمک به کاهش ریسک‌ها و افزایش قابلیت پیش‌بینی
- ترویج یک فرهنگِ کاری و دیدگاهِ مشترک
- کمک به نهادینه‌سازی و بکارگیری راهکارهای موفق در سازمان

یک فرایندِ مؤثر و مناسب باید دارای ویژگی‌های زیر باشد:

- به خوبی تعریف شده[۱] و به خوبی سازماندهی[۲] شده باشد، به گونه‌ای که قابل دسترسی برای همه‌ی ذینفعان اعم از مشتری و کارشناسانِ تیم تولید باشد.
- قابلیت سفارشی‌شدن[۳] و مقیاس‌پذیری[۴] برای پروژه‌ها و سازمان‌های مختلف
- قابلیتِ تکرارِ مجدد[۵]
- قابلیتِ پیش‌بینی[۶]
- امکانِ تکامل و بلوغ در طولِ زمان و در پروژه‌ها و سازمان‌های مختلف

باید توجه داشت که کیفیتِ فرایند به طورِ مستقیم بر کیفیتِ محصول تأثیرگذار است. بنابراین سازمان‌های مختلف در پی فرایند یا فرایندهایی با کیفیت و بلوغِ مناسب می‌باشند تا اینکه بتوانند بر اساسِ آن و با دسترسی به تجارب موفق دیگران، نسبت به موفقیتِ پروژه‌های خود اطمینان داشته باشند.

آر.یو.پی، فرایندی است که با پیاده‌سازی راهکارها و تجارب موفق، راههای موفق شدن را به ما می‌آموزد. از این منظر، آر.یو.پی گنجینه‌ای است ارزشمند از تجارب، توصیه‌ها، راهکارها. هدف اصلی این کتاب، معرفی چنین موجودیتی است. در واقع، در این کتاب چیستیِ[۷] آر.یو.پی و فلسفه‌ی آن را بررسی خواهیم نمود. البته، پیش از آن و در همین فصل‌های ابتدا، چراییِ[۸] مطرح شدن آر.یو.پی و سرچشمه‌ی آن را بررسی نموده‌ایم.

در بخش دوم از این کتاب، با اصول و مفاهیمِ بنیادی آر.یو.پی آشنا شده و سپس در بخش‌های سوم و چهارم جزئیاتی از مفاهیم فازها و دیسیپلین‌های آن را بررسی خواهیم نمود.

---

[1] - Well-defined
[2] - Well-Organized
[3] - Customization
[4] - Scalability
[5] - Repeatability
[6] - Predictability
[7] - What
[8] - Why

چکیده‌ی فصل

مهم‌ترین نکاتی که در این فصل عنوان گردید، عبارتند از:

- لزومِ دقت در نشانه‌های شکستِ پروژه‌ها و ریشه‌یابی عوامل و دلایلِ ریشه‌ای بروز این نشانه‌ها
- بررسی مهم‌ترین نشانه‌های حاکی از شکست و عدمِ موفقیت در پروژه‌های نرم‌افزاری
- بررسی دلایل و عواملِ ریشه‌ای شکست و عدمِ موفقیت در پروژه‌های نرم‌افزاری
- معرفی شش راهکارِ موفق که بر اساسِ تجارب پروژه‌ها و سازمان‌های موفق بدست آمده است. این راهکارها عبارتند از:
  - بهره‌گیری از فرایندی مبتنی بر رویکردِ تکرارشونده به جای رویکردِ منسوخِ آبشاری
  - داشتنِ روشی سیستماتیک و اصولی برای مدیریتِ نیازمندی‌های نرم‌افزار
  - بهره‌گیری از معماری‌های مبتنی بر مؤلفه
  - بهره‌گیری از تکنیکِ مدل‌سازی بصری و استفاده از زبانِ مدل‌سازیِ استانداردِ یو.اِم.اِل
  - ارزیابی مستمرِ کیفیت در طولِ فرایندِ تولید و نه فقط در انتهای آن
  - کنترل و مدیریتِ تغییرات

پرسش‌هایی برای مطالعه‌ی بیشتر

۱- لیستِ نشانه‌های شکستِ پروژه‌های نرم‌افزاری را کامل‌تر نمایید.

۲- با بررسی مقالات و منابع مختلف، سایر دلایلِ شکستِ پروژه‌های نرم‌افزاری را بیابید.

۳- غیر از شش راهکارِ موفقِ ذکر شده در این فصل، راهکارهای موفقِ دیگری را نیز در آر.یو.پی بیابید.

۴- درباره‌ی ویژگی‌ها و گرامرِ زبانِ مدل‌سازی یو.اِم.اِل تحقیق نمایید.

۵- در ارتباط با توسعه مبتنی بر مدل[1] و ارتباط آن با فرایندِ آر.یو.پی، تحقیق نمایید.

۶- Test Driven Development چیست و چه ارتباطی با راهکارِ ارزیابی مستمرِ کیفیت دارد؟

۷- ارتباط و تناظرِ میانِ راهکارهای موفق، دلایلِ شکست، و نشانه‌های ذکر شده در ابتدای فصل را در قالب یک نگاشت بیان کنید.

[1] - Model Driven Development

منابع و مراجع

[1]. Ivar Jacobson, Grady Booch, James Rumbaugh, (1999). *The Unified Software Development Process*, Reading, MA: Addison-Wesley.

[2]. Steve McConnell, (2003). *Professional Software Development: Shorter Schedules, Higher Quality Products, More Successful Projects, Enhanced Careers*, Reading, MA: Addison Wesley.

[3]. Robert L. Glass, (2002). *Facts and Fallacies of Software Engineering*, Reading, MA: Addison Wesley.

[4]. Scott E. Donaldson, Stanley G. Siegel, (2000). *Successful Software Development*, Reading, NJ: Prentice Hall PTR.

[5]. Pressman, R. S. (2000). *Software engineering: A practitioner's approach*. 5th ed. New York: McGraw-Hill.

[6]. Boehm, B. W., and K. Sullivan. (2000). *Software economics: A roadmap.* In The future of software engineering, ed. A. Finkelstein. 22d International Conference on Software Engineering.

[7]. Software Academy. (2006) "Unified Process Knowledge Base," Available at: http://www.unifiedProcess.info/

[8]. Philippe Kruchten, (2003). *The Rational Unified Process: An Introduction*, Reading, MA: Addison-Wesley.

[9]. Per Kroll, Philippe Kruchten, (2003). *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*, Reading, MA: Addison-Wesley.

[10]. Rational Software Corporation. (2003) "Rational Unified Process 2003.06," Available at: http://www.rational.com/

[11]. Craig Larman, (2003). *Agile and Iterative Development: A Manager's Guide*, Reading, MA: Addison-Wesley.

بخش دوم

فصل سوم: آر.یو.پی چیست؟

فصل چهارم: ویژگی‌ها و روحِ آر.یو.پی

فصل سوم

آر.یو.پی چیست؟

مهم‌ترین موضوعات مورد بررسی در این فصل، عبارتند از:

- تعاریف و مفاهیم کلیدی آر.یو.پی
- ساختار آر.یو.پی
  - ساختار پویا (دینامیک)
  - ساختار ایستا (استاتیک یا محتوایی)

۳ آر.یو.پی چیست؟

در این فصل مروری اجمالی و در عین حال جامع بر ساختار و مفاهیم کلیدی آر.یو.پی خواهیم داشت. پس از بررسی مفهوم آر.یو.پی، ساختار دو بُعدی آن معرفی شده و به طور مختصر تشریح می‌گردد. توصیف کامل هر یک از این اَبعاد، به ترتیب، موضوعِ بخش‌های دوم و سومِ این کتاب می‌باشد. خواننده‌ی محترم توجه داشته باشد که به منظور همراهی با مطالب و درک بهتر مفاهیم، مرور این فصل و به طور کلی مرور بخش اولِ این کتاب مفید خواهد بود.

فرایند یکپارچه‌ی رَشنال[1] (یا به اختصار، آر.یو.پی[2]) چیست؟

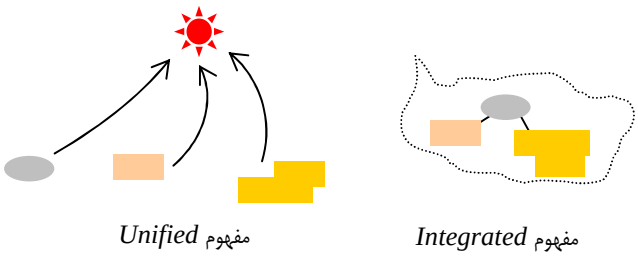
اجازه دهید پیش از تعریف آر.یو.پی و آشنایی با معنای دقیق آن، به معنای لغوی آن اشاره‌ای داشته باشیم. حرفِ اولِ این واژه (یعنی حرفِ آر) مخفف کلمه‌ی رَشنال[3] می‌باشد. رَشنال نام یکی از شرکت‌های بزرگ در صنعتِ نرم‌افزار است. این شرکت نقش مهمی در توسعه‌ی صنعت نرم‌افزار ایفا نموده است. این شرکت در سال ۲۰۰۳ رسماً توسط شرکت آی.بی.اِم[4] خریداری شد. بنابراین در حال حاضر، مالکیت آر.یو.پی (به عنوان یکْ محصول) در اختیار شرکت آی.بی.اِم می‌باشد. حرفِ دوم در واژه‌ی آر.یو.پی (یعنی حرفِ یو) مخففِ کلمه‌ی یونی‌فاید[5] و به معنای تلفیقْ شده، یکی‌شده، و متحدالشکل می‌باشد. با وجودیکه در این کتاب از واژه‌ی **یکپارچه** استفاده شده است، اما توجه داشته باشید که اصطلاحِ یکپارچه در اینجا به معنای یکی‌شده بکار می‌رود و در واقع، یکپارچگی معادل واژه‌ی Integrated مورد نظر نیست.

---

[1] - Rational Unified Process
[2] - RUP
[3] - Rational
[4] - IBM
[5] - Unified

شکل ۳-۱

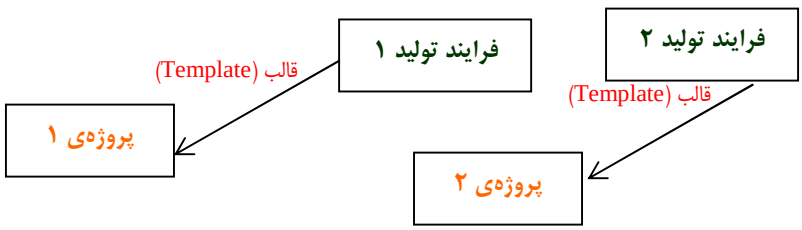
مقایسه‌ی مفاهیم Unified و Integrated



حرفِ سوم واژه‌ی آر.یو.پی (یعنی حرفِ پی) مخفف واژه‌ی پروسس[1] به معنای فرایند می‌باشد. البته ممکن است مفهومِ فرایند، موضوعات مختلفی را در ذهن شما تداعی نماید. توجه داشته باشید که در این کتاب، مفهومِ فرایند، به طور خلاصه به جای عبارتِ فرایندِ تولید، استفاده می‌شود. اجازه دهید فعلاً فرایند تولید را مجموعه‌ای از فعالیت‌های مشخص و دارای ترتیبی معیّن که به منظور تولید یک محصولْ با توجه به یک نیاز و درخواست مشخص، تعریف نماییم. فرایند تولید، قالب و الگوی پروژه را تعریف می‌نماید. البته در ادامه‌ی این کتاب، تعریف دقیق‌تر و مناسب‌تری از مفهوم فرایند تولید ارائه خواهیم داد.

شکل ۳-۲

ارتباط میان فرایند تولید و پروژه



[1] - Process

حال می‌توانیم تعریف دقیق‌تری از آر.یو.پی داشته باشیم. آر.یو.پی، سه مفهوم و معنای تا حدی متفاوت را در بر می‌گیرد:

۱- آر.یو.پی یک **رویکرد**[۱] و روش برای تولید نرم‌افزار می‌باشد. این رویکرد، دارای ویژگی‌های برجسته‌ای مانند تکرارشونده[۲] بودن، تمرکز بر معماری[۳] و مبتنی بودن بر مواردِ کاربرد[۴] (یا به عبارت ساده‌تر، مبتنی بر خواسته‌های مشتری) می‌باشد.

۲- آر.یو.پی یک **فرایند** به خوبی تعریف‌شده[۵] و سازماندهی‌شده‌ی مهندسی نرم‌افزار می‌باشد. نقش‌ها[۶]، فعالیت‌ها[۷]، دستاوردها[۸]، و جریان‌های کار[۹] (ترتیب و توالی فعالیت‌ها) تعریف شده در آر.یو.پی، عناصر اصلی یک فرایند (یعنی **چه‌کسی**[۱۰]، **چه‌کاری**[۱۱]، **چگونه**[۱۲]، و **چه‌موقع**[۱۳]) را تعریف و تبیین می‌نماید. آر.یو.پی ساختار مناسبی برای کنارِ هم گذاشتنِ این مؤلفه‌ها فراهم نموده است. نحوه‌ی سازماندهی این ساختار به دو بُعدِ دینامیک و استاتیک، یکی از ویژگی‌های کم‌نظیرِ آر.یو.پی می‌باشد.

۳- آر.یو.پی **محصولی**[۱۴] است دربرگیرنده‌ی **چارچوب**[۱۵] و قالبِ کلی فرایندهای تولیدِ سیستم‌های نرم‌افزاری. از این منظر، آر.یو.پی یک فرایند نیست که بتوان آن‌را مستقیماً به عنوان قالب تعریف یک پروژه‌ی تولیدِ نرم‌افزار بکار گرفت، بلکه مفهومی است بسیار فراتر و جامع‌تر. در واقع آر.یو.پی، مانند یک میز پر از غذاهای متنوع در یک رستوران می‌باشد؛ مسلماً، هیچ یک از مهمان‌های این رستوران قادر نخواهد بود همه‌ی غذاها را میل نماید. در عوض، هر کس با توجه به ذائقه و نیازش (و البته موجودی داخل جیبش!) گلچینی از غذاها را انتخاب و میل می‌نماید. آر.یو.پی مخزن یا بانک دانشِ بزرگی از راهکارها و تجاربِ موفق برای شرایط و طیف گسترده‌ای از پروژه‌های مختلف،

---

[1] - Approach
[2] - Iterative
[3] - Architecture-Centric
[4] - Use-Case Driven
[5] - Well-defined
[6] - Roles
[7] - Activities
[8] - Artifacts
[9] - Workflows
[10] - Who
[11] - What
[12] - How
[13] - When
[14] - Process Product
[15] - Process Framework

فراهم نموده است. هیچ پروژه‌ای در جهان نمی‌تواند منطقاً (و البته مادامی که بتوان آن‌را پروژه نامید) از همه‌ی این راهکارهای موفق، فعالیت‌های تعریف شده، راهنمایی‌ها و دستاوردهای مختلف، آنگونه که در آر.یو.پی تعریف شده، استفاده نماید. هر پروژه‌ای با توجه به ذائقه، نیازها، محدودیت‌ها، و امکاناتِ خاصش، به گونه‌ی متفاوتی از این بانک دانش استفاده می‌نماید. آر.یو.پی به عنوان چارچوبِ فرایند، دارای قابلیت سفارشی‌سازی[۱] و پیکربندی[۲] برای طیف وسیعی از پروژه‌ها می‌باشد. به کمک آر.یو.پی، می‌توان فرایندی مناسب برای تولید یک محصول نرم‌افزاری بسیار کوچک (پروژه‌ای در ابعاد یک نفر و دو هفته زمان!) یا یک پروژه‌ی نرم‌افزاری بزرگ (مثلاً پروژه‌ای چند ملیتی در طول ۵ سال و با بیش از ۱۰ هزار نفر نیروی انسانی) را تعریف و با موفقیت اجرا نمود. این ویژگی که آن‌را **مقیاس‌پذیری**[۳] می‌نامند، یکی دیگر از ویژگی‌های کم‌نظیر آر.یو.پی محسوب می‌شود.

کتاب‌ها و مقالات متعددی در رابطه با آر.یو.پی نوشته شده است. اما کامل‌ترین و جامع‌ترین منبع اطلاعاتی مرتبط با آر.یو.پی، در قالب یک محصولِ پیاده‌سازی شده با فناوری وب[۴] و روی یک لوح فشرده[۵]، تولید شده است. این محصول که توسط شرکت رَشنال[۶] (و اکنون آی.بی.اِم[۷]) عرضه می‌شود، شامل کلیه‌ی راهنمایی‌ها، مثال‌ها، تعاریف، الگوها، و قالب‌هایی است که سرتاسر چرخه‌ی تولید[۸] محصولات نرم‌افزاری (یعنی از زمان شروع پروژه تا انتهای کار و تحویلِ کاملِ محصول بدست آمده) را تحت پوشش قرار می‌دهد. در ادامه‌ی این فصل، هر یک از تعاریف ذکر شده و مفاهیم مرتبط با آنها را تشریح خواهیم نمود.

---

[1] - Customization
[2] - Configuration
[3] - Scalability
[4] - Web Technology
[5] - CD or Compact Disk
[6] - Rational
[7] - IBM
[8] - Development Cycle

آر.یو.پی به عنوان یک رویکردِ مهندسی نرم‌افزار

از این منظر، آر.یو.پی همانند یک بانک دانش و گنجینه‌ای از تجارب، الگوها، و راهکارهای موفق در صنعت نرم‌افزار می‌باشد. ویژگی‌های کلیدی آر.یو.پی که آن‌را از سایر رویکردهای مهندسی نرم‌افزار متمایز می‌نماید، عبارتند از:

- توسعه و تولید با رویکردِ تکرارشونده[۱] در مقابلِ رویکرد توسعه به روشِ آبشاری[۲] و یا سایر رویکردهای دیگر (مانند حلزونی[۳] ، توسعه‌ی سریعِ سیستمِ کاربردی[۴] ، پیش‌الگوسازی[۵] و ...)
- تمرکز بر معماری[۶] (محوریت معماری در فرایند)
- توسعه مبتنی بر مواردِ کاربرد[۷] (مشتری مداری)

اصول بنیادی آر.یو.پی که آنها را روحِ آر.یو.پی[۸] نیز نامیده‌اند، عبارتند از:

- از همان ابتدا و به طور مستمر بر **ریسک‌ها**[۹] (مخاطرات) اصلی و مهم پروژه‌تان غلبه نمایید، در غیر این صورت، این ریسک‌ها بر شما غلبه خواهند کرد!
- اطمینان یابید که در طول فرایند، فعالیت‌های شما همواره برای مشتری **ارزش**[۱۰] **افزوده‌ای** در بر دارند.
- همواره بر داشتن یک **نرم‌افزارِ قابل اجرا**[۱۱] در تمام مقاطع و در طول پروژه (نه فقط در انتهای آن) تأکید داشته باشید.
- از همان ابتدای پروژه، در اندیشه‌ی راهکار مناسبی برای **مدیریتِ تغییرات**[۱] باشید و هرگز این کار را به تعویق نیاندازید.

---

[1] - Iterative Development
[2] - Waterfall
[3] - Spiral
[4] - Rapid Application Development (RAD)
[5] - Prototyping
[6] - Architecture-Centric
[7] - Use-Case Driven Development
[8] - RUP Spirit
[9] - Risks
[10] - Value
[11] - Executable Software

- رسیدن به یک چارچوب یا **معماری**[۲] **مستحکم** و قابل اجرا[۳] و مبنا قرار دادن آن را در اولویت قرار دهید. (تنها به داشتن طرح و نقشه‌ی یک معماری اکتفا ننمایید، معماری باید قابلیت اجرا شدن داشته باشد و بدین منظور، باید بخش‌های کلیدی سیستم، پیاده‌سازی شده باشد)
- سیستم را با استفاده از **مؤلفه‌های**[۴] **نرم‌افزاری** بنا نمایید. لازم به ذکر است که در اینجا منظور از یک مؤلفه‌ی نرم‌افزاری، ماژول[۵] یا پیمانه‌ای است که اجزای داخلی آن بیشترین چسبندگی[۶] را با هم دارند. مجموعه‌ی کلی این پیمانه به مانند یک جعبه‌ی سیاه عمل کرده و کمترین تداخل[۷] را با سایر مؤلفه‌های سیستم دارد.
- تولید یک فراورده‌ی نرم‌افزاری، مانند یک ورزش تیمی است. همه‌ی افراد مشارکت کننده در جریان تولید یک محصول نرم‌افزاری باید در قالب یک و تنها **یک تیم** فعالیت کنند. در اینجا لازم است خواننده‌ی گرامی به لفظِ یک و تنها یک تیم به دقت توجه داشته باشد.
- **کیفیت** را در بطنِ همه‌ی فعالیت‌های خود قرار دهید. کیفیت چیزی نیست که بتوان آن‌را در انتهای کارها، پس از پیاده‌سازی سیستم و مثلاً با انجام تست، بدست آورد! کیفیتِ تک‌تکِ فعالیت‌های هر یک از افرادِ تیم، بر کیفیتِ کلی محصول، تاثیرگذار می‌باشد.

توصیف کامل‌تری از ویژگی‌های ذکر شده و نیز تشریحِ اصول بیان شده را به فصل آینده موکول می‌نماییم.

---

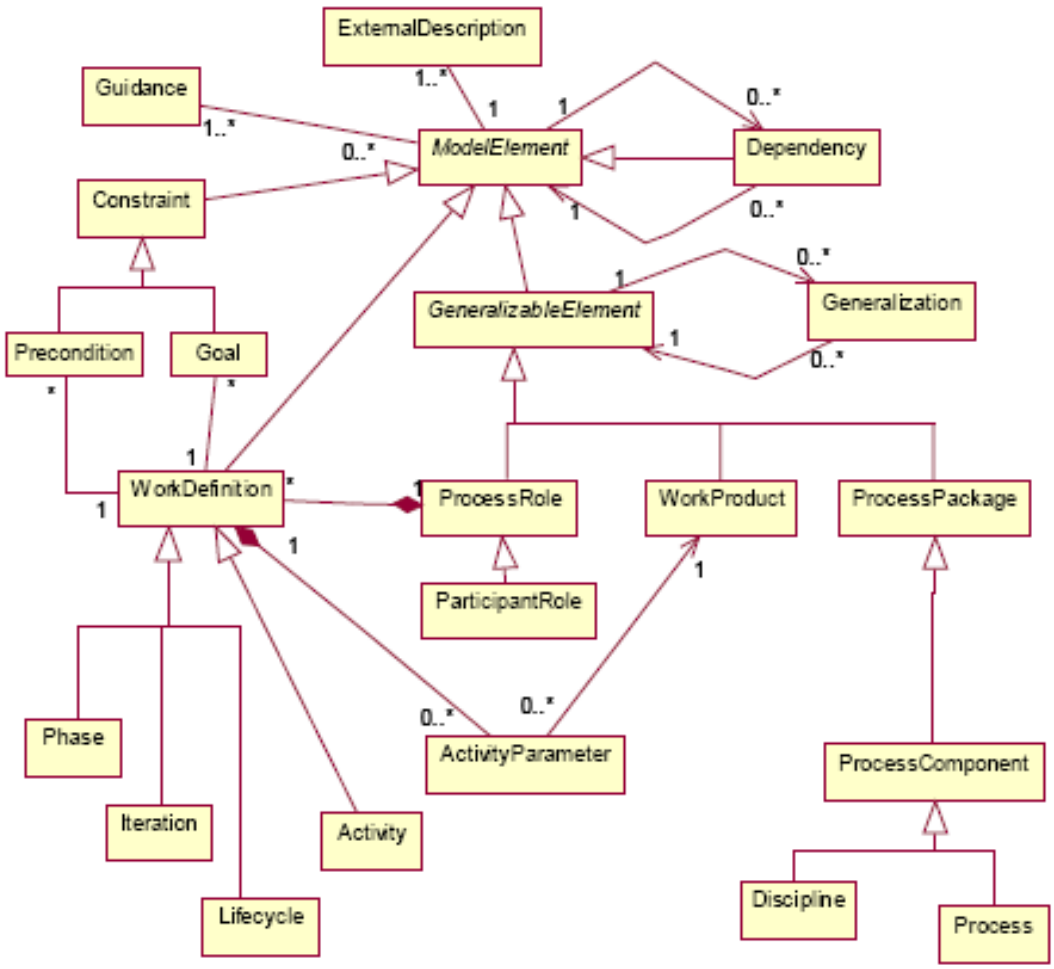
[1] - Change Management
[2] - Architecture
[3] - Executable Architecture
[4] - Component
[5] - Module
[6] - Cohesion
[7] - Coupling

آر.یو.پی به عنوان یک فرایندِ به‌خوبی تعریف شده‌ی تولید نرم‌افزار

آر.یو.پی خود به عنوان یک فرایند تولید و مهندسی نرم‌افزار، با استفاده از تکنیک‌های طراحی و مدل‌سازی نرم‌افزار، طراحی شده است. تعریفِ کامل این فرایند به وسیله‌ی اَبَر مدلی تحت عنوان مدلِ مهندسی فرایند نرم‌افزار[1] که استانداردی برای مدل‌سازی فرایندِ مبتنی بر زبان مدل‌سازی استاندارد یو.اِم.اِل می‌باشد، صورت پذیرفته است. این مدل مرجع، در شکل ۳-۳ نشان داده شده است.

شکل ۳-۳

مدلِ مهندسیِ فرایند نرم‌افزار؛ ارائه شده توسط OMG

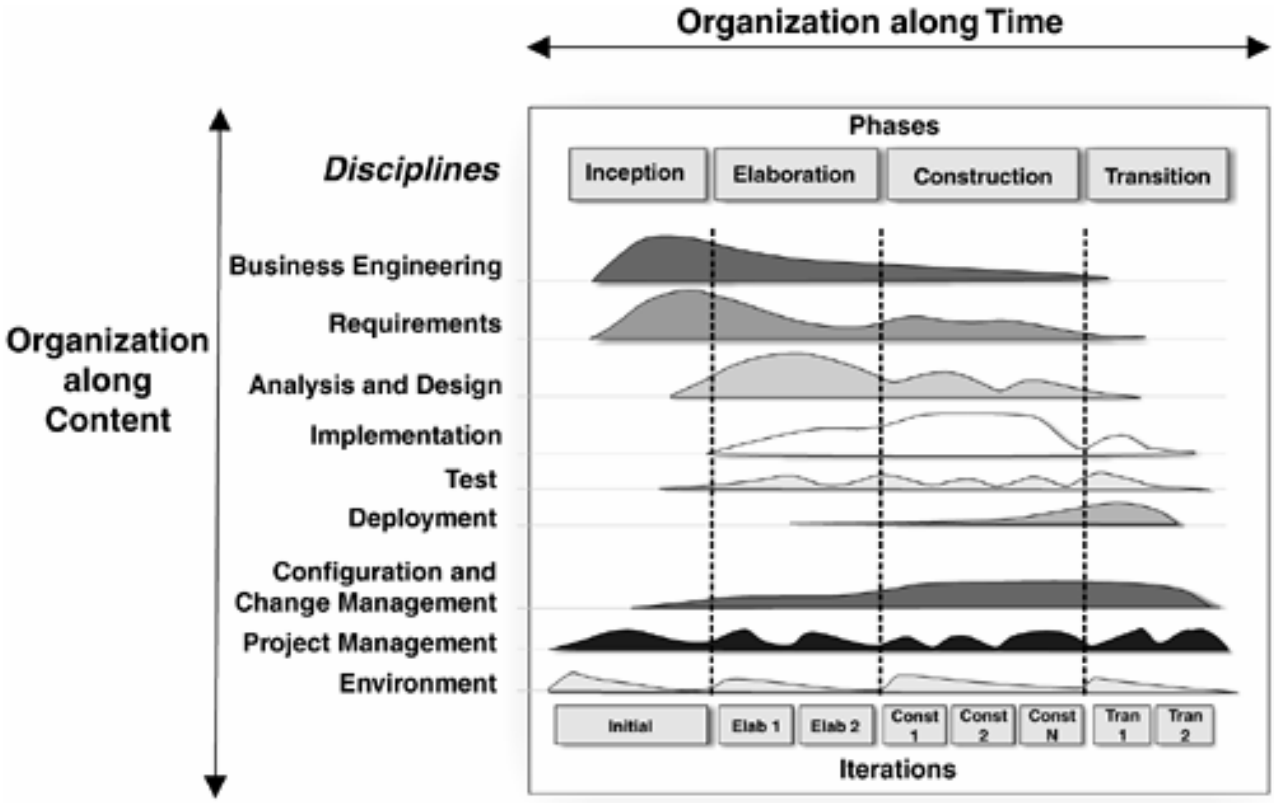


[1] - Software Process Engineering Model (SPEM)

در شکل ۳-۴ ، معماری کلی آر.یو.پی نشان داده شده است. همانگونه که در این شکل ملاحظه می‌نمایید، این فرایند دارای ساختاری دو بُعدی است.

شکل ۳-۴

سازماندهی فرایند آر.یو.پی در دو بُعد زمانی و محتوایی (دینامیک و استاتیک)



دو بعد تشکیل دهنده‌ی ساختاری متعامدِ[1] آر.یو.پی، عبارتند از:

- ساختار دینامیک[2] (پویا): ساختار دینامیک آر.یو.پی، بُعد افقیِ نشان داده شده در شکل ۳-۴ و بیانگر ساختار پویا و ملاحظات مرتبط با زمان در فرایند می‌باشد. در این بُعد، ملاحظاتی مانند چرخه‌های[3] توسعه (یا چرخه‌های تولید)، فازها[4] ، تکرارها[5]، و نقاطِ تصمیم‌گیری کلیدی[6] مطرح می‌باشد. این

[1] - Orthogonal
[2] - Dynamic
[3] - Cycles
[4] - Phases
[5] - Iterations
[6] بکار رفته است.Business Decision Point و Major Milestone - معادلی است که در این کتاب به جای عبارت

مفاهیم در کنارِ هم، چرخه‌ی عمر یک پروژه‌ی نرم‌افزاری (پروژه‌ی تولید یک محصول نرم‌افزاری) را تعریف می‌نمایند.

- ساختار محتوایی[۱] (استاتیک): همانگونه که در شکل ۳-۴ نشان داده شده است، ساختارِ آر.یو.پی دارای یک بُعد عمودی نیز می‌باشد که بیانگر ساختار استاتیک یا محتوایی[۲] آن است. در این بُعد، توصیفی از چگونگی دسته‌بندی و سازماندهی عناصر محتوایی فرایند یعنی مجموعه‌ی فعالیت‌ها[۳]، راهنمایی‌ها[۴]، دستاوردها[۵] و نقش‌ها[۶] در قالب دیسیپلین‌ها[۷] یا جریان‌های منظم و منطقی مجموعه‌ی کارها[۸] می‌باشد.

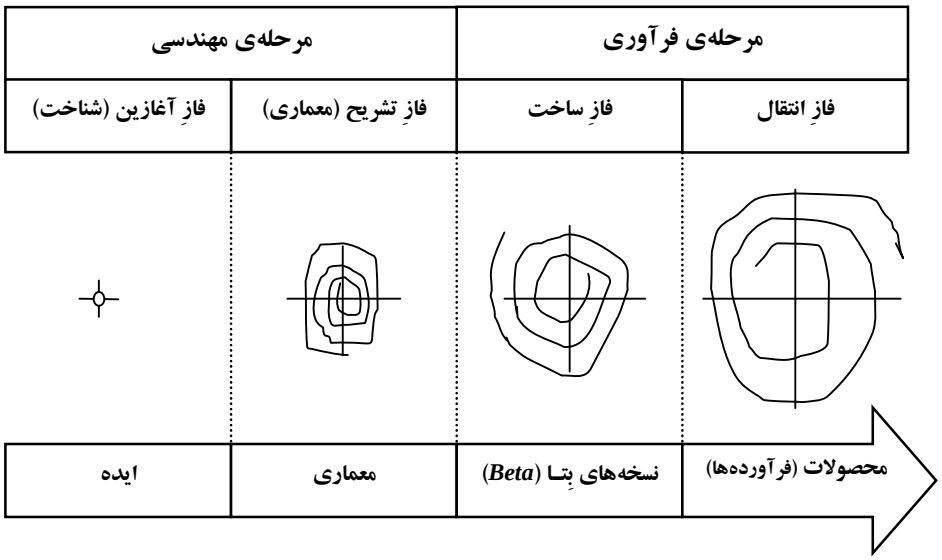
[1] - Content
[2] - Content
[3] - Activities
[4] - Guidelines
[5] - Artifacts
[6] - Roles
[7] - Disciplines
[8] - Workflow

ساختارِ دینامیکِ آر.یو.پی

بر اساس فرایند آر.یو.پی، یک پروژه‌ی نرم‌افزاری که با هدف تولید یک فراورده‌ی نرم‌افزاری تعریف و اجرا می‌گردد، از نظر دینامیکی یا بُعدِ زمانی دارای چهار فاز می‌باشد: فازِ آغازین[۱] (یا فاز شناخت)، فازِ تشریح[۲] (یا فاز معماری)، فازِ ساخت[۳] و فازِ انتقال[۴]. در شکل ۳–۵ ، نمای کلی این چهار فاز نشان داده شده است.

شکل ۳–۵

چهار فاز یک پروژه در آر.یو.پی و مراحل مهندسی و فرآوری



دو فازِ اول را مرحله‌ی مهندسی[۵] و دو فازِ آخر را مرحله‌ی فرآوری[۶] می‌نامند. همانگونه که در شکل ۳–۵ ملاحظه می‌نمایید، نرم‌افزار در طول چهار فاز آر.یو.پی، روندی تکاملی را پشت سر می‌گذارد.

در این فرایند، تکامل فراورده‌ی نرم‌افزاری را می‌توان به بزرگ‌شدن یک گلوله‌ی کوچک برف که از بالای یک کوه به سمت پایین سرازیر شده است، تشبیه نمود. البته اگر در مسیر این گلوله‌ی برف موانع پیش‌بینی نشده‌ای (یا همان ریسک‌ها) وجود داشته باشد، گلوله متلاشی خواهد شد و احتمالاً به جای یک گلوله‌ی کامل برف که معادل یک فراورده‌ی نرم‌افزاری با کیفیت مطلوب می‌باشد، چندین قطعه‌ی کوچک

[1] - Inception
[2] - Elaboration
[3] - Construction
[4] - Transition
[5] - Engineering Stage
[6] - Production Stage

نصیبمان خواهد شد. بنابراین یک جایی در این مسیر باید آگاهی و اطلاع خوبی از موانع احتمالی و همچنین استحکام مناسب گلوله‌ی برف داشته باشیم و البته این کار با نشستن بالای کوه و نقشه و طرح ریختن ممکن نیست! این ملاحظات در مورد نرم‌افزار یعنی تثبیت و استحکام معماری آن. آر.یو.پی که بر اساس تجربه‌ی موفق پروژه‌های مختلف ایجاد شده، به دلیل اهیمت موضوع معماری و تثبیت هر چه سریع‌تر آن، یک فاز را به معماری اختصاص داده است.

در انتهای هر یک از فازهای چهارگانه‌ی آر.یو.پی، یک نقطه‌ی تصمیم‌گیری کلیدی یا سازمانی[۱]، وجود دارد. در واقع، بر خلاف فازهای فرایند آبشاری که ماهیتِ فعالیت و کار را در بر دارند، مفهوم فازهای آر.یو.پی، رسیدن به یک نقطه‌ی تصمیم‌گیری کلیدی و اتخاذ تصمیم مناسب می‌باشد (به عنوان مثال، فاز شناخت در رویکرد آبشاری، بازه‌ی زمانی است که در آن مجموعه‌ی فعالیت‌های مرتبط با شناخت انجام شده و پایان این فاز منوط به انجام شدن کاملِ این مجموعه فعالیت‌ها می‌باشد). بدین ترتیب برگشت از یک فاز به فاز قبلی در فرایند آر.یو.پی معنایی ندارد. در صورتی‌که داشتن برگشت‌های مکرر از یک فاز به فاز یا فازهای قبلی، از ویژگی‌های فرایندهای مبتنی بر رویکرد آبشاری می‌باشد. توجه داشته باشید که فرایند آبشاری با داشتن ویژگی برگشت‌های مکرر و عمدتاً غیر قابل پیش‌بینی، دیگر جایگاهی در صنعت نرم‌افزار ندارد!

در بسیاری از موارد در فازهای مختلف دستیابی به تصمیمات کلیدی با انجام یکباره‌ی مجموعه‌ی فعالیت‌های مهندسی امکان پذیر نمی‌باشد. اینجاست که مفهوم تکرار[۲] مطرح می‌گردد. هر یک از فازهای آر.یو.پی می‌تواند شامل یک یا چند تکرار باشد. تکرارها عمدتاً به تصمیمات فنی[۳] منجر می‌شوند.

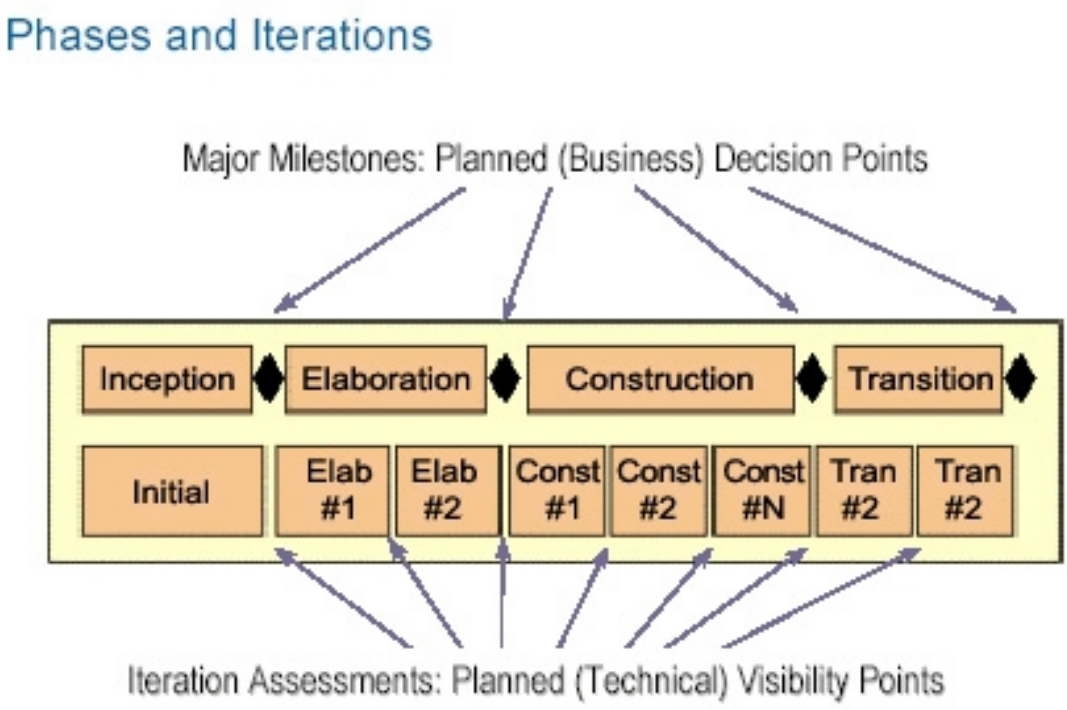
در آر.یو.پی، اصطلاح *مِی‌ژور مایل اِستون*[۴] اشاره به نقاط تصمیم‌گیری انتهای هر فاز داشته و اصطلاحِ مینور *مایل اِستون*[۵] برای توصیفِ نقاطِ انتهای تکرارها که نقاط تصمیم‌گیری فنی یا نقاطِ آشکار شدنِ ملاحظات فنی[۶] در داخل فازهای مختلف می‌باشند، بکار می‌رود.

---

[1] - Business Decision Point
[2] - Iteration
[3] - Technical
[4] - Major Milestone
[5] - Minor Milestone
[6] - Technical Visibility Point

شکل ۳-۶

نقاط تصمیم‌گیری کلیدی (سازمانی) و ارتباط آنها با انتهای فازهای آر.یو.پی



همانگونه که اشاره گردید، فازهای آر.یو.پی مفهوم تصمیم‌گیری و دستیابی به یکسری تصمیمات کلیدی را در بر می‌گیرند. تمام فعالیت‌های یک فاز طوری برنامه‌ریزی می‌شود که با توجه به ویژگی‌های پروژه و ملاحظات مختلف مرتبط با آن، به تصمیمات خاصی منجر شوند. هرگاه در یک فاز با تکرارهای[1] برنامه‌ریزی شده، دستیابی به نتایج مورد انتظار برای اتخاذ تصمیم‌های کلیدی ممکن نباشد، بدون رفتن به فاز بعدی، باید یک تکرار دیگر در همان فاز، در نظر گرفته شود. بنابراین در هر تکرار، باید به دقت، به اهداف پیش‌بینی شده توجه داشت و از انجام فعالیت‌های اضافی که نقش کمتری در دستیابی به تصمیم‌های کلیدی دارند، پرهیز نمود.

[1] - Iteration

به طور کلی، اهداف کلیدی هر یک از فازهای آر.یو.پی به شرح زیر می‌باشد:

- فاز آغازین[۱] (شناخت): اثبات درک صورت مسأله و مشکلات موجود، تسکین[۲] و فرونشانی ریسک‌ها (مخاطرات) سازمانی، و جلب نظرِ موافق تمام ذینفعان[۳] نسبت به مقرون به‌صرفه‌بودن و نیز امکان‌پذیر بودن ادامه‌ی پروژه.
- فاز تشریح[۴] (معماری): غلبه بر ریسک‌های فنی با تثبیت یک معماری قابل اجرا[۵] ، دقیق‌تر نمودن برنامه‌ی اجرایی پروژه[۶].
- فاز ساخت[۷] : ایجاد سیستمی با تمام قابلیت‌های مورد توافق (نسخه‌ی بِتا[۸]).
- فاز انتقال[۹] : کسب اطمینان نسبت به اینکه سیستم نرم‌افزاری حاصل، تمام خواسته‌ها و نیازهای تثبیت‌شده‌ی کاربران را برآورده می‌کند و انتقال کامل محصول به محیط کاربران نهایی آن.

در روشِ سنتیِ برنامه‌ریزی[۱۰] پروژه (یعنی روش آبشاری[۱۱])، سازماندهی برنامه‌ی اجرایی، عمدتاً به صورت بالا به پایین[۱۲] و مبتنی بر اجزاء محصول[۱۳] انجام می‌شد. به عبارت دیگر، برنامه‌ریزی پروژه بر اساسِ تجزیه‌ی سیستم به مؤلفه‌ها و انواع دستاوردهای مختلفِ مربوط به محصول نهایی مانند توصیف‌ها، طرح‌ها و نقشه‌ها، صورت می‌گرفت. این شیوه‌ی برنامه‌ریزی، از صنایع ساخت و تولید[۱۴] به ارث رسیده است. در آر.یو.پی، برنامه‌ریزی مبتنی بر شکستنِ فرایند[۱۵] می‌باشد. این بدان معناست که برنامه‌ی اجرایی یک پروژه بر اساس الگوی تعریف شده در آر.یو.پی، بیانگر اینست که برای دستیابی به یکسری اهداف و مقصودهای[۱۶] مشخص در طول زمان، چه کارهایی باشد انجام شود. البته توجه داشته باشید که در آر.یو.پی، برنامه‌ریزی با

---

1 - Inception
2 - Mitigation
3 - Stakeholders
4 - Elaboration
5 - Executable Architecture
6 - Project Plan
7 - Construction
8 - Beta Release
9 - Transition
10 - Planning
11 - Waterfall
12 - Top-Down
13 - Product Breakdown
14 - Manufacturing & Production
15 - Process Breakdown
16 - Objectives

هر دو رویکرد بالا به پایین و نیز پایین به بالا[1] انجام می‌شود. در مرحله‌ی مهندسی (یعنی فازهای آغازین و تشریح)، رویکرد غالب، بالا به پایین می‌باشد و در مرحله‌ی فرآوری (یعنی فازهای ساخت و انتقال)، از رویکردِ پایین به بالا، برای برنامه‌ریزی استفاده می‌شود. رویکردِ پایین به بالا، از دستاوردهای تعریف شده[2] و نیز مبنایی که بر اساس معماری شکل گرفته[3] ، استفاده می‌نماید.

ساختارِ محتوایی آر.یو.پی

از بُعدِ ساختارِ استاتیک یا محتوایی، عناصر کلیدی فرایند، یعنی فعالیت‌ها[4]، دستاوردها[5]، و نقش‌ها[6] در قالب تعدادی دیسیپلین[7] ، گروه‌بندی و سازماندهی شده‌اند. برای توصیف و مدل‌سازی ترتیبِ منطقی مجموعه فعالیت‌های هر یک از این دیسیپلین‌ها، از مدل جریان‌کار[8] استفاده می‌شود. لازم به ذکر است که تمامی مؤلفه‌های ساختار محتوایی آر.یو.پی با زبان مدل‌سازی استانداردی تحت عنوانِ یو.اِم.اِل[9] ، مدل‌سازی شده اند.

همانگونه که می‌دانید، یک فرایند (و در واقع، یک فرایند تولید) توصیف می‌نماید که چه‌کسی[10]، چه‌کاری[11] را، چگونه[12]، و چه‌موقع[13] باید انجام دهد تا اینکه به نتیجه‌ی مطلوب که یک محصولِ با کیفیت[14] است، دست یابیم. از این منظر، نقش‌ها معادل مؤلفه‌ی چه‌کسی، فعالیت‌ها معادل مؤلفه‌ی چگونگی، دستاوردها معادل مؤلفه‌ی چه‌چیزی، و جریان‌کار معادل مؤلفه‌ی چه‌موقع در فرایند می‌باشد.

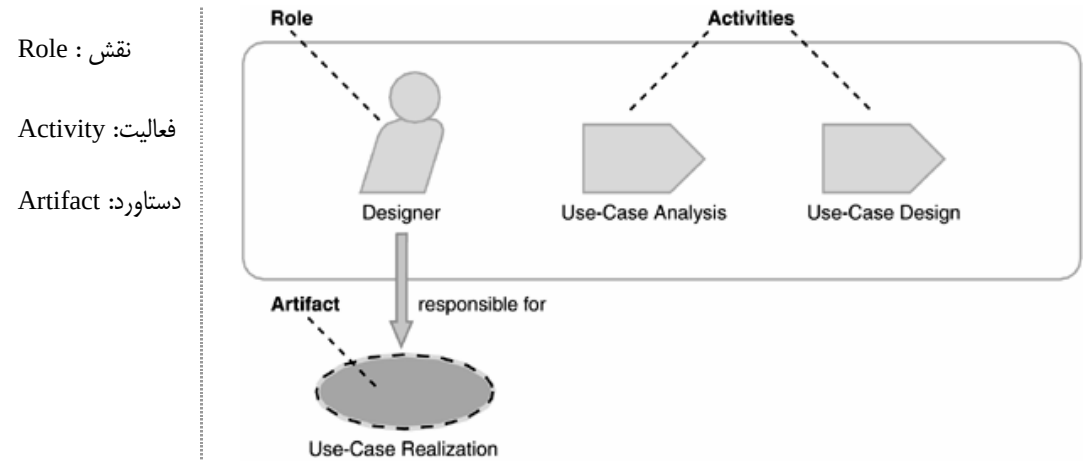
---

1 - Bottom-Up
2 - Defined Artifacts
3 - Architectural Baseline
4 - Activities
5 - Artifacts
6 - Roles
7 - Core Disciplines
8 - Workflow
9 - UML
10 - Who
11 - What
12 - How
13 - When
14 - Quality Product

شاید برای شما سؤال باشد که چرا آر.یو.پی از واژه‌ی **دیسیپلین**[1] استفاده کرده است. دیسیپلین در لغت به معنای انضباط و نظمِ کاری می‌باشد. در واقع، از آنجایی که بطور مثال مجموعه‌ی فعالیت‌ها، دستاوردها، و نقش‌هایی که همگی با هدف فراهم نمودن، تجزیه و تحلیل، مدل‌سازی، و پالایش نیازمندی‌های نرم‌افزاری و نیازهای مشتری انجام می‌شود، دارای یکسری ویژگی‌های فکری، رفتاری، هدف، و عملکرد مشابهی می‌باشند که با مجموعه‌ی فعالیت‌ها، دستاوردها، و نقش‌های دیگرِ فرایند که هدف آنها تحلیل و طراحی سیستم، پیاده‌سازی، تست و یا استقرار محصول می‌باشد، متفاوت است. در واقع، مفهوم دیسیپلین این ذهنیت را تداعی می‌نماید که نوعی ارتباط منطقی میان مجموعه‌ی فعالیت‌ها، نقش‌ها، و دستاوردهای هر دیسیپلین وجود دارد. در شکل۳-۷ ، نمونه‌ای از ارتباط میان عناصر کلیدی در ساختار محتوایی آر.یو.پی نشان داده شده است.

شکل ۳-۷

ارتباط میان مولفه‌های کلیدی در ساختار محتوایی آر.یو.پی



حال که با مفهوم دیسیپلین آشنا شدیم، اجازه دهید تعریف مختصری از مفاهیم نقش، فعالیت، دستاورد، و جریانِ‌کار ارائه نماییم.

نقش[2] **:** در آر.یو.پی، مفهوم نقش، ارتباط نزدیکی با فعالیت و دستاورد دارد. رفتارهای یک نقش بر اساسِ انجامِ فعالیت‌ها و مسئولیت‌های ان در رابطه با یکسری دستاورد تعریف می‌شود. توجه داشته باشید که نقش‌های تعریف شده در آر.یو.پی، ممکن است با ساختار سازمانی و احتمالاً نقش‌هایی که با آنها آشنا

[1] - Discipline
[2] - Role

می‌باشید، متفاوت باشدو البته، توجه داشته باشید که ایفای همه‌ی نقش‌های تعریف شده در آر.یو.پی، ممکن است ضرورتی نداشته باشد. یک نقش ممکن است بوسیله‌ی یک نفر یا چند نفر ایفا شود. یک نفر در طول یک پروژه می‌تواند در نقش‌های مختلفی ظاهر شود.

فعالیت[۱] **:** فعالیت عبارتست از واحدی از کار که از یک نقش، انجامِ آن، انتظار می‌رود. یک فعالیت دارای هدف مشخصی است و معمولاً برای تولید یا به‌روز رسانی یک دستاورد، مانند یک مدل، یک مؤلفه[۲] یا قسمتی از کدِ یک برنامه، انجام می‌شود. طولِ زمانی فعالیت‌ها، بین چند ساعت تا چند روز متغیر است. هر فعالیت به تعدادی **گام**[۳] شکسته می‌شود. مفهومِ فعالیت تا حدود زیادی معادلِ مفهومِ وظیفه[۴] می‌باشد.

دستاورد[۵] **:** یک دستاورد عبارتست از قطعه‌ای از اطلاعات که به وسیله‌ی یک یا چند فعالیت، تولید شده، بازبینی می‌شود و یا مورد استفاده قرار می‌گیرد. این قطعه‌ی اطلاعاتی می‌تواند یک سند، یک مدل، کد برنامه، یا فایل اجرایی باشد. توجه داشته باشید که تنها تعدادِ محدودی از دستاوردها شکل سند[۶] به خود می‌گیرند. با توجه به عواملی مانند ماهیت پروژه، ممکن است تعداد و ماهیت دستاوردهای مختلف تولید شده، متفاوت باشد. معمولاً توصیه می‌شود که دستاوردها را در شکل و قالب اصلی‌شان نگهداری نماییم. مثلاً برای نگهداری یک مدل بصری بهتر است از یک ابزار مدل‌سازی مناسب و برای تولید و نگهداری یک سند متنی مانند سند چشم‌انداز از یک نرم‌افزار واژه‌پرداز استفاده شود.

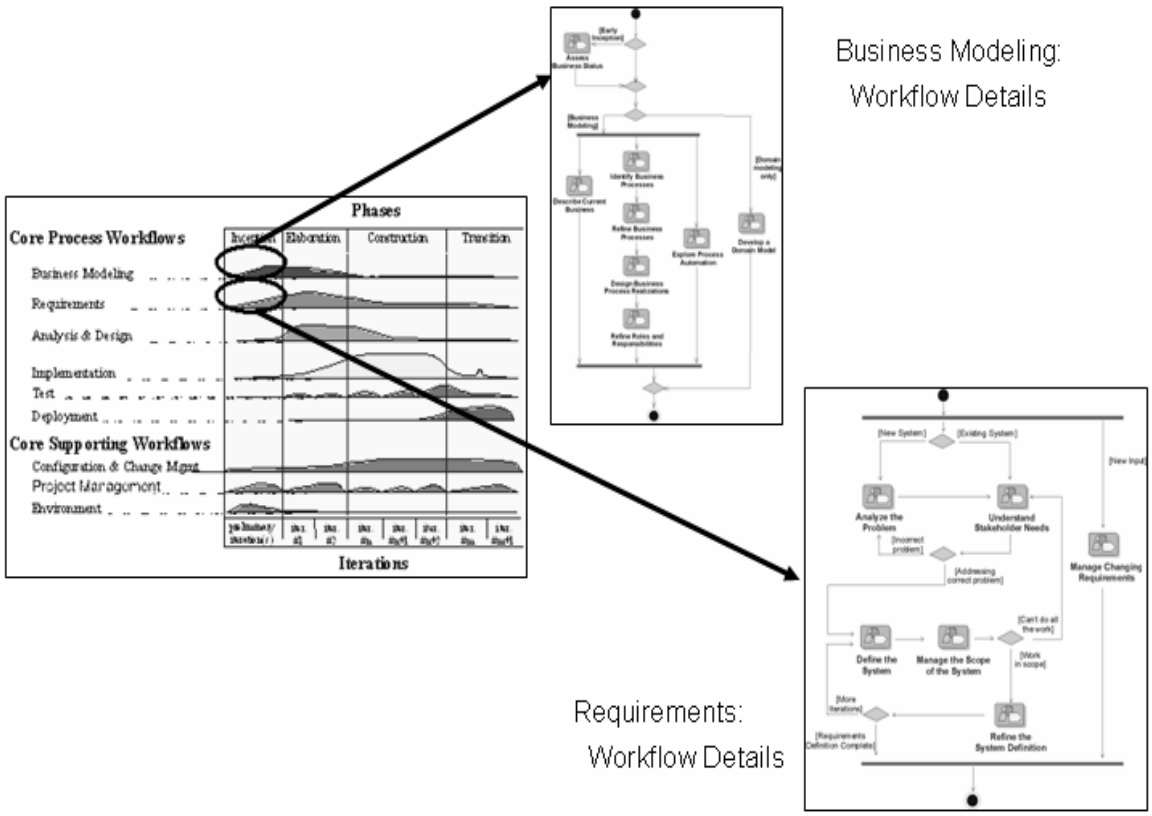
جریان‌کار[۷] **:** توالی معنادار و منطقی از فعالیت‌های مختلف است که منجر به ارائه‌ی نتیجه‌ای با ارزش و قابل توجه شده و در آن چگونگی تعامل میان نقش‌های مختلف توصیف می‌شود. دو شکل عمده از جریانِ‌کار در آر.یو.پی مطرح می‌باشد. به ازای هر یک از دیسیپلین‌های تعریف شده، مدلی از یک جریانِ‌کار ارائه شده است. این جریانِ‌کار که جریانِ‌کارِ اصلی[۸] نیز نامیده می‌شود، بیانگر توالی منطقی مجموعه‌ی فعالیت‌های یک دیسیپلین می‌باشد. از آنجایی که معمولاً فعالیت‌های متعددی در یک دیسیپلین وجود دارد، امکان مدل‌سازی

[1] - Activity
[2] - Component
[3] - Step
[4] - Task
[5] - Artifact
[6] - Document
[7] - Workflow
[8] - Core Workflow

آنها در قالب یک نمودار فعالیت، درک آن‌را با مشکل مواجه می‌کند. بنابراین، آر.یو.پی به منظور مدیریتِ پیچیدگی، با معرفی مفهومِ جزئیاتِ جریان‌کار[1] یا بسته‌های جزئی جریانِ کار، مجموعه‌ی فعالیت‌هایی که نوعی ارتباط منطقی نزدیک‌تری با هم دارند را در قالب یک بسته‌ی کوچک جریان‌کار دسته‌بندی می‌نماید. در شکل ۳-۸ ، ارتباط میان دیسیپلین‌های فرایند و جریان‌های‌کار متناظر با آن‌ها نشان داده شده است. شایان ذکر است که یکی از مهم‌ترین کاربردهای مدلِ جریانِ کار، برنامه‌ریزی تکرارها[2] در یک رویکرد تکرارشونده[3] می‌باشد. توجه داشته باشید که این جریانِ کارها، الگوهای ثابت و تغییر ناپذیری نیستند؛ شما می‌توانید بر حسب پروژه‌های مختلف و حتی پروژه‌های غیر نرم‌افزاری، گونه‌ی خاصی از جریانِ کار و نیز دیسیپلین‌های متفاوتی داشته باشید.

شکل ۳-۸

ارتباط میان دیسیپلین‌های فرایند و جریانِ‌کار متناظر با آن

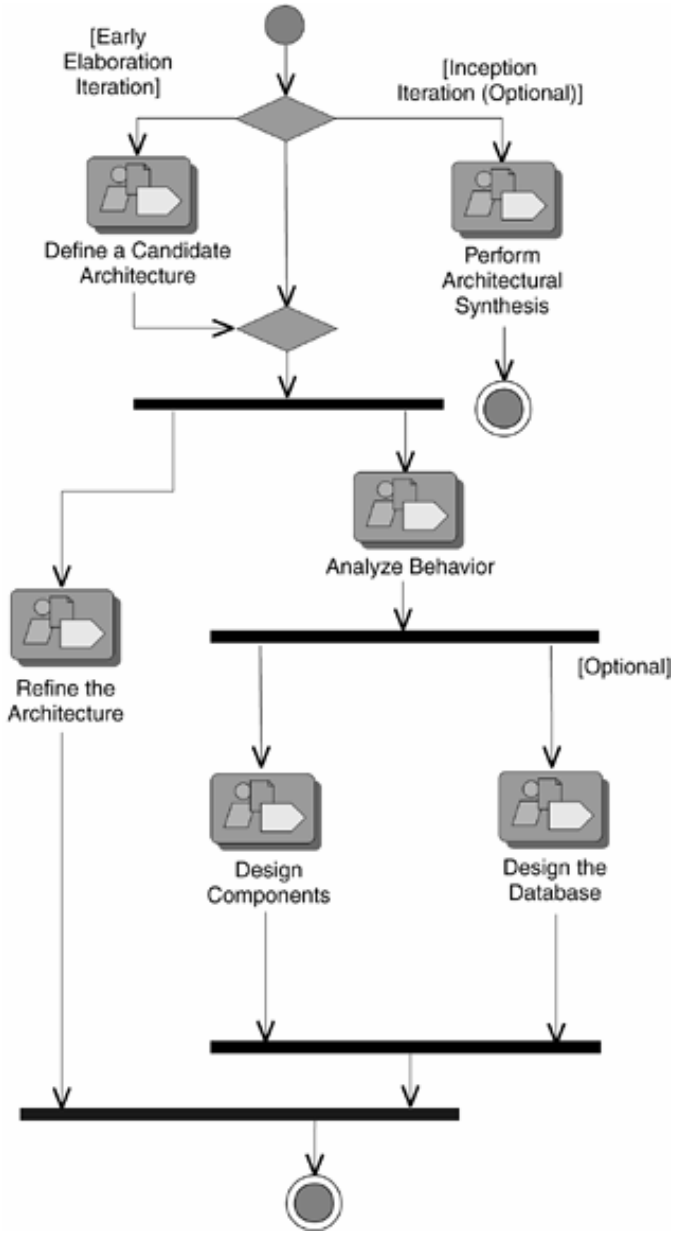


[1] - Workflow Details
[2] - Iteration
[3] - Iterative

در شکل ۳-۹ ، نمونه‌ای از یک جریان‌کار که متناظر با دیسیپلین تحلیل و طراحی می‌باشد، نشان داده شده است.

شکل ۳-۹

نمونه‌ای از مدل یک جریان‌کار متناظر با دیسیپلین تحلیل و طراحی در آر.یو.پی



سایر عناصرِ محتوایی آر.یو.پی، عبارتند از:

- **توصیه‌ها و راهنمایی‌ها**[1] : شامل اطلاعات و تجربیات ارزشمندی می‌باشند که در آنها برخی قوانین[2]، راهکارهای موفق، و یا روش‌های مکاشفه‌ای[3] برای انجام فعالیت‌ها و نیز ایجاد و به‌روزرسانی دستاوردها، ارائه شده است.
- **الگوها و قالب‌ها**[4] : آر.یو.پی، برای برخی از دستاوردهای مهم در فرایند تولید، قالب‌هایی را فراهم نموده است. این قالب‌ها که بر اساس تجارب موفق در پروژه‌های مختلف سازماندهی شده‌اند، بیانگر توصیه‌ها و راهنمایی‌هایی در رابطه با چگونگی سازماندهی مطالب و اطلاعاتِ مستند در یک دستاورد می‌باشد. توجه داشته باشید که این الگوها برای هر پروژه، به تناسبِ نیازها و ملاحظاتِ خاص آن، مستلزم پیکربندی و سفارشی‌نمودن است و بنابراین به هیچ عنوان اقدام به استفاده از همان شکل اولیه‌ی الگو و پرکردن آن ننمایید! قالب‌ها و الگوهای ارائه شده، تنها یک ذهنیت و دیدِ مناسب نسبت به ماهیت یک دستاورد ایجاد می‌نمایند.
- **راهنمای بکارگیری ابزار**[5] : این مؤلفه، معرفی‌کننده‌ی ابزارهای مختلف برای تسریع و تسهیلِ انجام فعالیت‌های تعریف شده می‌باشد. به طور پیش‌فرض، آر.یو.پی ابزارهای شرکت رَشنال را معرفی نموده است. توجه داشته باشید که این موضوع به معنای لزوم بکارگیری این ابزارهای خاص نمی‌باشد. هر سازمان و یا تیمی، با توجه به ملاحظاتی مانند سهولتِ استفاده، تطابق با سایر ابزارها و محیطِ تولید و همچنین هزینه‌ی مالکیت ابزار که شامل هزینه‌های تهیه، آموزش، به‌روز رسانی و نگهداری می‌باشد، به انتخاب ابزارهای مناسب اقدام می‌نماید. توجه داشته باشید که راهنمای بکارگیری ابزار فقط به معرفی چگونگی بکارگیری یک ابزار برای انجام یک فعالیت خاص می‌پردازد.

---

[1] - Guidelines
[2] - Rules
[3] - Heuristic
[4] - Template
[5] - Tool Mentor

- **مفاهیم**[۱] **:** عبارتست از بیانِ تعاریف و اصولِ مهم و کلیدی مرتبط با فعالیت‌ها، نقش‌ها، و یا دستاوردهای مختلف. در آر.یو.پی، توضیحات نسبتاً جامع و مفصلی برای بسیاری از مفاهیم ذکر شده در دیسیپلین‌ها، نقش‌ها، فعالیت‌ها، و دستاوردها ارائه شده است.
- **نقشه‌های راه**[۲] **:** به منظور راهنمایی استفاده‌کنندگان در بکارگیری آر.یو.پی از یک منظر خاص و یا یک کاربرد خاص، تعدادی به اصطلاح، نقشه‌ی راه فراهم شده است. به عنوان مثال، یک نقشه‌ی راه به منظور راهنمایی درباره‌ی چگونگی پیکربندی محتوای آر.یو.پی برای پروژه‌های کوچک ارائه شده است. نقشه‌ی راه دیگری، نشان دهنده‌ی چگونگی بکارگیری آر.یو.پی در پروژه‌های کسب و کار الکترونیکی[۳] می‌باشد.

در آر.یو.پی دیسیپلین‌های موجود به دو دسته تقسیم شده‌اند. دسته‌ی اول، شامل دیسیپلین‌هایی است که ارتباط مستقیمی با شکل‌گیری ماهیتِ فراورده‌ی نرم‌افزاری دارند. این دیسیپلین‌ها را دیسیپلین‌های اصلی[۴] می‌نامند. این دیسیپلین‌ها عبارتند از:

- دیسیپلین مدل‌سازی سازمان[۵] (یا دیسیپلین مدل‌سازی کسب و کار)
- دیسیپلین نیازمندی‌ها[۶] (یا دیسیپلین مدیریت نیازمندی‌ها)
- دیسیپلین تحلیل و طراحی[۷]
- دیسیپلین پیاده‌سازی[۸]
- دیسیپلین تست[۹] (یا دیسیپلین ارزیابی[۱۰])
- دیسیپلین استقرار[۱۱]

---

[1] - Concepts
[2] - Roadmaps
[3] - e-Business
[4] - Core Disciplines
[5] - Business Modeling
[6] - Requirements
[7] - Analysis and Design
[8] - Implementation
[9] - Test
[10] - Assessment
[11] - Deployment

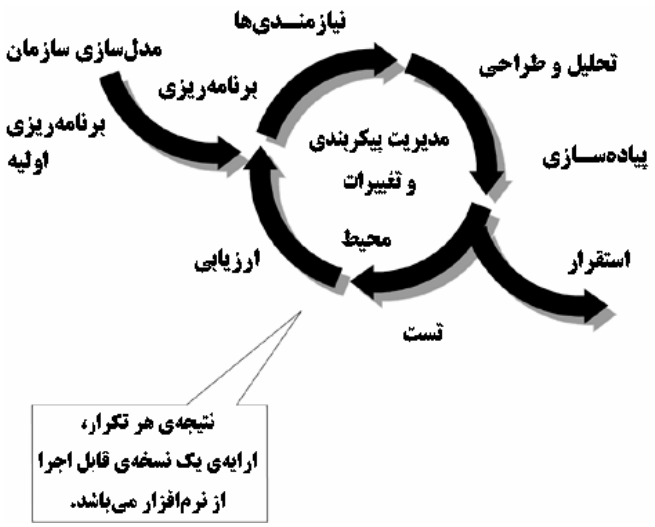
همانگونه که ملاحظه می‌نمایید، نوعی رابطه‌ی منطقی و ترتیبی میان این دیسیپلین‌ها وجود دارد. در واقع، این دیسیپلین‌ها دنباله و زنجیره‌ی مجموعه‌ی فعالیت‌ها، دستاوردها، و نقش‌های مطرح در الگوی متوالیِ[1] فرایند تولید یا همان الگوی آبشاری[2] می‌باشند.

دسته‌ی دوم، شامل سه دیسیپلین می‌باشد که عمدتاً نقش پشتیبانی و مدیریتی دارند. این سه دیسیپلین را که دیسیپلین‌های پشتیبانی[3] نیز می‌نامند، عبارتند از:

- دیسیپلین مدیریت پروژه[4]
- دیسیپلین مدیریت تغییرات و پیکربندی[5]
- دیسیپلین محیط[6] (یا دیسیپلین مدیریت محیط)

شکل ۳-۱۰

یک تکرار در چرخه‌ی تولید شامل تمام دیسیپلین‌ها می‌باشد.



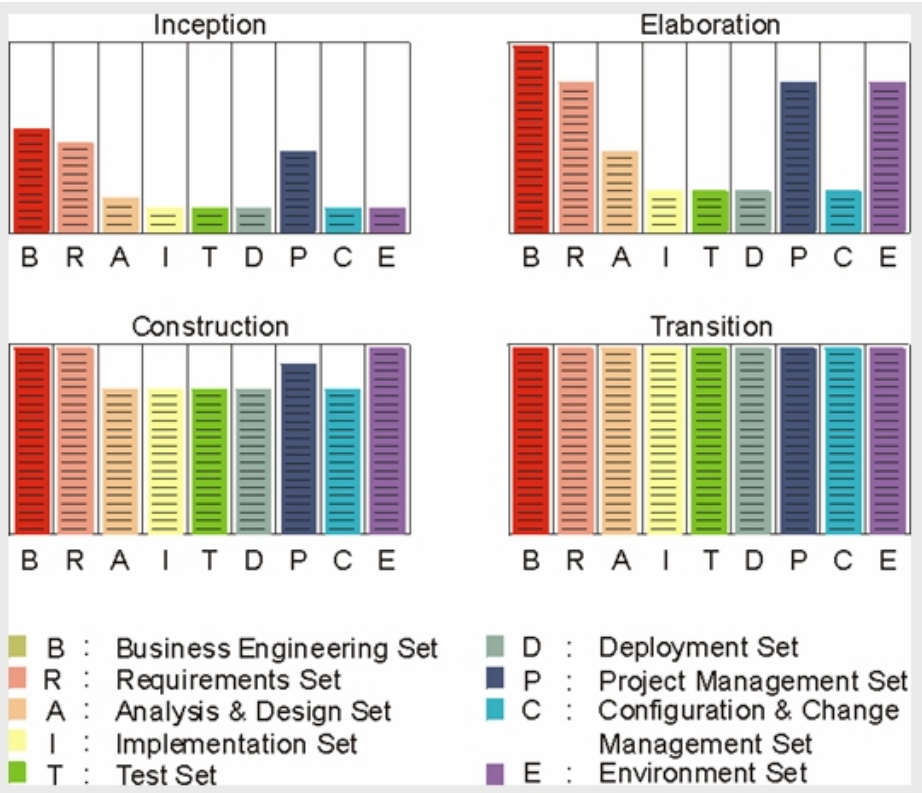
مجموعه‌ی فعالیت‌ها، دستاوردها، و نقش‌های مختلف بر حسب ماهیت و هدفی که برای آنها تعیین‌شده است، تقریباً در دیسیپلین‌های مشخص و متمایزی قرار می‌گیرند و در واقع، هیچ دستاورد، فعالیت، و یا نقشی

[1] - Sequential
[2] - Waterfall
[3] - Supporting Disciplines
[4] - Project Management
[5] - Change & Configuration Management
[6] - Environment

میان دیسیپلین‌های مختلف، مشترک نمی‌باشد. البته، این موضوع بیشتر در رابطه با دیسیپلین‌های اصلی، مصداق دارد.

شکل ۳-۱۱

مجموعه‌ی دستاوردها متناظر با مجموعه‌ی دیسیپلین‌ها



توجه داشته باشید که این دیسیپلین‌ها و جریان فعالیت‌هایی که متناظر با هر یک از آنها توسط آر.یو.پی مطرح شده است، مفاهیم ثابت و بدون تغییری نیستند. همانگونه که پیش از این نیز اشاره شد، دیسیپلین، چیزی نیست جز یک ظرف منطقی که مجموعه‌ی مؤلفه‌ها و عناصر محتوایی فرایند را سازماندهی می‌نماید. ممکن در شرایط مختلف بر حسب نوع پروژه و ماهیت فعالیت‌های آن و یا در سازمان‌های مختلف، با طیف متفاوتی از دیسیپلین‌ها برخورد داشته باشیم. البته، لازم نیست که فعلاً به این موضوع فکر کنید. بحث در این رابطه از محدوده‌ی این کتاب خارج است؛ فقط در خاطر داشته باشید که در پروژه‌های مختلف ممکن است بطور کلی ماهیت این سازماندهی‌های ارائه شده (در قالب نُه دیسیپلین در نسخه‌ی پایه‌ی آر.یو.پی) متفاوت باشد.

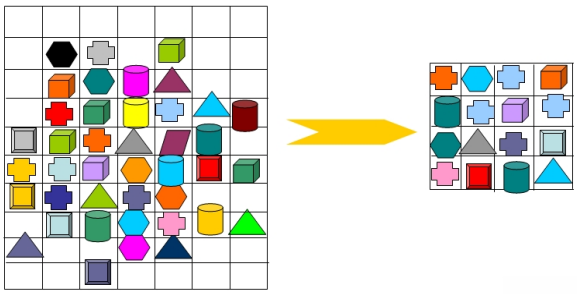
آر.یو.پی به عنوان محصولی دربرگیرنده‌ی چارچوبِ فرایند

هر سازمان و یا پروژه‌ی خاص، نیازمند به داشتن فرایند خاصی متناسب با نیازها، ضروریات، و ملاحظات مختصِ خودش می‌باشد. بنابراین، فرایند تولید در پروژه‌های مختلف، متفاوت است و یک فرایند مناسب، باید دارای قابلیت تطبیق با شرایط و موقعیت‌های خاصِ هر پروژه و یا سازمان باشد. یکی از ویژگی‌های برجسته و کم‌نظیر آر.یو.پی این است که آر.یو.پی محصولی است که چارچوب کاملی را برای الگوبرداری و تعریف فرایندهای مختلف در طیف گسترده‌ای از پروژه‌های با ابعاد متنوع (کوچک، متوسط و بزرگ) و در زمینه‌ها و موضوعات مختلف (حتی پروژه‌های غیر نرم‌افزاری) فراهم می‌آورد.

در واقع، آر.یو.پی یک فرایند نیست که بتوان آن‌را به همان شکلی که ارایه شده، در یک پروژه بکار گرفت. آر.یو.پی گنجینه‌ای است غنی از راهکارهای موفق برای گستره‌ی وسیعی از پروژه‌ها. بنابراین، آر.یو.پی یک بانک دانش[1] و چارچوب فرایند[2] است نه صرفاً یک فرایند.

شکل ۳-۱۲

چارچوب فرایند (سمت چپ) و فرایند حاصل از آن (سمت راست)



اجزاء و مؤلفه‌های اصلی این محصول، عبارتند از:

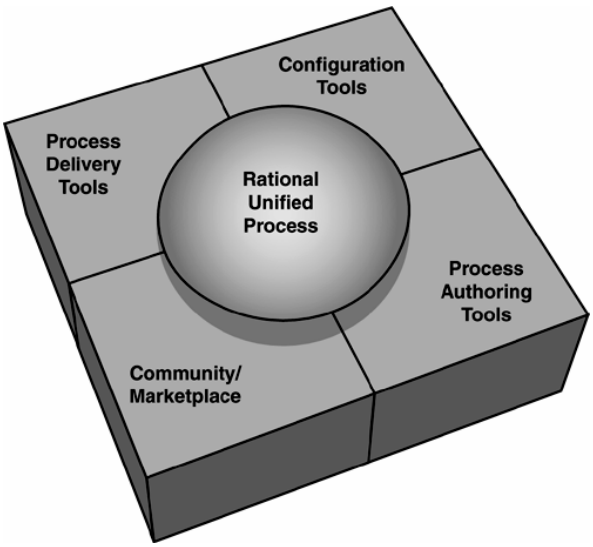
- راهکارهای موفق
- ابزارهای تحویل فرایند[3]

[1] - Knowledge-base
[2] - Process Framework
[3] - Process Delivery Tools

- ابزارهای پیکربندی[1] فرایند
- ابزارهای تألیف[2] فرایند
- اجتماع و گستره‌ی وسیعی از استفاده‌کنندگان[3]

شکل ۳-۱۳

مؤلفه‌ها و اجزاء مختلف آر.یو.پی به عنوان یک محصول



آر.یو.پی به عنوان یک محصول با ماهیت نرم‌افزاری، تقریباً هر شش ماه یک بار، به‌روز رسانی می‌شود. همانگونه که پیش از این عنوان شد، در حال حاضر، با ادغام شرکت رَشنال در شرکت آی.بی.اِم، مالکیت این محصولِ نرم‌افزاری، برعهده‌ی شرکت آی.بی.اِم می‌باشد.

ابزارهای تألیف[4] و پیکربندی[5] آر.یو.پی

آر.یو.پی فرایندی است که نمی‌توان آن‌را به همان صورتی که هست، در یک پروژه بکار گرفت. متأسفانه در طول سال‌های اخیر، سازمان‌ها و تیم‌های زیادی در کشور ما به این نکته توجه نداشته و سعی نموده‌اند که مثلاً تمام دستاوردهای مطرح در آر.یو.پی را به همان شکل اولیه‌ی آنها تولید نموده و فعالیت‌های مطرح شده

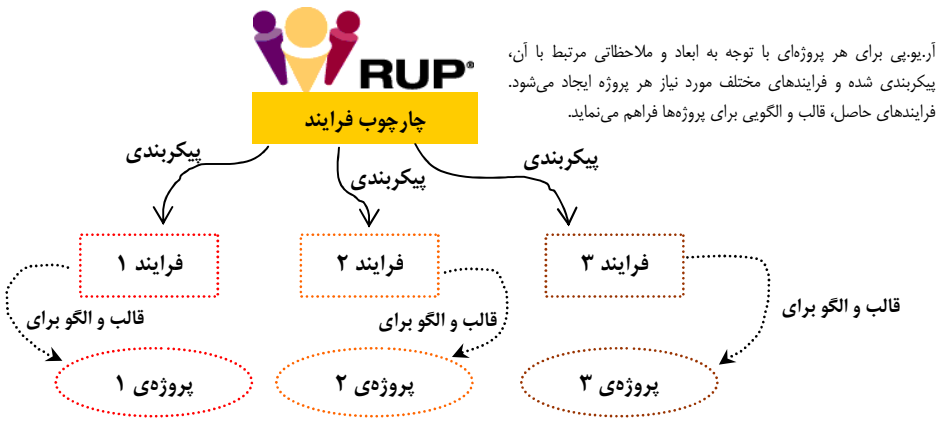
[1] - Configuration Tools
[2] - Authoring Tools
[3] - Users Community
[4] - Authoring
[5] - Configuration

را انجام دهند. این کار تنها موجب انجام کارهای اضافی و زائد شده و در پایان، با مجموعه‌ی حجیمی از دستاوردهایی غیر ضروری با ارزش افزوده‌ی بسیار کم، حاصل می‌شود.

در اینجا، دوباره تأکید می‌نماییم که هیچ پروژه و سازمانی در دنیا نمی‌تواند (و اصولاً این کار کاملاً از منطق مهندسی و اقتصادی به دور است که) آر.یو.پی را همان‌گونه که در قالب محصول نرم‌افزاری ارائه شده است، بکار بگیرد. پیکربندی آر.یو.پی و به اصطلاح دوخت و دوز[1] مناسب آن با توجه به اندازه‌ها، نوع و ملاحظات پروژه و سازمان بسیار ضروری است. نتیجه‌ی این کار، فرایند موردِ نیاز در سطحِ پروژه، سازمان، و یا در طیفی از پروژه‌های خاص می‌باشد. شکل ۳-۱۴ بیانگر همین موضوع است.

شکل ۳-۱۴

ارتباط میان آر.یو.پی به عنوان یک چارچوبِ فرایند با فرایندهای مورد نیاز هر پروژه



توجه داشته باشید که تجارب موفق و دستاوردهای کنونی سازمان نیز نقش بسیار مهمی در چگونگی پیکربندی و سفارشی‌سازی فرایند، ایفا می‌نمایند. در واقع، هر سازمان با توجه به تجارب موفق و قابلیت‌های خود، پیکربندی خاصی از آر.یو.پی را برای هر یک از پروژه‌هایش فراهم می‌نماید.

مسلماً، وقتی می‌توان انتظار داشت که آر.یو.پی یک چارچوب فرایند، با قابلیت پیکربندی در طیف وسیعی از پروژه‌ها و سازمان‌ها باشد که روز به روز غنی‌تر شده و بر گنجینه‌ی راهکارها و تجارب موفق موجود در آن

[1] - Tailoring

افزوده شود. بنابراین، به هیچ عنوان از بزرگ بودنِ حجمِ آر.یو.پی واهمه‌ای نداشته باشید و حتی باید سعی نماییم که در غنی‌تر نمودنِ آن نقش داشته باشیم. در اینجا لازم است دو نکته‌ی مهم را به یاد داشته باشید:

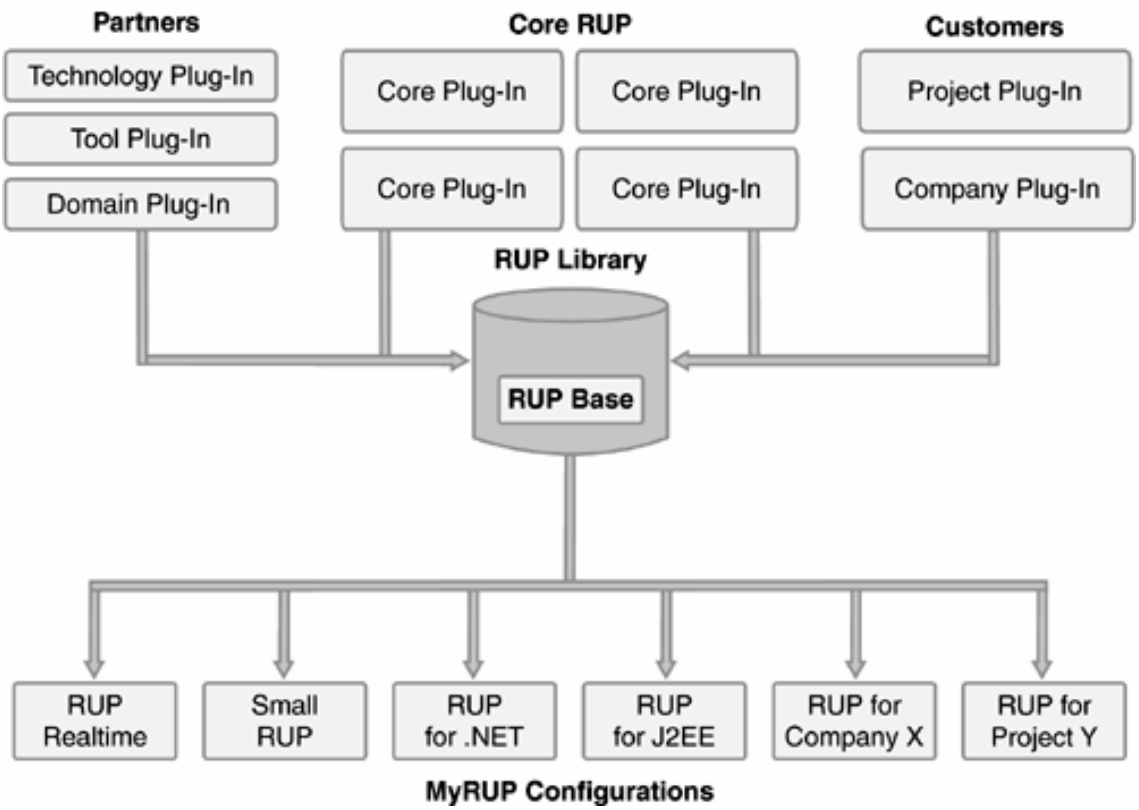
- نخست آنکه، ویژگی فراورده‌ی نرم‌افزاری بودنِ آر.یو.پی (ارائه شده به صورتِ یک فراورده‌ی نرم‌افزاری مبتنی بر فناوری وِب[1]) و نیز اینکه ساختار و محتوای آن به خوبی تعریف و سازماندهی شده است، امکان به‌روز رسانی، گسترش، و پیکربندی آن‌را تسهیل نموده است. تصور نمایید که اگر آر.یو.پی مانند بسیاری از فرایندهای مطرحِ امروزی، در قالب یکسری مستندات و کتاب ارائه می‌شد، پیکربندی، توسعه، و به‌روز رسانی آن تا چه حد مشکل می‌بود.
- نکته‌ی دوم آنکه، بسیاری از کارشناسان و سازمان‌ها (خصوصاً در کشور خودمان) آر.یو.پی را یک فرایند حجیم و قابل استفاده در تنها طیف خاصی از پروژه‌ها (احتمالاً متناسب با حجم آر.یو.پی!) تصور می‌کنند. نکته‌ی مهم اینست که آر.یو.پی اصلاً یک فرایند نیست! آر.یو.پی چارچوب و گنجینه یا به اصطلاح بانک دانشی[2] است حجیم و قابل پیکربندی برای گستره‌ی وسیعی از پروژه‌ها و سازمان‌ها. این قابلیت تنها وقتی امکان پذیر می‌باشد که آر.یو.پی طیف بسیار متنوع و حجیمی از دانش، تجربه، و راهکارهای موفق را در خود داشته باشد و روز به روز نیز بر حجم و محتوای این بانک دانش، افزوده شود.

در شکل ۳–۱۵ ، نمایی از روش‌های مختلف پیکربندی آر.یو.پی[3] نشان داده شده است. همانگونه که در این شکل مشخص است، آر.یو.پی به عنوان یک چارچوبِ فرایند[4] از مؤلفه‌های مختلفی تشکیل شده است. هسته‌ی اصلی آر.یو.پی از ماجول‌های مستقلی تشکیل شده که قابلیت پیکربندی آن‌را تسهیل می‌نماید. در هنگام پیکربندی، ماجول‌های دیگری را نیز که ممکن است اختصاص به یک تکنولوژی، ابزار، شرکت، و یا دامنه‌ی کاربرد خاص داشته باشند، می‌توان به این چارچوب فرایند، اضافه نمود.

[1] - Web-based
[2] - Knowledge-base
[3] - RUP Configuration
[4] - Process Framework

شکل ۳-۱۵

نمایی از روش‌های مختلف پیکربندی آر.یو.پی

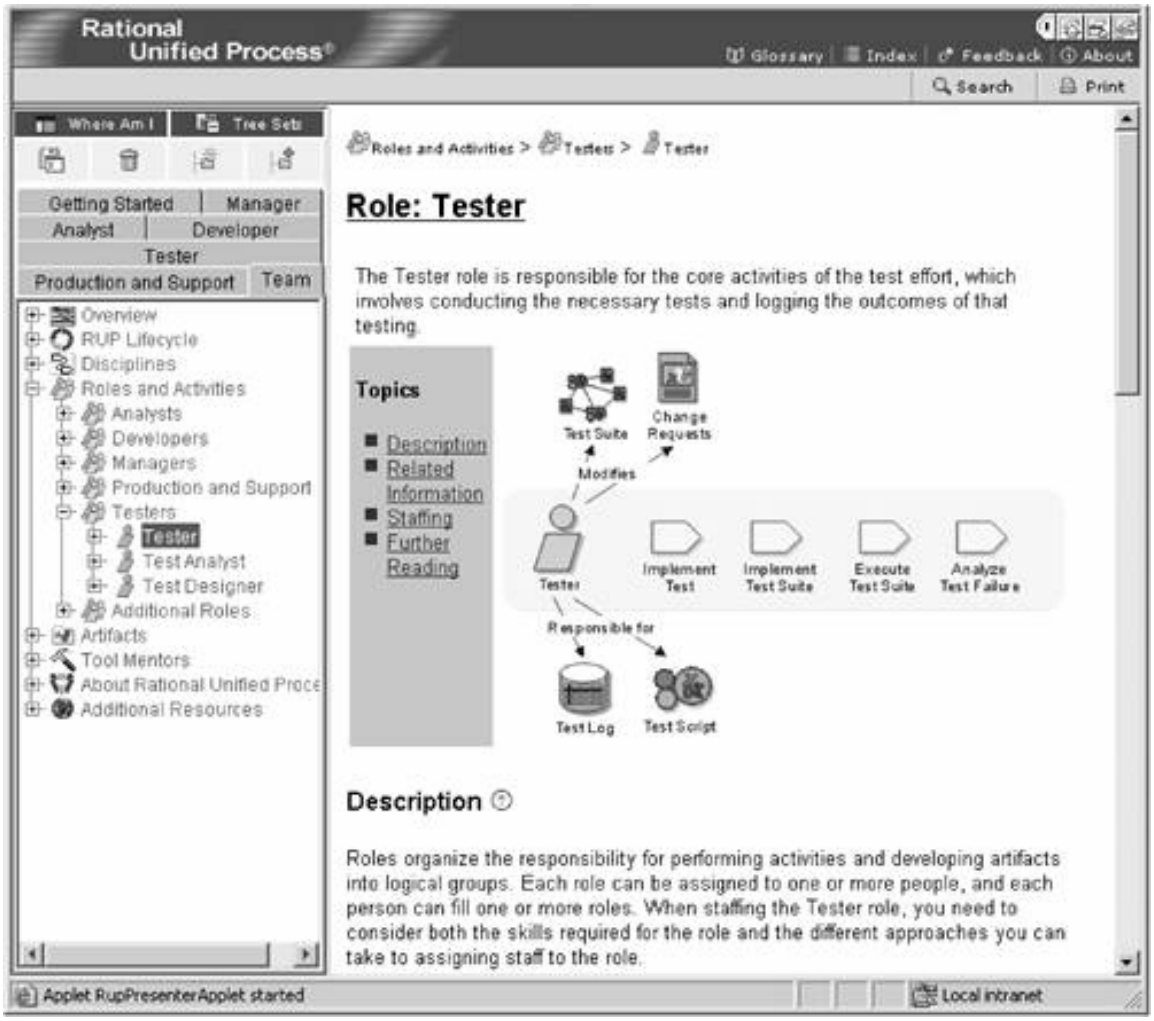


در هر پیکربندی[1] از آر.یو.پی، می‌توان منظرهای مختلفی از فرایند[2] را ایجاد نمود. این منظرها، افراد مختلف در یک پروژه یا سازمان را قادر می‌سازد که بتوانند پیکربندی ارائه شده از فرایند را از جنبه‌ی مسائل و ملاحظات مربوط به نقش و مسئولیت‌های مربوط به خودشان، سازماندهی نمایند. آر.یو.پی به طور پیش‌فرض منظرهایی را فراهم نموده‌است که قابلِ تغییر می‌باشند.

[1] - Configuration
[2] - Process Views

شکل ۳-۱۶

نمایی از واسط کاربرِ آر.یو.پی: در گوشه‌ی فوقانی و در سمت چپ، منظرهای مختلف فرایند قابل مشاهده است.



برخی از شرکت‌ها به کمک ابزار تألیف فرایند[1] آر.یو.پی، دانش فنی خود درباره‌ی ملاحظات یک فناوری، ابزار یا دامنه‌ی جدید را به صورت یک ماجول بسته‌بندی نموده و آن را در اختیار سایر استفاده‌کنندگان از آر.یو.پی قرار می‌دهند. تشریح چگونگی پیکربندی آر.یو.پی در حیطه‌ی این کتاب نیست. این موضوع را در کتاب دیگری مفصلاً بررسی نموده‌ایم.

[1] - RUP Process Authoring Tool

ابزارهای تحویل و ارائه‌ی فرایند[1]

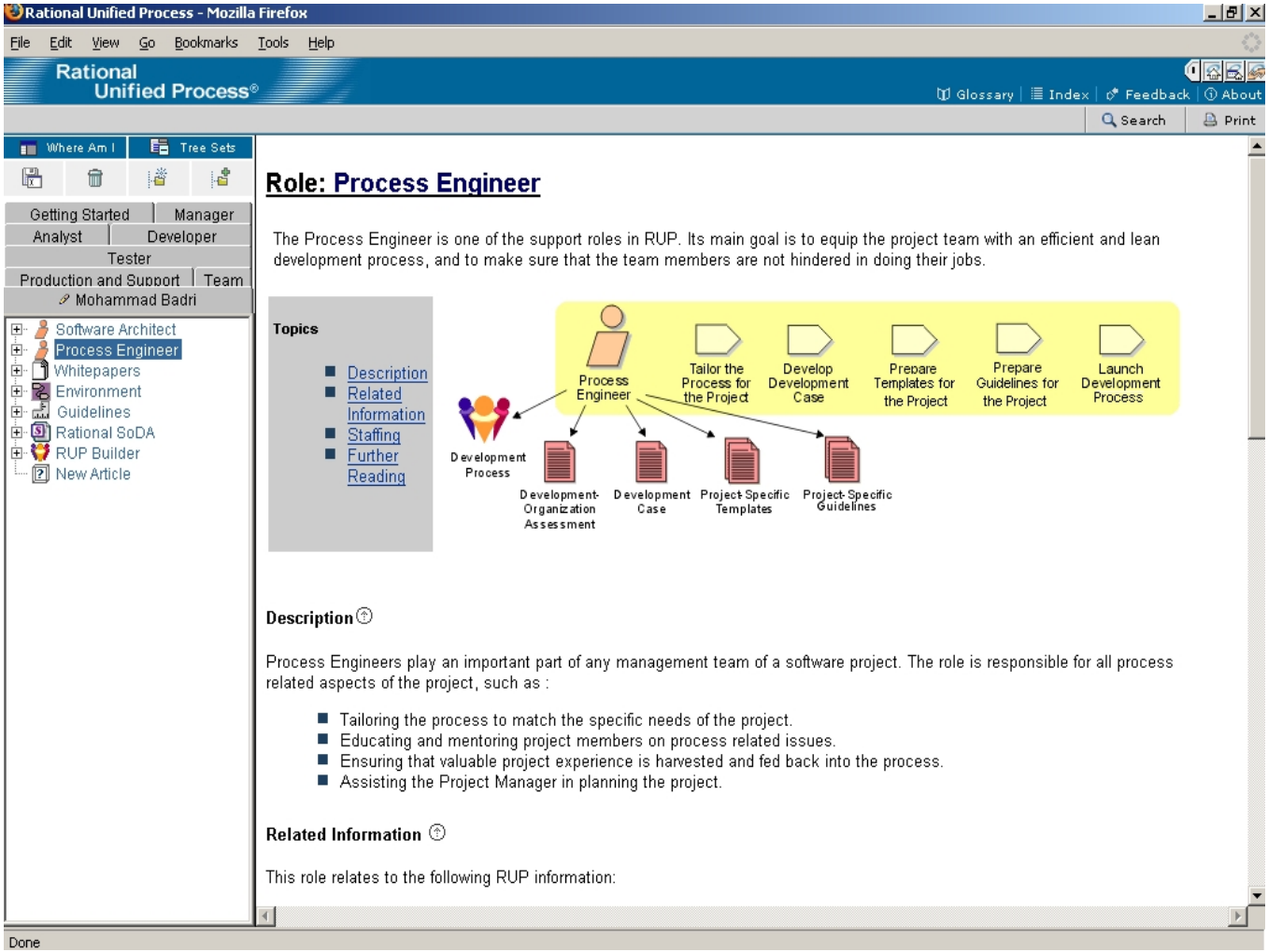
آر.یو.پی با کمک مکانیزمی تحت عنوان myRUP ، راهنمای آموزشی ابزارها[2] و نیز راهنمای گسترش‌یافته[3] ، کاربران را در بکارگیری بهینه‌تر و آسان‌تر فرایند یاری می‌دهد. با کمک راهنمای آموزشی ابزارها و راهنمای گسترش‌یافته که در داخل ابزارهای مختلف تعبیه شده است، موجبات تسهیل پیاده‌سازی و بهره‌گیری از فرایند فراهم می‌گردد.

با کمک مکانیزم myRUP ، شما می‌توانید منظرهایی از فرایند را بر حسب نیاز و انتظارات شخصی یا تیمیِ خود، به صورت سفارشی ایجاد نمایید. با کمک این قابلیت، امکان بهره‌گیری از مؤلفه‌های فراهم شده توسط آر.یو.پی و نیز امکان افزودن سایر منابع اطلاعاتی خاص به منظرِ ایجاد شده، وجود خواهد داشت. در شکل ۳–۱۷ ، نمونه‌ای از یک منظر ایجاد شده در آر.یو.پی، نشان داده شده است.

[1] - Process Delivery Tools
[2] - Tool Mentors
[3] - Extended Help

شکل ۳-۱۷

نمونه‌ای از یک منظرِ فرایند سفارشی شده در آر.یو.پی



راهنمای آموزشِ ابزار[1]

بیشتر مباحث مطرح در آر.یو.پی مستقل از ابزار می‌باشد. با این وجود، انجام بسیاری از فعالیت‌ها به منظور تسریع و بهبودِ فرایند[2]، مستلزم بکارگیری ابزارهای مناسب می‌باشد. بدیهی است نقش‌های انجام‌دهنده‌ی این دسته از فعالیت‌ها، باید درباره‌ی قابلیت‌های مورد انتظار و نیز چگونگی کار با ابزارهای مختلف اطلاعاتی داشته باشند. راهنمای آموزشِ ابزار مانند یک مربی[3]، گام به گام چگونگی استفاده از یک ابزار خاص را نشان

[1] - Tool Mentor
[2] - Process Improvement
[3] - Mentor

می‌دهد. به‌طور پیش‌فرض، آر.یو.پی ابزارهای کمک به مهندسی نرم‌افزارِ[1] شرکت رَشنال را معرفی نموده است. البته، این بدان معنا نیست که الزاماً باید از این ابزارهای استفاده شود؛ هر سازمان و یا تیمی، به تناسب نیاز و شرایط خاصِ خود، به راحتی و بدون نگرانی از ملاحظات فرایند، می‌توانند ابزارهای دیگری را انتخاب و در پیکربندی آر.یو.پی وارد نماید.

نکته‌ی مهمی که یادآوری آن در اینجا ضروری است اینکه ابتدا باید به فرایند و فعالیت‌های ضروری آن توجه داشته باشیم. پس از درک منطقِ فرایند و یادگیری فعالیت‌های ذکر شده در آن، جایگاه ابزار را در رابطه با تسریع، بهینه‌سازی، و بهبودِ فرایند، درک نماییم. بنابراین تا جایگاهِ بکارگیری یک ابزار به خوبی درک نشده، به هیچ وجه به سراغ آن نخواهیم رفت.

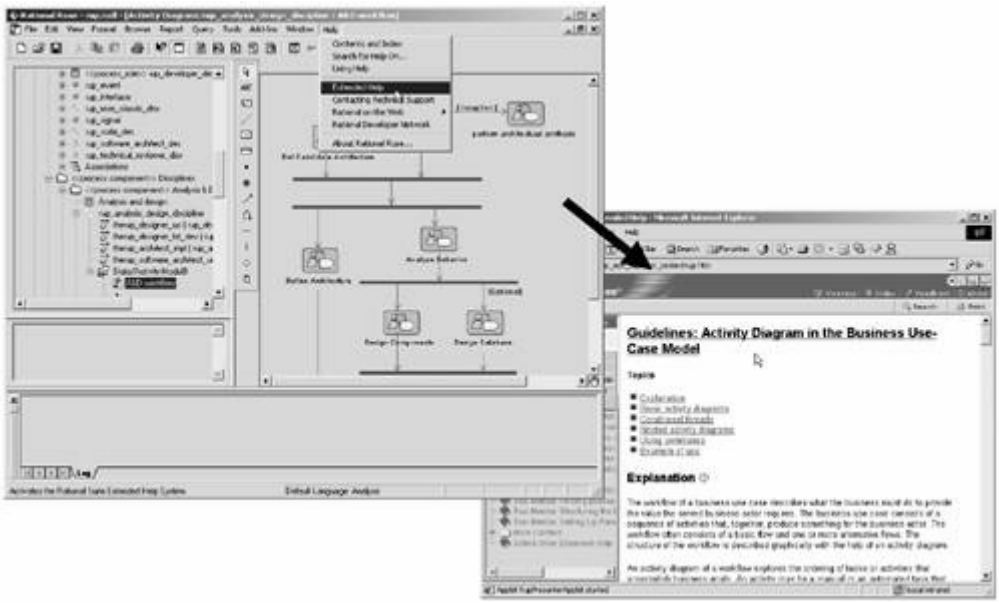
راهنمای گسترش‌یافته

راهنمای گسترش‌یافته[2] فراهم‌کننده‌ی امکانِ گرفتن راهنمایی به صورتِ به اصطلاح حساس به زمینه[3] ، در ابزارهای مختلف می‌باشد. برای مثال، هنگامی‌که در ابزاری مانند ابزار رَشنال رُز[4] ، روی یک کلاس کار می‌کنید و می‌خواهید بدانید که بعد از تکمیل آن چه کاری را باید انجام دهید، می‌توانید از راهنمای گسترش‌یافته‌ی آر.یو.پی در این ابزار استفاده نمایید. با این کار، لیستی از موضوعات مرتبط در فرایند به شما نشان داده خواهد شد. شکل ۳–۱۸، نشان می‌دهد که این نوع راهنما در یکی از ابزارهای مهندسی نرم‌افزار شرکت رَشنال (یعنی ابزار رُز) استفاده شده است. البته، این موضوع بیشتر در رابطه با ابزارهای شرکت رَشنال مصداق دارد.

[1] - CASE Tools
[2] - Extended Help
[3] - Context-Sensitive
[4] - Rational Rose

شکل ۳-۱۸

استفاده از راهنمای گسترش‌یافته‌ی آر.یو.پی در سایر ابزارها



چه‌کسانی از آر.یو.پی استفاده می‌نمایند؟

متأسفانه آمار دقیقی از تعداد استفاده‌کنندگان و مشتریان آر.یو.پی در دست نمی‌باشد. بر اساس آخرین اطلاعات موجود، در سال ۲۰۰۳ میلادی، بیش از ده هزار شرکت و سازمان در سرتاسر جهان از محصول آر.یو.پی استفاده می‌کردند. البته این اعداد و ارقام، بیانگر تعداد رسمی استفاده‌کنندگان می‌باشد. اما تعداد بسیار زیادی از شرکت‌ها و موسسات کوچک و متوسط، به صورت غیر رسمی (بدون خرید محصولِ آر.یو.پی و یا مشارکت رسمی در دوره‌های تخصصی شرکت رَشنال) از این فرایند استفاده می‌نمایند. مجموعِ آمارهای رسمی و غیر رسمی، نشان می‌دهد که تا انتهای سال ۲۰۰۳ ، تعداد استفاده‌کنندگان از آر.یو.پی در جهان، بیش از بیست‌هزار مورد می‌باشد.

این شرکت‌ها و سازمان‌ها، آر.یو.پی را به منظور تولید سیستم‌ها و نرم‌افزارهای کاربردی در زمینه‌ها و حوزه‌های مختلف و در طیف وسیعی از پروژه‌های بزرگ و کوچک بکار می‌گیرند. برخی از همین شرکت‌ها، آر.یو.پی را در توسعه‌ی سیستم‌های غیر نرم‌افزاری بکار گرفته‌اند.

برخی از مهم‌ترین صنایع استفاده‌کننده از آر.یو.پی عبارتند از:

- صنعت مخابرات[۱]
- صنعت حمل و نقل[۲]، هوافضا و صنایع دفاعی
- صنعت ساخت و تولید[۳]
- خدمات مالی
- یکپارچه‌سازی سیستم‌ها

پذیرش آر.یو.پی[۴] در حوزه‌ی وسیعی از صنایع در طی سال‌های گذشته، بیانگر نوعی تغییر و تحوّل جالب توجه در صنعت نرم‌افزار می‌باشد. با افزایش فشار زمانی ورود به بازار و همچنین تقاضاهای روزافزون برای تولید سیستم‌های کاربردی با کیفیتِ برتر[۵]، شرکت‌های مختلف را به سمت بهره‌گیری بیش از پیش از تجارب و راهکارهای موفق[۶] سوق داده است؛ آر.یو.پی نیز گنجینه‌ای است از راهکارها و تجارب موفق.

برخی از شرکت‌ها و سازمان‌های پذیرنده‌ی آر.یو.پی، بر اساس نوع و ماهیت پروژه‌هایشان، پیکربندی خاصی از فرایند را بر اساس آر.یو.پی ایجاد نموده‌اند. برخی دیگر، آر.یو.پی را به همان شکل اصلی و رسمی[۷] آن و با کمی دوخت و دوز، در هر یک از پروژه‌های خود، بکار می‌گیرند. در این میان، شرکت‌ها و سازمان‌هایی هم وجود دارند که آر.یو.پی را به صورتی غیرِ رسمی[۸] و عمدتاً به شکل گنجینه و مخزنی از راهکارها و تجارب موفق، توصیه‌ها، راهنمایی‌ها، و نیز قالب‌هایی برای تعریف دستاوردها، بکار می‌گیرند. برخی دیگر، از آر.یو.پی به عنوان راهنمای مسیر بهبود فرایند و برای رسیدن به سطوح بالای بهره‌وری فرایند تولید، استفاده می‌کنند.

---

[1] - Telecommunication
[2] - Transport
[3] - Manufacturing
[4] - RUP Acquisition
[5] - High-quality application
[6] - Best Practices
[7] - Formal
[8] - Infomal

تاریخچه‌ی مختصری از آر.یو.پی

همانگونه که در ابتدای این فصل نیز بیان شد، آر.یو.پی نتیجه‌ی گردآوری تجاربِ موفقِ شرکت‌ها و تیم‌های زیادی در طول چندین دهه می‌باشد. برخی از ریشه‌های اصلی آر.یو.پی به مدل حلزونی[۱] ارائه شده توسط آقای بِری بوهم[۲]، برمی‌گردد. این مدل، با داشتن رویکردی تکرارشونده[۳] و مشتق از ریسک[۴]، کارشناسان برجسته‌ی شرکت رَشنال، یعنی افرادی نظیر فیلیپْ کراچِن[۵] ، گْرَدی بووچ[۶] ، مایکْ دوْلین[۷] ، ریچْ رِیْتْمَن[۸] و وُکِر رُیسْ[۹] ، را تحت تأثیر قرار داد.

رُیسْ و بوهم در زمینه‌ی تحقیقات و نیز یکسری پروژه، همکاری‌های مشترکی داشتند. یکی از کارهای موفق و شناخته‌شده‌ی آقای بوهم، مفهوم نقاط لنگرگاه[۱۰] می‌باشد که ایشان در سال ۱۹۹۶ در مقاله‌ای آن را معرفی نمودند. همین مفهوم، امروزه در آر.یو.پی تحت عنوان نقاط تصمیم‌گیری سازمانی[۱۱] یا نقاط کلیدی اصلی[۱۲] مطرح می‌باشد.

تیمِ مستقر در شرکت رَشنال، آر.یو.پی را در مشارکت با مشتریان و با شروع از فرایندی تحت عنوانِ رویکردِ رَشنال[۱۳] که در طی سال‌های دهه‌ی هشتاد و ابتدای دهه‌ی نود میلادی در این شرکت تهیه شده بود، ایجاد نمود. این تیم از فرایندی تحت عنوان آبْجِکتُری[۱۴] که متعلق به آقای ایوار جَکوبْسُنْ[۱۵] می‌باشد نیز در ایجاد و تکمیل آر.یو.پی، استفاده نمود. ایشان نیز بعدها (در سال ۱۹۹۵) به شرکت رَشنال پیوست. بخش اصلی پروژه‌ی ایجاد آر.یو.پی در سال‌های ۱۹۹۵ تا ۱۹۹۸ میلادی انجام شد. معمار ارشدِ[۱۶] آر.یو.پی، آقای

---

1 - Spiral
2 - Barry Boehm
3 - Iterative
4 - Risk Driven
5 - Philippe Kruchten
6 - Grady Booch
7 - Mike Devlin
8 - Rich Reitman
9 - Walker Royce
10 - Anchor Point
11 - Business Decision Point
12 - Major Milestone
13 - Rational Apporach
14 - Objectory
15 - Ivar Jacobson
16 - Chief Architect

فیلیپْ کراچِن بود. ایشان شخصی بسیار فعال و با تجربه در معماری و راهبری فرایند در سیستم‌های بزرگی مانند سیستم جدید کنترل ترافیک در کانادا و نیز یکی از متفکرین برجسته در زمینه‌ی معماری می‌باشد.

با وجودیکه از همان ابتدای کار ایده‌ی ایجادِ آر.یو.پی به عنوان یک محصول و راهکار تجاری مطرح بود، اما شرکت رَشنال به ترویج این فرایند به صورتی آزاد و قابل دسترس برای عموم نیز مبادرت نمود. در این راستا، مفهوم فرایند یکپارچه[۱] مطرح گردید. در کنار این مفهوم، زبان مدل‌سازی استانداردی نیز تحت عنوان زبان یکپارچه‌ی مدل‌سازی[۲] با هدف متحدالشکل کردن روش‌ها و تکنیک‌های نمادگذاری و مدل‌سازی ایجاد گردید. آقای جَکوبْسُنْ ، ایده‌ی یک فرایند آزاد و باز[۳] را با نوشتن کتابی تحت عنوانِ فرایند تولید (توسعه‌ی) یکپارچه‌ی نرم‌افزار[۴] در سال ۱۹۹۹ میلادی و با کار روی نسخه‌ی پیش‌نویسی از محصول آر.یو.پی، تحقق بخشید. از آن موقع، کتاب‌های زیادی درباره‌ی فرایند یکپارچه و با هدف ترویج راهکارهای موفق، مفاهیمِ جدید فازها، دیسیپلین‌ها، و موارد مشابه آن، به رشته‌ی تحریر درآمد.

شرکت رَشنال در سال ۱۹۹۵ ، شرکت آبجکتوری[۵] را که متعلق به آقای جَکوبْسُنْ بود، در خود ادغام نمود. با ادغام این شرکت، فرایندی تحت عنوان فرایند رَشنالْ آبجکتوری[۶] ایجاد گردید. در نسخه‌ی چهارم از این فرایند جدید، بحث‌های مدیریت نیازمندی‌ها[۷] بر اساس دستاوردها و تجارب موفقِ شرکتی به نام رِکوویزیت[۸] به آن اضافه شد. مباحث مربوط به تست نیز از شرکت اِس.کیو.اِی[۹] که در شرکت رَشنال ادغام گردید، به فرایند رشنالْ آبجکتوری افزوده شد.

---

[1] - Unified Process
[2] - Unified Modeling Language
[3] - Open
[4] - Unified Software Development Process (USDP)
[5] - Objectory AB
[6] - Rational Objectory Process (ROP)
[7] - Requirements Management
[8] - Requisite Inc.
[9] - SQA

همان‌گونه که ذکر شد، آر.یو.پی نتیجه‌ی مستقیم نسخه‌ی چهارم از فرایند رَشنالْ آبجکتوری می‌باشد. در سال ۱۹۹۷ میلادی، شرکت رَشنال، شرکت پیور آتْریا[1] را که در زمینه‌ی مباحث مرتبط با پیکربندی[2] پیشرو بود، در خود ادغام نمود و این مباحث را نیز به فرایند آر.یو.پی اضافه نمود.

سرانجام در فِوریه‌ی سال ۲۰۰۳ میلادی، یک اِدغام دیگر هم انجام شد. ولی این بار نوبتِ خودِ شرکت رَشنال بود که در یکی از تحوّل‌های بزرگ صنعت نرم‌افزار جهان، یعنی در شرکت آی.بی.اِم[3] ادغام گردد. در این زمان، شرکت رَشنال، شرکتی با حدود بیست سال قدمت و ارزشی معادل چند میلیارد دلار بود.

امروزه شرکت رَشنال به عنوان قلبِ مهندسی نرم‌افزار در شرکت آی.بی.اِم فعالیت خود را ادامه می‌دهد. محصول آر.یو.پی نیز اکنون در مالکیت شرکت آی.بی.اِم می‌باشد.

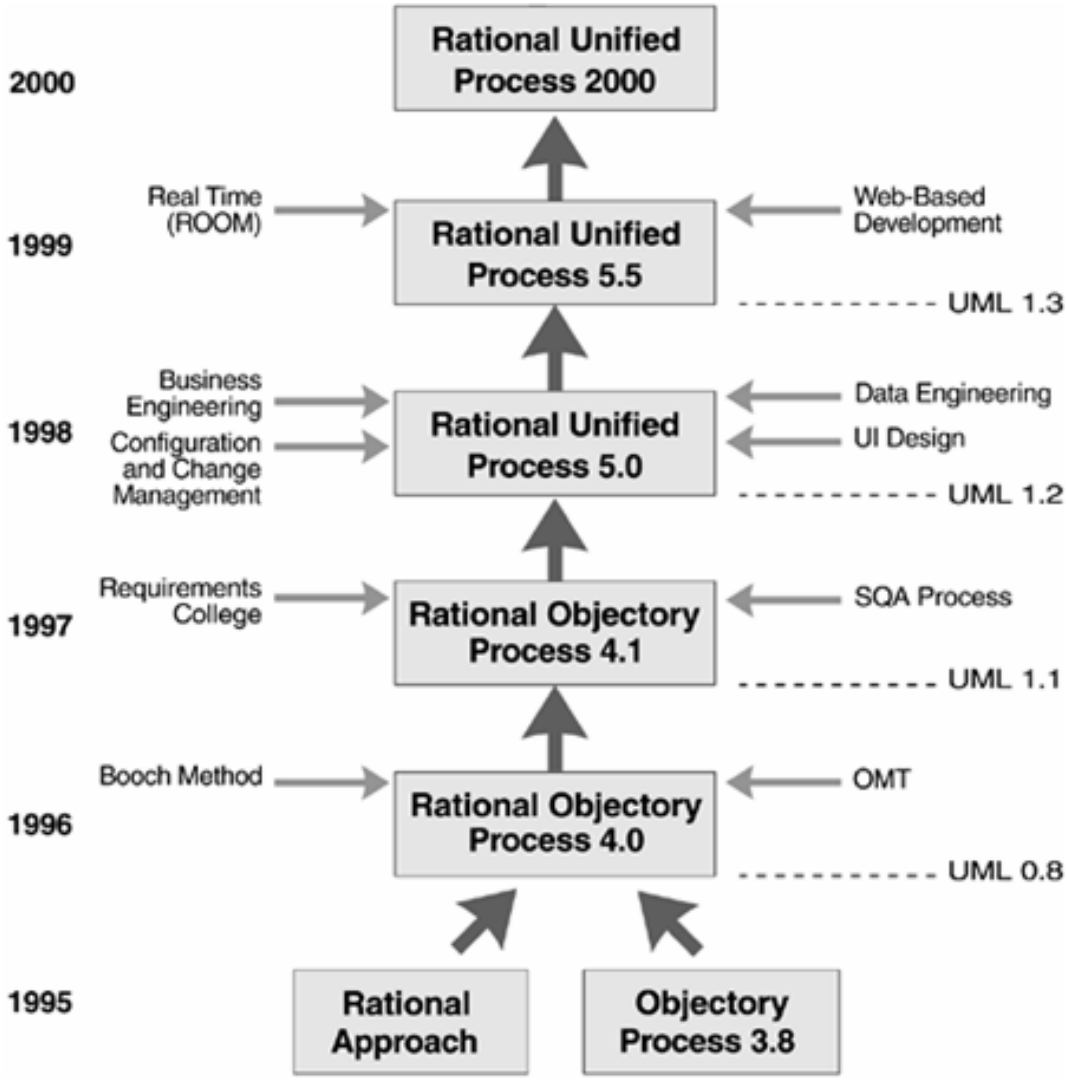
در شکل ۳–۱۹، نمایی از روند تکامل آر.یو.پی در طول سال‌های مختلف نشان داده شده است. توجه داشته باشید که آر.یو.پی، یک محصول از شرکت رَشنال (و اکنون آی.بی.اِم) می‌باشد. این محصول تقریباً دو بار در طول سال (یعنی هر شش ماه یک‌بار) به روز رسانی می‌شود.

برخلاف یو.اِم.اِل، آر.یو.پی یک استاندارد نمی‌باشد. اما اگر روزی صنعت نرم‌افزار به ضرورت وجود فرایند یا الگوی فرایندهایی استاندارد برخورد نماید، بی‌گمان فرایند یا چارچوبِ فرایندی شبیه آر.یو.پی (با داشتن ویژگی‌های منحصر به فردی مانند مدل‌سازی شده، کاملاً مستند شده، قالب تحتِ وب، قابل گسترش و به‌روز رسانی، و قابلیتِ پیکربندی) کاندید اصلی خواهد بود. امید است بتوانیم به یاری ایزد منان و همکاری نزدیک کارشناسان و متخصصان دلسوز، رویکردی مشابه را در کشور پایه‌ریزی نماییم.

[1] - Pure-Atria
[2] - Configuration
[3] - IBM

شکل ۳-۱۹

تاریخچه‌ی تکامل آر.یو.پی



چکیده‌ی فصل

در این فصل، با مفاهیم مرتبط با آر.یو.پی و برخی نکات مهم مرتبط با آن، آشنا شدیم. دیدیم که آر.یو.پی به سه مفهومِ تا حدی متفاوت اشاره دارد:

- آر.یو.پی، رویکرد و روشی است برای تولید نرم‌افزار که تکرارشونده[۱] (و با افزایش تدریجی[۲])، متمرکز بر معماری[۳] (تاکید بر تثبیت معماری)، و مبتنی بر مواردِ کاربرد[۴] (مشتری‌مدار)، مشتق از ریسک[۵] می‌باشد.
- آر.یو.پی، یک فرایند مهندسی نرم‌افزار می‌باشد که به خوبی تعریف و سازماندهی شده است. این فرایند، به ما می‌گوید که چه کسی[۶]، چه کاری[۷] را، چه موقع[۸]، و چگونه[۹] انجام دهد تا اینکه تولید نرم‌افزار به صورت تیمی انجام شده و نتیجه‌ی آن فراورده‌ای نرم‌افزاری با کیفیت مطلوب باشد.
- آر.یو.پی، فرایندی است دربرگیرنده‌ی چارچوبی برای تعریف فرایندهای تولید سیستم‌های نرم‌افزاری. بر این اساس، آر.یو.پی بسیار فراتر از یک فرایند است. آر.یو.پی دارای قابلیت پیکربندی و تناسب با طیف وسیعی از پروژه‌ها، سازمان‌ها، فناوری‌ها، و فراورده‌ها می‌باشد. سازمان‌ها و تیم‌های مختلف می‌توانند تجاربِ موفق خود را به این فراورده بیافزایند.

برخی از مهم‌ترین نکات بیان شده در این فصل عبارتند از:

- کیفیت فراورده به کیفیت فرایند وابسته است. بنابراین داشتن یک فرایند با کیفیت و تعریفِ قالب[۱۰] پروژه بر اساسِ آن، گامی است به سوی دستیابی به کیفیت برترِ فراورده.

---

[1] - Iterative
[2] - Incremental
[3] - Architecture-Centric
[4] - Use-Case Driven
[5] - Risk-Driven
[6] - Who
[7] - What
[8] - When
[9] - How
[10] - Template

- با توجه به اینکه آر.یو.پی گنجینه‌ای است از راهکارهای موفق، بنابراین بزرگ و جامع بودن، نه تنها مانع و مشکلی محسوب نمی‌شود، بلکه مزیتی کم‌نظیر برای آن می‌باشد و عملاً موجبات بکارگیری آن را در سطحی وسیع در سرتاسر دنیا فراهم نموده است.
- فراورده بودنِ آر.یو.پی، این امکان را فراهم نموده است که به راحتی در اختیار همه‌ی ذینفعان اَعم از تیم تولیدکننده و نیز مشتری و کارفرما قرار گیرد.
- آر.یو.پی بر خلاف برداشت و تصوری که در برخی شرکت‌ها و تیم‌ها وجود دارد، فرایند چابکی[1] است. به عبارت دیگر، آر.یو.پی را می‌توان آن را در طیف وسیعی از پروژه‌ها حتی پروژه‌های کوچک نیز با کمترین سربار[2] بکار برد.
- اگر قرار باشد که استانداردی برای فرایندهای تولید نرم‌افزار ارائه گردد، به احتمال زیاد، آر.یو.پی یا الگویی شبیه آن که دارای بسیاری از ویژگی‌های آر.یو.پی خواهد بود، به عنوان این استاندارد معرفی خواهد شد.
- ماهیت فازهای آر.یو.پی با ماهیت فازهای رویکرد آبشاری[3]، به طور کلی متفاوت می‌باشد.
- یک فرایندِ مطلوب که به کمک آن بتوان قالب یک پروژه‌ی موفق را تعریف نمود، باید دارای ویژگی‌هایی مانند قابلیتِ پیش‌بینی[4] در طول پروژه، اِمکان یادگیری مستمر و نیز اِمکان تکرارِ مجددِ موفقیت‌ها و پرهیز از تکرارِ مجددِ شکست‌ها و همچنین قابلیتِ مقیاس‌پذیری[5] (بکارگیری در پروژه‌هایی با اَبعاد و اندازه‌های مختلف)، باشد.

در فصل بعد، ویژگی‌های کلیدی و نیز اصول پایه‌ای آر.یو.پی را تشریح خواهیم نمود.

---

[1] - Agile
[2] - Overhead
[3] - Waterfall
[4] - Predictability
[5] - Scalability

پرسش‌هایی برای مطالعه‌ی بیشتر

۱- برای درک بهتر آر.یو.پی، آن را با فرایندهای مطرحی مانند اُپِن (OPEN) ، اِسکروم (Scrum) ، اِکس.پی (XP) ، کاتالی‌سیس (Catalysis) و اِوُ (Evo) مقایسه‌ی نمایید. مقایسه‌ی آر.یو.پی و اِکس.پی را می‌توانید در داخلِ آر.یو.پی بیابید.

۲- دستاوردها و مستنداتِ مهمِ فرایند آر.یو.پی کدامند؟

۳- چگونه می‌توان آر.یوپی را در پروژه‌های کوچک بکار گرفت؟

۴- چه ارتباطی میان آر.یو.پی و استاندارد سی.اِم.اِم (CMM) وجود دارد؟

۵- آیا سازمانی که استاندارد ایزو دارد می‌تواند آر.یو.پی را بکار گیرد؟ در صورت مثبت بودن پاسخ، چگونگی آن و نیز چالش‌های ممکن را بررسی نمایید؟

۶- آیا می‌توان از آر.یو.پی برای تعریف فرایند یک پروژه که در آن افراد تیم با متدولوژی و تفکر ساختیافته آشنایی دارند، استفاده نمود؟

۷- یک سازمان در پذیرش آر.یو.پی با چه مشکلاتی مواجه می‌گردد؟

منابع و مراجع

[1]. Ivar Jacobson, Grady Booch, James Rumbaugh, (1999). *The Unified Software Development Process*, Reading, MA: Addison-Wesley

[2]. Philippe Kruchten, (2003). *The Rational Unified Process: An Introduction*, Reading, MA: Addison-Wesley.

[3]. Per Kroll, Philippe Kruchten, (2003). *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*, Reading, MA: Addison-Wesley.

[4]. Rational Software Corporation. (2003) "Rational Unified Process 2003.06," Available at: http://www.rational.com/

[5]. Pankaj Jalote, (2002). *Software Project Management in Practice*, Reading, MA: Addison-Wesley.

[6]. Peter Eeles, Kelli Houston, Wojtek Kozaczynski, (2002). *Building J2EE™ Applications with the Rational Unified Process*, Reading, MA: Addison-Wesley.

[7]. Craig Larman, (1998). *Applying UML and Patterns: An Introduction to OOA/D and the Unified Process*, Reading, NJ: Prentice Hall PTR.

[8]. Walker Royce, (1998). *Software Project Management: A Unified Framework*. Reading, MA: Addison-Wesley.

[9]. Craig Larman, (2003). *Agile and Iterative Development: A Manager's Guide*, Reading, MA: Addison-Wesley.

[10]. OMG (2001). Object Management Group. *Software Process Engineering Metamodel (SPEM)*. OMG, doc ad/01-03-08, April 2, 2001. Available at: http://cgi.omg.org/cgi-bin/doc?ad/01-03-08.

[11]. Software Academy. (2006) "Unified Process Knowledge Base," Available at: http://www.unifiedProcess.info/

[12]. Scott W. Ambler, (2000). *The Unified Process Inception Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[13]. Scott W. Ambler, (2000). *The Unified Process Elaboration Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[14]. Scott W. Ambler, (2000). *The Unified Process Construction Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[15]. Scott W. Ambler and Larry L. Constantine, (2001). *The Unified Process Transition Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

فصل چهارم

ویژگی‌ها و روحِ آر.یو.پی

مهم‌ترین موضوعات مورد بررسی در این فصل، عبارتند از:

- ویژگی‌های کلیدی آر.یو.پی
  - توسعه مبتنی بر رویکرد تکرارشونده
  - متمرکز بر معماری
  - توسعه بر مبنای مواردِ کاربرد
- اصول هشت‌گانه‌ی فلسفه و روحِ آر.یو.پی

۴ ویژگی‌ها و روحِ آر.یو.پی

در ابتدای این فصل، سه ویژگی کلیدی آر.یو.پی را بررسی خواهیم نمود. این ویژگی‌ها، شکل‌دهنده‌ی ماهیّت آر.یو.پی می‌باشند. سپس، با اصول و فلسفه‌ی آر.یو.پی که آنها را روحِ آر.یو.پی نیز می‌نامند، آشنا خواهیم شد.

ویژگی‌های کلیدی آر.یو.پی، که آن را به عنوان یک فرآیندِ بالغ و تکامل یافته برای تولیدِ فراورده‌های نرم‌افزاری مطرح نموده است، عبارتند از:

۱- داشتنِ رویکردِ مبتنی بر توسعه‌ی تکرارشونده[۱] و تکامل تدریجی[۲]

۲- متمرکز بر معماری[۳]

۳- توسعه بر مبنای مواردِکاربرد[۴] (بر اساسِ نیازها و خواسته‌های مشتری[۵]).

برای شناخت و درک بهترِ مفاهیمِ آر.یو.پی، آشنایی با این سه ویژگی کلیدی، ضروری است. بنابراین در ادامه، با هر یک از این ویژگی‌ها آشنا خواهیم شد.

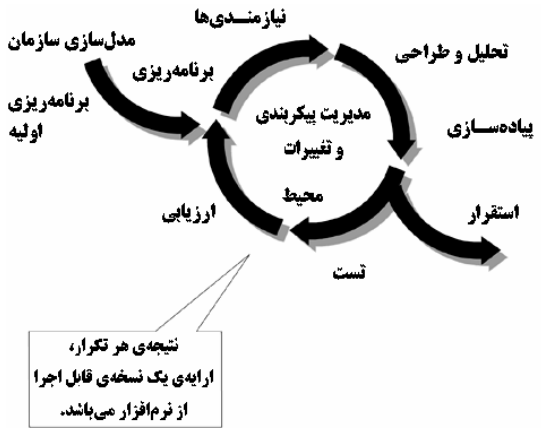
[1] - Iterative
[2] - Incremental
[3] - Architecture-Centric
[4] - Use-Case Driven
[5] - Customer-Oriented

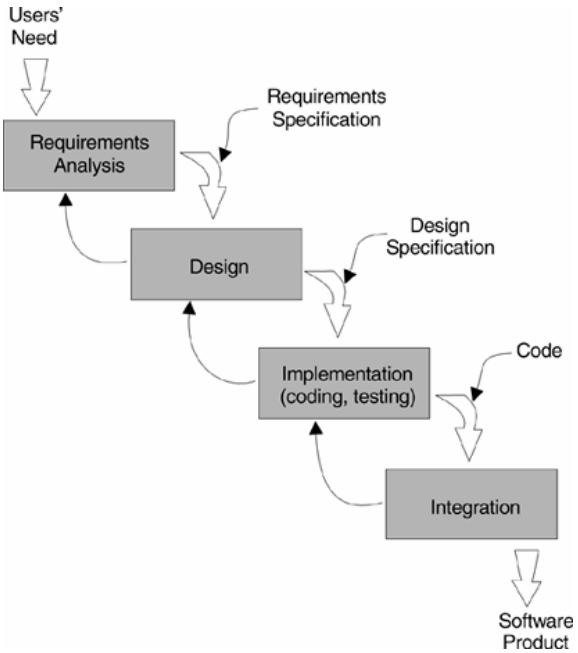
رویکرد مبتنی بر توسعه‌ی تکرارشونده[1]

یکی از ویژگی‌های برجسته‌ی فرآیند آر.یو.پی، داشتن رویکردِ مبتنی بر توسعه‌ی تکرارشونده و تکاملِ تدریجی[2] فراورده در طی آن، می‌باشد. رویکرد تکرارشونده در مقابل رویکرد آبشاری[3] قرار دارد.

شکل ۴-۱

رویکرد آبشاری (سمتِ راست) در کنار رویکردِ تکرارشونده (سمتِ چپ)





ذکر این نکته جالب است که رویکردِ تکرارشونده نسبت به رویکرد سنتی آبشاری هم مشکل‌تر است و هم آسان‌تر! مشکل بودن این رویکرد به این دلیل است که رویکردِ تکرارشونده مستلزمِ برنامه‌ریزی مستمر و کاملاً پویا[4] در طول پروژه می‌باشد. این در حالی است که در رویکردِ آبشاری برنامه‌ریزی یک بار و آن هم در ابتدای پروژه انجام می‌شود. اما دلیل آسان‌تر بودن رویکرد تکرارشونده نسبت به رویکرد آبشاری، این است که در رویکردِ تکرارشونده، فرصتِ یادگیری و بهبود در طول مسیر (چرخه‌ی تولید فراورده) فراهم می‌باشد. به این ترتیب، در طول پروژه، اِمکان تصحیح به موقعِ اشتباهات وجود خواهد داشت؛ اگر در یک تکرار[5] اشتباه کردید،

[1] - Iterative Development
[2] - Incremental
[3] - Waterfall
[4] - Dynamic
[5] - Iteration

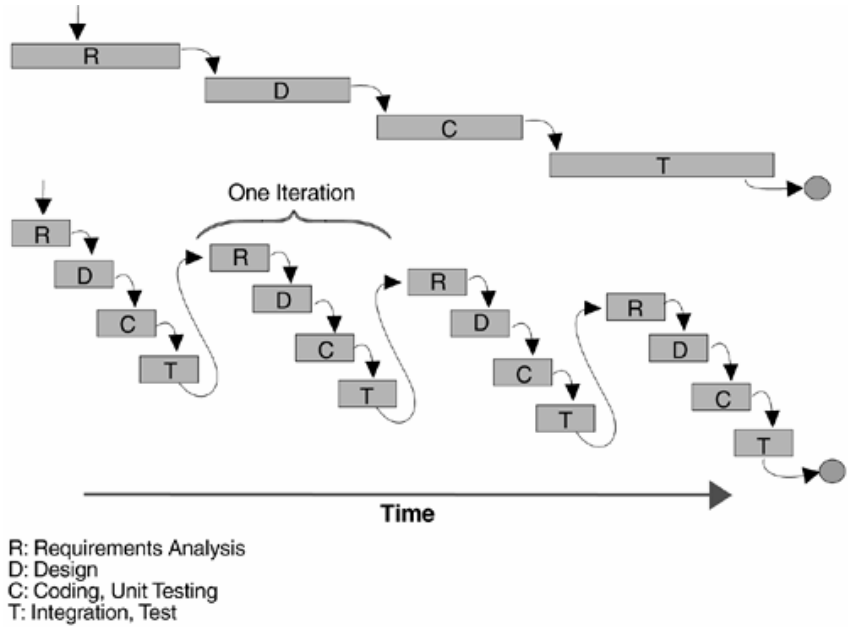
نگران نباشید، در تکرارِ بعد، می‌توانید جبران کنید. در حالی‌که در رویکردِ آبشاری، بسیاری از اشتباهات در انتهای پروژه آشکار می‌شود و بالطبع فرصت کمی برای تصحیح وجود خواهد داشت.

مزیّت‌های عمده‌ی رویکردِ تکرارشونده نسبت به رویکردِ آبشاری، عبارتند از:

- ریسک‌ها[1] را زودتر می‌توان تسکین داد.
- تغییرات، بسیار راحت‌تر قابل مدیریت و کنترل می‌باشد.
- امکان استفاده‌ی مجدد در سطوح بالاتری فراهم می‌گردد.
- تیمِ انجام دهنده‌ی پروژه، قادر خواهد بود در طول پروژه، یادگیری داشته باشد.
- کیفیت کلی محصول نیز بهتر خواهد بود.

شکل ۴-۲

شکل کلی رویکرد تکرارشونده: تکرارِ چندین باره‌ی رویکرد آبشاری، البته با ملاحظاتی خاص



[1] - Risks

مرکزیت قرار گرفتن معماری[1]

در این بخش، ضمنِ ارائه‌ی تعریفی از مفهومِ معماری و بیان اهمیت و جایگاه آن در فرایندِ تولید فراورده‌های نرم‌افزاری، یکی دیگر از ویژگی‌ها و اصول بنیادی آر.یو.پی را که تمرکز بر معماری در طول فرایند می‌باشد، بررسی خواهیم نمود.

برای مفهوم معماری نرم‌افزار، تعاریف متعددی ارائه شده است. در آر.یو.پی، معماری نرم‌افزار به صورت زیر تعریف می‌شود:

معماری نرم‌افزار دربرگیرنده‌ی مجموعه‌ای است از تصمیم‌های مهم و کلیدی در رابطه با سازماندهی یک سیستمِ نرم‌افزاری؛ از جمله، تصمیماتی پیرامونِ انتخابِ عناصرِ ساختاری[2] و واسط‌هایشان[3]، رفتارِ سیستم در قالبِ همکاری‌های[4] میانِ این اجزاء، ترکیبِ این عناصرِ ساختاری و رفتاری و تشکیل زیر سیستم‌های[5] بزرگ‌تر، و شیوه‌ی[6] معماری که راهنمای شکل‌گیری آن است. ملاحظات دیگری مانندِ کاربری[7]، کارکِرد[8]، کارایی[9]، انعطاف‌پذیری[10]، استفاده‌ی مجدد[11]، جامعیت، محدودیت‌ها و انتخاب‌های فنی و اقتصادی، و ملاحظاتِ زیبایی‌شناسی[12]، از دیگر ملاحظات و مواردِ مرتبط با معماری می‌باشند.

---

1 - Architecture-Centric
2 - Structural Elements
3 - Interfaces
4 - Collaborations
5 - Subsystems
6 - Style
7 - Usage
8 - Functionality
9 - Performance
10 - Resilience
11 - Reuse
12 - Aesthetic

با این تعریف، معماری ذینفعان[1] زیادی دارد از جمله:

- تحلیل‌گر سیستم[2]، که به کمک معماری قادر به سازماندهی نیازمندی‌ها و نیز درکِ محدودیت‌ها و ریسک‌های فناوری می‌باشد.
- کاربران نهایی یا مشتریان که توصیفی سطح بالا از آنچه روی آن سرمایه‌گذاری کرده یا در اختیارشان قرار خواهد گرفت، بدست می‌آورند.
- مدیر پروژه، برای سازماندهی تیم و نیز برنامه‌ریزی فعالیت‌ها
- طراحان که با بکارگیری معماری، اصول زیربنایی، الگوها، و نیز مکانیزم‌ها را درک کرده و محدوده‌های طراحی‌شان را شناسایی می‌نمایند.
- سازمان‌های تولیدکننده‌ی دیگر (در یک سیستم باز) برای درک چگونگی برقراری واسط با آن
- و نیز معماران و سایرِ پیمانکارانِ جزء[3]

برای اینکه همه‌ی ذینفعان قادر باشند با معماری ارتباط برقرار کرده، درباره‌ی آن بحث نمایند و آن را به کار گیرند، لازم است که معماری، برای هر یک از ذینفعان، منظرهای[4] مختلفی فراهم کرده باشد. در هر یک از این منظرها، ملاحظاتِ خاصی از معماری حذف شده تا درک آن برای ذینفعانِ علاقمند به آن منظر، ساده‌تر و دقیق‌تر باشد.

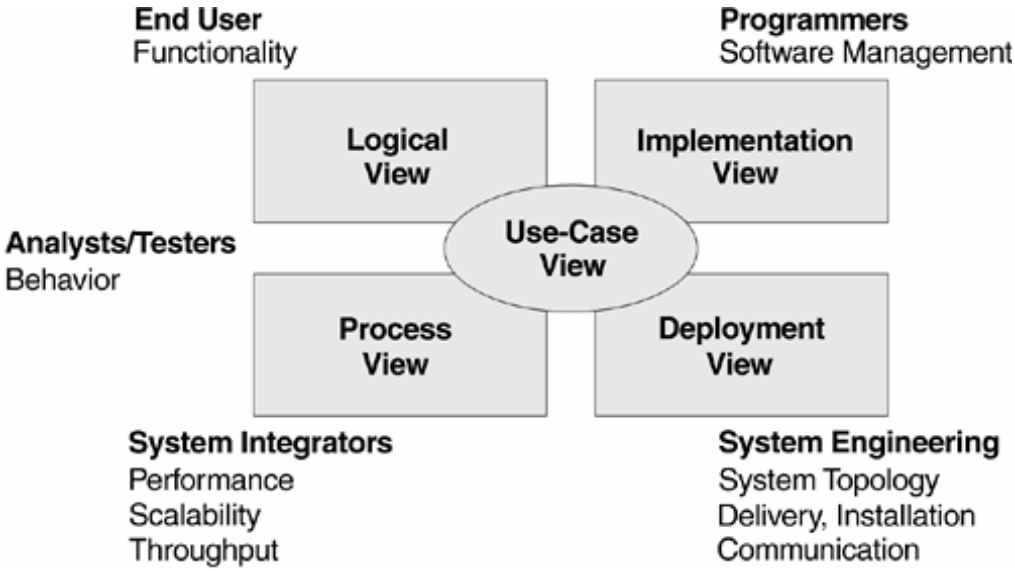
بنابراین، یک معماری کامل، چند بُعدی بوده و منظرهای مختلفی در آن فراهم می‌شود. بر همین اساس، آر.یو.پی مدلی از معماری تحتِ عنوان معماری ۱+۴ را معرفی می‌نماید. معماری که در آن حداقل ۵ منظرِ مختلف از معماری فراهم می‌گردد. شکل ۴-۳، این مدل را نشان می‌دهد. در ادامه هر یک از منظرهای اشاره شده را مختصراً بررسی می‌نماییم.

---

[1] - Stakeholder
[2] - System Analyst
[3] - Sub-contractor
[4] - View

شکل ۴- ۳

مدلِ معماری ۱+۴ در آر.یو.پی



منظرِ منطقی[1]

در این منظر از معماری، ملاحظات مربوط به نیازمندی‌های وظیفه‌مندی[2] و کارکِردی سیستم مطرح می‌گردد. به عبارت دیگر، این منظر به چیستی‌های[3] سیستم یا مجموعه‌ی بایدها و نبایدهای مورد انتظارِ کاربران نهایی مرتبط می‌باشد. این منظر، سطح تجریدی[4] (سطحی با حذف جزئیات غیر ضروری) از مدلِ طراحی می‌باشد و در آن بسته‌ها[5]، زیرسیستم‌ها[6]، و کلاس‌های اصلی طراحی، دیده می‌شود.

[1] - Logical View
[2] - Functional Requirements
[3] - What
[4] - Abstraction
[5] - Packages
[6] - Sub-systems

منظرِ پیاده‌سازی[۱]

این منظر از معماری، به منظورِ توصیف چگونگی سازماندهی مؤلفه‌ها و ماجول‌های نرم‌افزاری (کُدها، داده‌ها، مؤلفه‌ها، نسخه‌های اجرایی، و دیگر دستاوردهای[۲] مرتبط) در قالب بسته‌ها[۳]، لایه‌ها[۴]، و ملاحظات مرتبط با مدیریتِ پیکربندی[۵] ارائه می‌گردد. مسائلی مانند سهولتِ تولید، مدیریتِ دارایی‌ها[۶]، استفاده‌ی مجدد[۷]، پیمانکاران جزء[۸]، و مؤلفه‌های آماده[۹] در این منظر مورد بررسی قرار می‌گیرد.

منظرِ پردازه‌ای[۱۰]

ملاحظاتِ مرتبط با جنبه‌های همرندی[۱۱] سیستمْ در زمان اجرا، مانندِ وظیفه‌های[۱۲] سیستمی، نخ‌کشی‌ها[۱۳]، یا پردازه‌ها[۱۴]، و نیز تعاملِ میان آنها، در این منظر بررسی می‌شود. همروندی و توازی، ملاحظات مرتبط با بالا آوردن[۱۵] و پایین بردنِ سیستم[۱۶]، میزان تحمل‌پذیری نسبت به خطا[۱۷]، و توزیع‌شدگی[۱۸]، از مهم‌ترین مسائلِ مورد بررسی در این منظر می‌باشند. نقاطِ بارِ زیاد[۱۹] و بُن‌بَست‌ها[۲۰]، زمان پاسخ، ظرفیتِ ورودی و خروجی سیستم[۲۱]، و نیز مقیاس‌پذیری[۲۲]، از دیگر ملاحظات مورد بررسی در این منظر می‌باشند.

---

[1] - Implementation View
[2] - Artifact
[3] - Packages
[4] - Layers
[5] - Configuration Management
[6] - Assets
[7] - Reuse
[8] - Subcontaractors
[9] - COTS or Commercial off the Shelf Components
[10] - Process View
[11] - Concurrency
[12] - Task
[13] - Thread
[14] - Processes
[15] - Start up
[16] - Shut down
[17] - Fault Tolerance
[18] - Distribution
[19] - Load
[20] - Deadlock
[21] - Throughput
[22] - Scalability

منظرِ استقرار[۱]

در این منظر از معماری، چگونگیِ استقرارِ مؤلفه‌های زمان اجرا، مانند فایل‌های اجرایی و بانک‌های اطلاعاتی، روی گره‌های بسترِ[۲] سخت‌افزاری پیش‌بینی شده، بررسی می‌شود. در این منظر از معماری است که مهندسِ نرم‌افزار و مهندسِ سیستم[۳] با هم همکاری می‌نمایند. مسائلی مانند استقرار، نصب[۴]، و کارایی[۵] در این منظر، مورد بررسی قرار می‌گیرند.

منظرِ موردِ کاربرد[۶]

این منظر، نقش خاصی در معماری نرم‌افزار ایفا می‌نماید. این منظر، شاملِ یکسری از کلیدی‌ترین سناریوها[۷] یا مواردِکاربرد می‌باشد. این مجموعه سناریوها یا مواردِکاربرد، در فازهای آغازین[۸] (شناخت) و تشریح[۹] (معماری) برای کشف، طراحی، و ایجادِ معماری مناسب و در فازهای بعدی، برای اعتبارسنجی[۱۰] منظرهای مختلف بکار می‌رود.

منظرهای مختلف در تناظر با مدل‌های مختلفِ تولید شده در فرایند می‌باشند. در جدول ۴-۱، این ارتباط نشان داده شده است. منظرهای مختلفِ معماری سیستم، در قالبِ دستاوردِ[۱۱] سند معماری نرم‌افزار[۱۲]، مستند و توصیف می‌شوند.

---

1 - Deployment View
2 - Platform
3 - System Engineer
4 - Installation
5 - Performance
6 - Use-Case View
7 - Scenario
8 - Inception
9 - Elaboration
10 - Validation
11 - Artifact
12 - Software Architecture Document

جدول ۴-۱

منظرهای معماری در تناظرِ با مدل‌های تولید شده در فرایند

منظرِ معماری	مدل
منظرِ منطقی	مدلِ طراحی
منظرِ پردازه‌ای	مدلِ طراحی (یا مدل پردازه در سیستم‌های پیچیده)
منظرِ پیاده‌سازی	مدلِ پیاده‌سازی
منظرِ استقرار	مدلِ استقرار
منظرِ مواردِ کاربرد	مدلِ موارد کاربرد

باید توجه داشت که معماری، تنها نقشه‌ای از سیستم نیست. ملاحظات مهمِ کیفیت، از جمله، امکان‌پذیری، کارایی، انعطاف‌پذیری، و استحکام به وسیله‌ی معماری نشان داده می‌شوند. بنابراین، ارزیابی و تثبیت معماری به معنای تثبیت این ملاحظات می‌باشد. در آر.یو.پی، معماری به اندازه‌ای مهم است که هدف اصلی فازِ دومِ فرایند، یعنی فاز تشریح[۱]، به تثبیت آن اختصاص یافته است. در طی این فاز، علاوه بر توصیفِ معماری نرم‌افزار، یک پیش‌الگوی معماری[۲] پیاده‌سازی شده و تصمیم‌گیری‌های فنی در آن تست می‌شود. این پیش‌الگو، در فازِ سومِ فرایند، یعنی فازِ ساخت[۳]، تکامل یافته و سیستمِ نهایی را ایجاد می‌نماید.

بر اساس این دو دستاوردِ اصلی مرتبط با معماری، یعنی سند معماری نرم‌افزار و پیش‌الگوی معماری، سه دستاورد دیگر حاصل می‌شود که عبارتند از:

- رهنمودهای طراحی[۴]، چگونگی بهره‌گیری از الگوها[۵]
- ساختارِ فراورده در محیط تولید بر اساس منظرِ پیاده‌سازی[۶]
- ساختارِ تیم بر اساس ساختار منظرِ پیاده‌سازی

[1] - Elaboration Phase
[2] - Architectural Prototype
[3] - Construction Phase
[4] - Design Guidelines
[5] - Patterns
[6] - Implementation View

بنابراین، آر.یو.پی با تمرکز بر معماری، حل و فصلِ ملاحظاتِ مرتبط با ریسک‌های فنی را در اولویت قرار داده و پس از تثبیتِ معماری، سیستمِ نهایی را بر اساس آن کامل می‌نماید. تمرکز بر معماری، امکان استفاده از الگوها یا تجارب موفق و نیز کنترلِ منطقی و کاهشِ هزینه‌های زمانی و مالی پروژه را فراهم می‌آورد.

توسعه مبتنی بر مواردِ کاربرد[1]

هدف آر.یو.پی، تولیدِ سیستمِ مورد انتظار کاربران و برای پاسخگویی به نیازهای واقعی آنان است؛ سیستمی که در تطابق با نیازهای کاربران بوده و با هزینه‌ی زمانی و مالی مناسب و نیز با کیفیت مطلوب تولید شده باشد. آنچه مسلم است اینکه خواسته‌ها و نیازهای ذینفعان[2] را باید به درستی درک نموده و سپس راهکاری برای پاسخ‌گویی به این نیازها پیاده‌سازی نماییم. در نهایت باید اطمینان حاصل کنیم که آنچه ساخته‌ایم، همان است که ذینفعان خواسته‌اند. همانطور که ملاحظه می‌نمایید، در تمامِ طول مسیر، خواسته‌ها، انتظارات، و نیازهای ذینفعان بر فعالیت‌های مختلف تأثیرگذار می‌باشد. بنابراین در طول فرایند هیچگاه نباید از خواسته‌ها و نیازها فاصله بگیریم.

در گرفتن نیازمندی‌ها[3] دو هدف عمده را دنبال می‌نماییم: پیدا کردن خواسته‌ها و نیازهای واقعی کاربران و بیان آنها به صورت مناسبی برای همه‌ی ذینفعان از جمله مشتری، کاربران، و تیم تولید. آر.یو.پی راهکار موفقی را سرلوحه‌ی تمام فعالیت‌ها قرار داده است که بر اساس الگوی مشتری‌مداری[4] شکل گرفته است. این راهکار موفق، توسعه مبتنی بر مواردِ کاربرد[5] نام دارد.

مدل‌سازی مواردِ کاربرد، یکی از تکنیک‌های موفق در مدل‌سازی و مدیریتِ نیازمندی‌ها[6] می‌باشد. در این مدل، تمامِ افراد و چیزهایی که بیرون از سیستم واقع بوده و با آن در تعامل می‌باشند، اَکتور[7] نامیده می‌شوند. تعامل یا دیالوگِ میان اَکتورها و سیستم را موردِ کاربرد می‌نامند. مدلِ مواردِ کاربرد، منظر بیرونی[8] سیستم را

---

1 - Use-Case Driven Development
2 - Stakeholders
3 - Requirements
4 - Customer-Oriented
5 - Use-Case
6 - Requirements Management
7 - Actor
8 - External View

آنگونه که کاربران یا اَکتورها می‌بینند بیان می‌دارد. تفاوت مفاهیم موردِ کاربرد و تابع[1] این است که موردِ کاربرد، وظیفه‌مندیِ سیستم از دیدِ یک یا چند اَکتور می‌باشد؛ یک موردِ کاربرد ممکن است شامل یک یا چند تابع از سیستم باشد ولی هر تابعی یک موردِ کاربرد نیست.

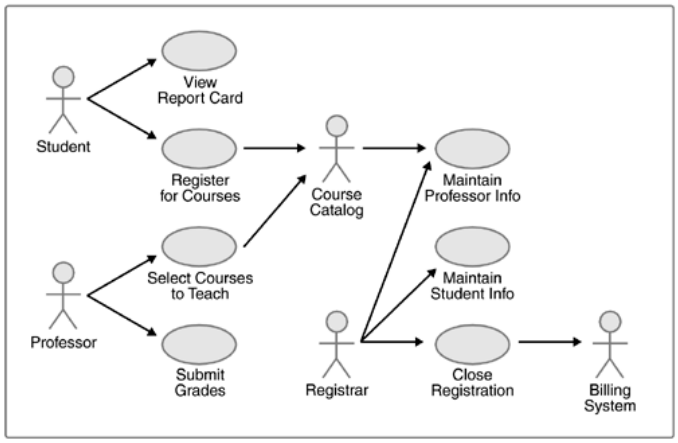
تشریحِ کاملِ مفهوم و چگونگی مدل‌سازی مواردِ کاربرد، از حوصله‌ی بحث‌های این کتاب خارج است. در اینجا، تنها به ذکر چند نکته‌ی مهم در رابطه با ملاحظات این تکنیک در آر.یو.پی، اِکتفا می‌نماییم.

به طور کلی، نیازمندی‌ها دو دسته‌اند: نیازمندی‌های وظیفه‌مندی[2] و نیازمندی‌های غیر وظیفه‌مندی[3]. در مدلِ مواردِ کاربرد، عمدتاً نیازمندی‌های وظیفه‌مندی بررسی می‌شود. سادگی و درعین حال جامعیّت این مدل، آن را به عنوان مکانیزمِ مناسبی برای مدیریت نیازمندی‌ها مطرح نموده است. دیاگرامی تحت عنوانِ دیاگرامِ مواردِ کاربرد که از دیاگرام‌های مطرح در زبانِ مدل‌سازیِ استانداردِ یو.اِم.اِل می‌باشد، یکی از اجزاء اصلی این مدل است. البته بخش عمده‌ای از این مدل، به صورت متنی و توصیفی است.

[1] - Function
[2] - Functional Requirements
[3] - Non-functional Requirements

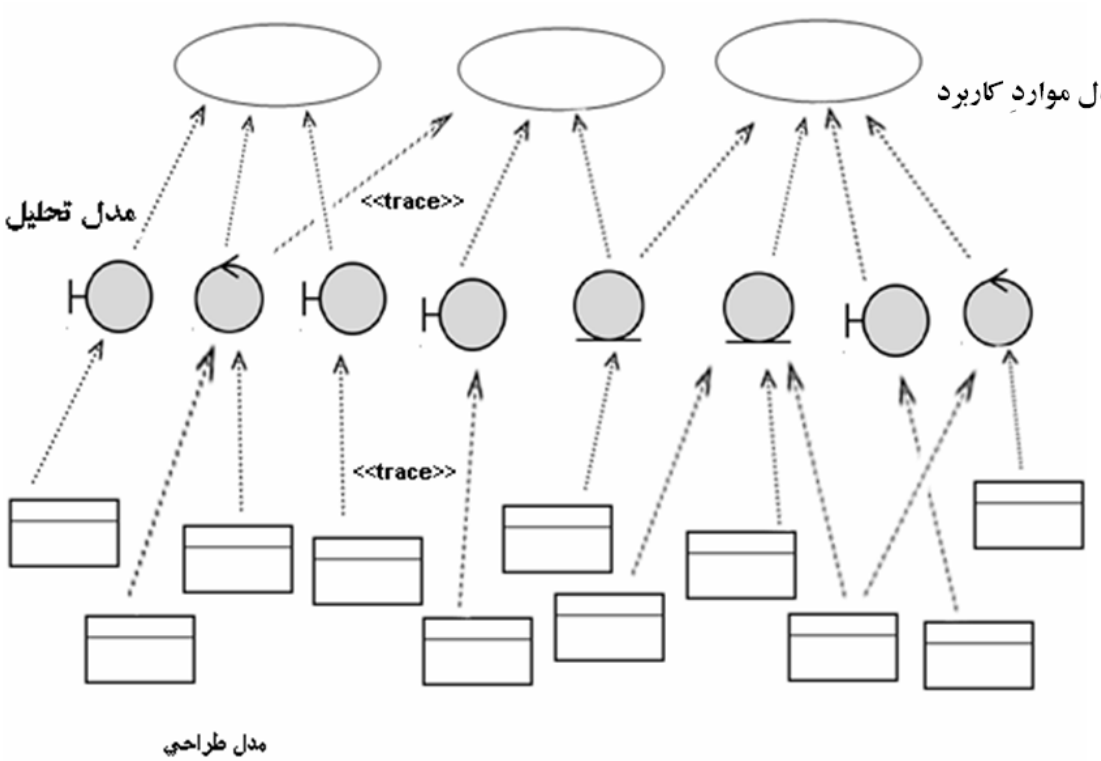
شکل ۴- ۴

نمونه‌ای از یک دیاگرامِ مواردِ کاربرد



شکل ۴- ۵

ارتباطِ میان مدلِ مواردِ کاربرد و مدل‌های تحلیل و طراحی

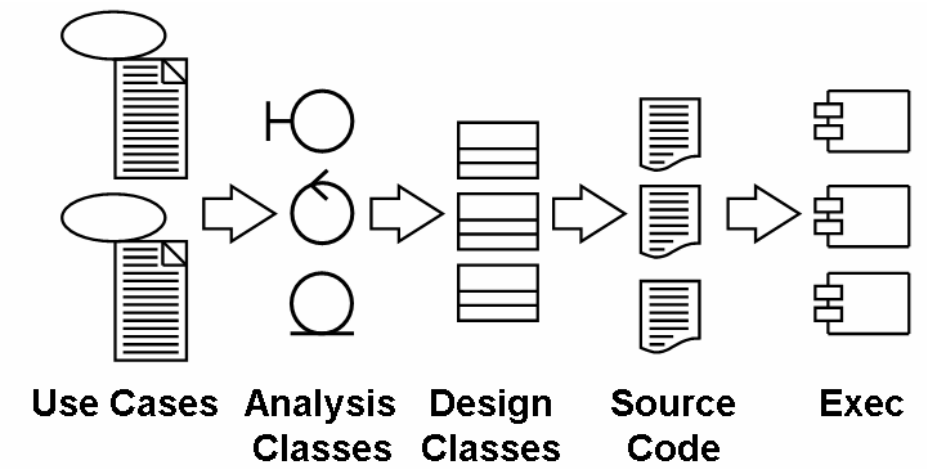


در آر.یو.پی، برنامه‌ریزی تکرارها[1] بر اساسِ مواردِ کاربرد[2] صورت می‌پذیرد. مبنا قرار گرفتنِ مواردِ کاربرد، این اطمینان را به ذینفعان[3] می‌دهد که هر کاری که در حال انجام است، اعم از تحلیل، طراحی، پیاده‌سازی، تست، یا استقرار[4]، با خواسته‌ها و انتظارات آنها در تطابق می‌باشد. با تثبیتِ معماری در انتهای فازِ دومِ فرایند، تمامِ فعالیت‌ها متوجه تشریح کامل‌تر مواردِ کاربرد، تحلیل، طراحی، پیاده‌سازی، و تست آنها خواهد بود.

مدلِ مواردِ کاربرد در آر.یو.پی، همانند یک چسب، دستاوردها و مدل‌های دیگر را به هم می‌چسباند. در واقع توسعه بر مبنای مواردِ کاربرد، جزء بنیادی از فرایند بوده و به نحوی قالبِ آن را شکل می‌دهد. مواردِ کاربرد در سرتاسرِ چرخه‌ی تولید، مبنای انجامِ بسیاری از فعالیت‌ها بوده، اطلاعات را میان مدل‌های مختلف به جریان انداخته، و موجبات حفظ سازگاری میان این مدل‌ها را فراهم می‌نماید.

شکل ۴- ۶

از مواردِ کاربرد تا مؤلفه‌های اجرایی سیستم



[1] - Iteration
[2] - Use-Case
[3] - Stakeholder
[4] - Deployment

لازم به ذکر است که همین تکنیک را می‌توان برای مدل‌سازی نیازمندی‌های یک سازمان و به طور کلی مدل‌سازی کسب و کار[۱] نیز به کار گرفت. در واقع، مدل‌سازی کسب و کار در آر.یو.پی بر اساسِ متدولوژی شیءگرا[۲] و تکنیکِ مدل‌سازی مبتنی بر مواردِ کاربرد سازمانی، بنا شده است.

برخی از مهم‌ترین کاربردهای مستقیم مواردِ کاربرد در فرایند آر.یو.پی عبارتند از:

- ایجاد و اعتبارسنجیِ[۳] مدلِ طراحی
- تعریفِ مواردِ تست[۴] و رویه‌های[۵] مرتبط با آن
- برنامه‌ریزی تکرارها[۶]
- ایجاد راهنمای کاربر[۷]
- استقرار[۸] سیستم

در ادامه‌ی این فصل، اصولِ هشت‌گانه‌ی آر.یو.پی را که روحِ آر.یو.پی[۹] نیز نامیده می‌شود، معرفی خواهیم نمود.

---

[1] - Busienss Modeling
[2] - Object Oriented Methodlogy
[3] - Validation
[4] - Test-Case
[5] - Procedures
[6] - Iteration
[7] - User Manuals
[8] - Deployment
[9] - RUP Spirit

فلسفه و روحِ آر.یو.پی

اصولی را که در این فصل تحتِ عنوان روح یا فلسفه‌ی آر.یو.پی مطرح خواهیم نمود، ریشه در چندین سال تجربه‌ی هزاران تیم در بکارگیری روش‌های موفق و خصوصاً فرایند یکپارچه‌ی آر.یو.پی در پروژه‌های بزرگ و کوچک دارد. جدول ۴-۲ ، این اصول را نشان می‌دهد.

جدول ۴-۲

اصول هشت‌گانه‌ی فلسفه و روحِ آر.یو.پی

از همان ابتدا و به طور مستمر بر ریسک‌ها (مخاطرات) اصلی و مهم پروژه غلبه نمایید، در غیر این صورت، این ریسک‌ها بر شما غلبه خواهند کرد!	*Attack major risks early and continuously, or they will attack you.*
اطمینان یابید که در طولِ فرآیند، فعالیت‌های شما همواره برای مشتری ارزش افزوده‌ای در بر دارند.	*Ensure that you deliver value to your customer.*
همواره بر داشتن یک نرم‌افزارِ قابلِ اجرا در تمام مقاطع و در طول پروژه (نه فقط در انتهای آن) تاکید داشته باشید.	*Stay focused on executable software.*
از همان ابتدای پروژه، در اندیشه‌ی راهکار مناسبی برای مدیریت تغییرات باشید و هرگز این کار را به بعد موکول ننمایید.	*Accommodate change early in the project.*
رسیدن به یک چارچوب (معماری) مستحکم و قابل اجرا و مبنا قرار دادن آن را در اولویت قرار دهید.	*Baseline and executable architecture early on.*
سیستم را با استفاده از مؤلفه‌ها بنا نمایید.	*Build your system with components.*
همه‌ی افراد مشارکت کننده در جریان تولید یک سیستم باید در قالب یک و تنها یک تیم فعالیت کنند.	*Work together as one team.*
کیفیت را در بطنِ همه‌ی فعالیت‌های خود قرار دهید. کیفیت چیزی نیست که بتوان آن‌را در انتهای کارها، پس از پیاده‌سازی سیستم و مثلاً با انجام تست، بدست آورد!	*Make quality a way of life, not an afterthought.*

هر چند که اصول بیان شده در جدول ۴-۲ را روح و فلسفه‌ی آر.یو.پی نامیدیم، ولی همواره به یاد داشته باشید که این اصول، بیانگر واقعیت‌ها و قواعد امروزی صنعتِ تولید فراورده‌های سیستمی، خصوصاً صنعتِ نرم‌افزار می‌باشند، به‌گونه‌ای که بکارگیری این اصول برای موفقیت در تولید این دسته از فراورده‌ها، بسیار ضروری و اجتناب‌ناپذیر است؛ خواه فرایند تولید مطابقِ آر.یو.پی باشد و یا اینکه از فرایندها و الگوهای دیگری استفاده شود. بنابراین، نهادینه کردن این اصول را در اولویت برنامه‌های خود برای بهبود فرایند تولید قرار دهید.

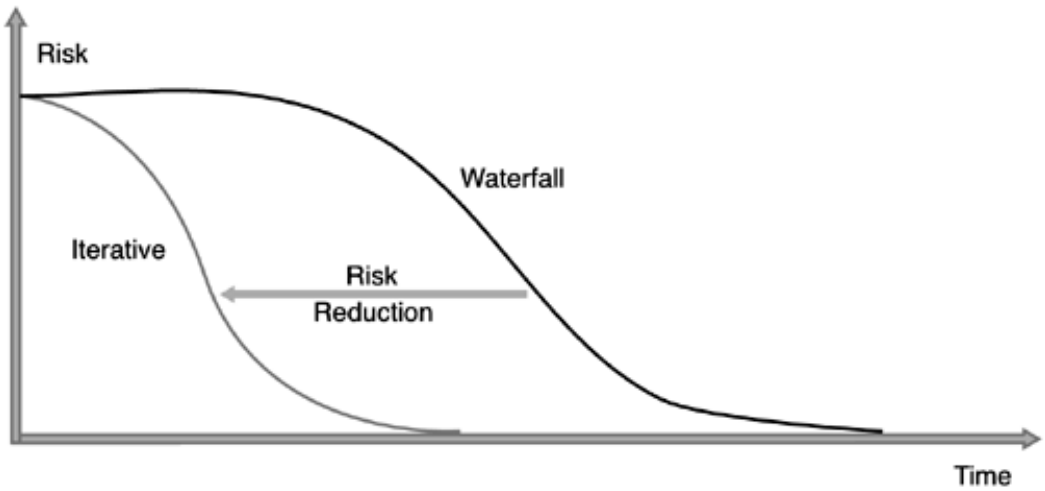
البته تجربه‌ی هزاران تیم در بیش از ده‌هزار شرکت و بیش از نیم میلیون استفاده‌کننده‌ی رسمیِ آر.یو.پی، نشان داده که آر.یو.پی می‌تواند شما را در راه رسیدن به این اصول و تحقق آن‌ها در تیم و سازمان، یاری نماید. همین موضوع، آر.یو.پی را از سایر رویکردها و فرایندهای دیگر، متمایز می‌نماید. در این میان، اصولِ مربوط به مدیریت ریسک‌ها و ایجاد معماری اثبات شده و قابل اجرا از ویژگی‌های بارز آر.یو.پی می‌باشد. در ادامه‌ی این فصل، اصول ذکر شده را مختصراً تشریح خواهیم نمود.

اصلِ اول. غلبه‌ی مستمر و هرچه‌سریع‌تر بر ریسک‌های عمده‌ی پروژه

یکی از مزایای اولیه‌ی رویکردِ تکرارشونده، این است که در این رویکرد، ریسک‌ها نسبت به رویکرد آبشاری زودتر خودشان را نشان داده و بنابراین می‌توانیم ریسک‌های عمده را در اوایل پروژه شناسایی کرده و با آنها مقابله کنیم. این موضوع در نمودار شکل ۴-۷ ، نشان داده شده است.

شکل ۴-۷

پروفایل ریسک‌ها در دو رویکردِ آبشاری و تکرارشونده



چرا باید بررسی ریسک‌های عمده زودتر انجام شود؟ واضح است که یک ریسک یا مخاطره‌ی دیده‌نشده و یا رفع نشده به معنای سرمایه‌گذاری روی یک معماری معیوب و یا نیازمندی‌های ناقص است. این موضوع از لحاظ اقتصادی به هیچ‌وجه قابل قبول نیست. علاوه بر این، میزان ریسک‌ها ارتباط مستقیمی با تفاوت میان

تخمین‌های بالا و پایین و نیز چگونگی تکمیل پروژه دارد. بنابراین برای داشتن تخمین‌هایی دقیق‌تر باید ریسک‌های با اولویتِ بالاتر، زودتر بررسی شوند.

اصلِ دوم. اطمینان از ارائه‌ی ارزش[۱] برای مشتری در طولِ فرایند

طبیعتاً، ارائه‌ی چیزهایی که برای مشتری ارزش محسوب می‌شوند، هدف بسیار ارزشمندی است؛ اما سؤال این است که این کار چگونه انجام می‌شود؟ رویکردِ تکرار شونده[۲] با فراهم کردن اِمکان دریافتِ بازخوردهای[۳] مستمر از مشتری در کنار دیدِ مبتنی بر مواردِکاربرد[۴]، زمینه‌ی مناسبی برای ارائه‌ی مستمرِ ارزش به مشتری فراهم می‌نماید.

تکنیکِ مدل‌سازی نیازمندی‌ها[۵] به وسیله‌ی مواردِ کاربرد که به نوعی تحقّقِ دیدِ مشتری محوری[۶] است، از روش‌های موفق در اخذ و مدیریت نیازمندی‌های عملکردی می‌باشد. با کمک این تکنیک، مدلی از سیستم و مسأله ارائه می‌گردد که برای همه‌ی ذینفعان[۷] قابل درک بوده و همانند یک چسب، سایر مدل‌های مورد نیاز، یعنی مدل‌های تحلیل، طراحی، پیاده‌سازی، تست، و استقرار[۸] را به هم متصل می‌نماید. در آر.یو.پی، برنامه‌ریزی تکرارهای مختلف، شکل‌دهی به معماری، برنامه‌ریزی تست‌ها، تشخیص مواردِ تست[۹]، و راهنمای کاربر، همه بر اساس مدلِ مواردِ کاربرد ایجاد می‌شوند.

مدلِ مواردِ کاربرد به زبان مشتری و از دیدِ او، قابلیت‌ها و وظایفِ سیستم را بیان می‌نمایند. و از آنجایی که مواردِ کاربرد در تمامی مراحلِ فرایند، سرمشق و مبنای فعالیت‌های مختلف و تولید دستاوردهای مهم می‌باشد، بنابراین همه‌ی افراد تیم قادر خواهند بود از نزدیک با خواسته‌های مشتری (کاربران سیستم) کار

---

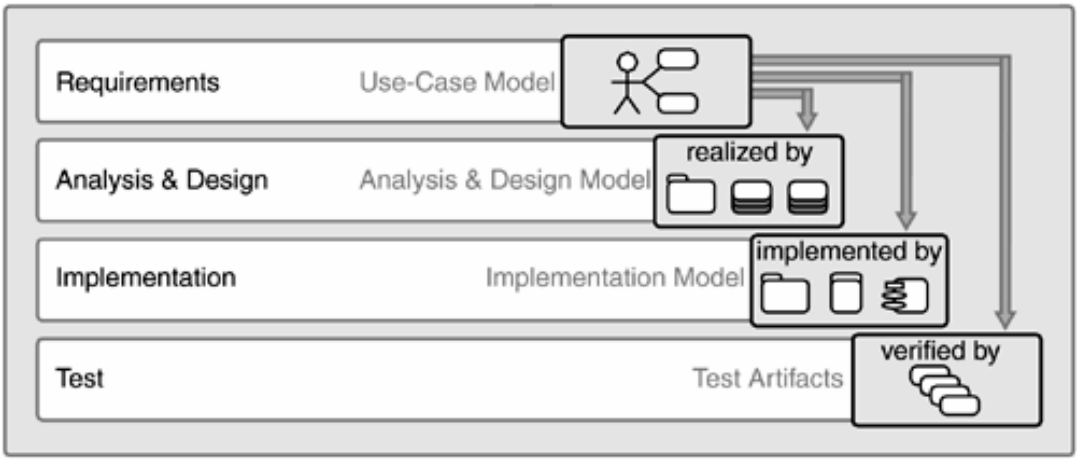
[1] - Value
[2] - Iterative
[3] - Feedback
[4] - Use-Case
[5] - Requirements
[6] - Customer-Oriented
[7] - Stakeholders
[8] - Deployment
[9] - Test Cases

کرده و آنچه بدست می‌اورند در تطابق کامل با خواسته‌ها و نیازهای همه‌ی ذینفعان[1] باشد. کنترل دقیق‌ترِ محدوده‌ی[2] سیستم، از دیگر مزایای تکنیک مدل‌سازی مواردِ کاربرد می‌باشد.

نتیجه‌ی مستقیمِ توسعه بر مبنای مواردِ کاربرد[3] به همراهِ فرایندِ مبتنی بر رویکردِ تکرارشونده[4]، اطمینان از تطابقِ کامل دستاوردها با خواسته‌ها و نیازهای ذینفعان و ارائه‌ی ارزش مورد انتظار به ایشان می‌باشد.

شکل ۴-۸

مدلِ مواردِ کاربرد، مبنای تولید سایر مدل‌ها در فرایند



[1] - Stakeholder
[2] - Scope
[3] - Use-Case Driven
[4] - Iterative Development

اصلِ سوم. تأکید مستمر بر داشتنِ یک نرم‌افزارِ قابل اِجرا[1]

به نظر شما مهم‌ترین معیارِ سنجش[2] پیشرفتِ یک پروژه چیست؟ آیا درصد کارهای انجام شده می‌تواند معیار[3] خوبی باشد؟ حجمِ مستندات تحویل شده یا درصد زمان سپری شده چطور؟ واقعیت این است که هیچ چیز به اندازه‌ی یک نرم‌افزارِ قابل اجرا[4] نمی‌تواند بیانگر پیشرفت یا عدم پیشرفت یک پروژه باشد. در واقع، مهم‌ترین معیار در سنجش پیشرفتِ کار، نه مستندات و کارهای انجام شده، بلکه نتیجه‌ی نهایی آنها، یعنی نرم‌افزار قابل اجرا است.

دانستن اینکه مثلاً ۱۰ موردِ کاربرد از مجموعِ ۲۰ موردِ کاربردِ سیستم، تشریح شده‌اند، لزوماً به معنای تکمیل ۵۰ درصد نیازمندی‌ها نیست. در اینجا لازم است به یک اصل و قائده‌ی تجربه‌شده به نام اصلِ ۲۰-۸۰ اشاره داشته باشیم. مطابق این اصل، که اصلِ پَرِتُ[5] نیز نامیده می‌شود، حتی درصورتی‌که ۸۰ درصد نیازمندی‌ها را تشریح کرده باشیم، این اِمکان وجود دارد که ۸۰ درصدِ قابلیت‌های کلیدی سیستم، در ۲۰ درصدِ باقی‌مانده قرار گرفته باشد. بنابراین نمی‌توان برای سنجش پیشرفتِ کار، به مستندات و دستاوردهای مشابه آن اِستناد نمود.

بهترین روش برای سنجش پیشرفتِ یک پروژه‌ی نرم‌افزاری، سنجش ویژگی‌ها و کارکردهای یک نرم‌افزارِ قابل اِجرا است. با این وصف، هر گاه یکی از اعضای تیم اظهار داشت که « من ۹۰ درصد کار را انجام داده‌ام » ، به او بگویید که « بسیار عالی، ایا ممکن است لطفاً نشان دهید که چه چیزی می‌تواند اجرا شود و کار کند؟ »

نرم‌افزارِ قابل اجرا را می‌توان تست کرد. میزان خطاها و نرخِ بهبود آنها، معیارهای مهمی در سنجش پیشرفتِ پروژه می‌باشند که تنها از روی یک نرم‌افزار قابلِ اِجرا بدست می‌آیند. توجه داشته باشید که اطلاعات ثبت شده در اسناد نیز بسیار مهم هستند، ولی به تنهایی کافی نمی‌باشند.

---

[1] - Executable Software
[2] - Measurement
[3] - Metrics
[4] - Executable Software
[5] - Pareto

تمرکز بر قابلِ اِجرا شدن نرم‌افزار، نوعی فرهنگِ کاری و تفکّر صحیح را در تیم ترویج می‌دهد و مانع نظریه‌پردازی‌ها و تحلیل‌های بیش از حد[۱] می‌شود.

تمرکز بر تولید یک نرم‌افزار با قابلیت اجرایی[۲]، اغلب سریع‌ترین و مناسب‌ترین راه برای تسکین[۳] ریسک‌ها نیز می‌باشد. اثبات اینکه راهکار الف از راهکارِ ب مناسب‌تر است، تنها با تولید یک نرم‌افزار قابل اِجرا منطقی است. ممکن است خیلی چیزها روی کاغذ صحیح به نظر برسد ولی در عمل و هنگام اِجرا، مشکلات آشکار شوند.

نکته‌ی مهمّ دیگری که همواره باید بدان توجه داشته باشیم، اینست که غیر از نرم‌افزارِ قابل اِجرا، بقیه‌ی دستاوردهایی[۴] که در طولِ فرایند تولید می‌شوند، به نوعی دستاوردهای پشتیبان و جانبی محسوب می‌شوند. بنابراین با تمرکز بر نرم‌افزار قابل اِجرا، تصمیم‌گیری درباره‌ی تولید یا عدمِ تولید سایر دستاوردها آسان‌تر خواهد شد. اصلِ کلی، کاهشِ سربار و پرهیز از رسمی‌شدن[۵] بیش از اندازه است. با این حال، با بزرگ‌شدنِ اندازه‌ی پروژه و افزایش پیچیدگی آن، وجود برخی از دستاوردهای پشتیبان، ضروری خواهد بود.

بنا به یک اصلِ کلّی، هرگاه نسبت به تولید یا عدمِ تولید یک دستاورد شک داشتید، آن را تولید نکنید. البتّه حذف فعالیت‌های ضروری و کلیدی پروژه، مانند تعیین چشم‌انداز، مستندسازی نیازمندی‌ها، برنامه‌ریزی فعالیت‌های تست، و انجامِ یک طراحی خوب، جایز نیست.

یکی از اشتباهات رایج کاربران آر.یو.پی، اینست که دستاوردهای مختلف را فقط به این دلیل که در آر.یو.پی، چگونگی تولیدشدن‌شان تشریح شده و قالب‌هایی[۶] برای آنها ارائه شده‌است، تولید می‌نمایند. توجه داشته باشید که آر.یو.پی یک فرایند نیست و آنچه در آن آمده، مجموعه‌ای است که باید بتوان آن را برای تعریفِ فرایند یک پروژه‌ی بزرگ چندین ساله با چندین هزار نفر و نیز برای تعریف فرایند یک پروژه‌ی کوچک استفاده نمود. بنابراین، لازم است که آر.یو.پی مجموعه‌ای غنی از دستاوردها، فعالیت‌ها، نقش‌ها، و

---

[1] - Analysis-Paralysis Antipattern
[2] - Executable Software
[3] - Mitigation
[4] - Artifacts
[5] - Formal
[6] - Template

راهنمایی‌ها را فراهم نموده باشد و هر تیم یا سازمان با توجه به شرایط و اِلزامات خاص خود، از میان این مجموعه، ضروری‌ترین دستاوردها، فعالیت‌ها، و نقش‌ها را انتخاب نماید.

اصلِ چهارم. تبیینِ راهکار مناسبی برای مدیریتِ تغییرات از ابتدای پروژه

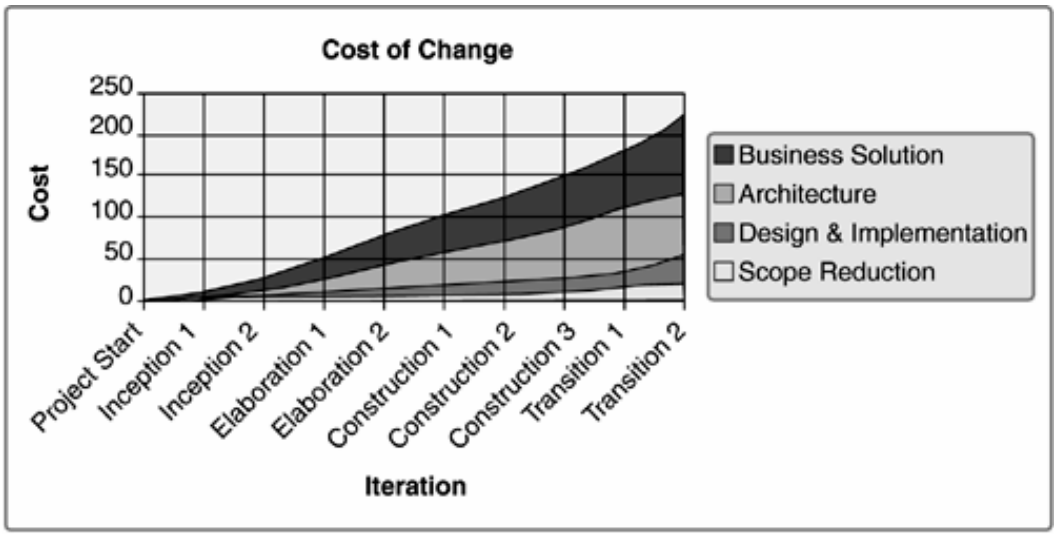
وجودِ تغییر[1]، از ثابت‌ترین پارامترهای دنیای اطراف ماست. تغییر هم خوب است و هم بد! تغییر وقتی خوب است که در کنترل ما باشد و گرنه مساوی است با فاجعه! بدون وجود تغییر، نرم‌افزارهای مختلف، نیازهای مختلف، و کاربردهای مختلف به وجود نمی‌آید. بنابراین باید به تغییر خوش‌آمد بگوییم و البته آن را کنترل کنیم.

باید توجه داشت که هر تغییری مطلوب نیست؛ مسلماً تغییر می‌تواند اثرات نامطلوبی نیز در پی داشته باشد. مثلاً، تغییرِ نیازمندی‌های[2] کلیدی در زمانی‌که آماده‌ی تحویل فراورده هستیم، به هیچ وجه مطلوب نیست. ثابت ماندن تغییرات، مانع از تکمیل پروژه می‌شود. برخی از تغییرات باعث دوباره‌کاری زیاد، افزایش هزینه، کاهش کیفیت، و به احتمال زیاد، تأخیر در تحویل فراورده خواهند شد. بنابراین، باید راهکار مناسبی برای مدیریت تغییر داشته باشیم و این راهکار را باید از همان ابتدای پروژه برنامه‌ریزی و اجرا نماییم. هزینه‌ی تغییر و اِعمال اثر آن در طول زمان به صورت نمایی افزایش می‌یابد. شکل ۴-۹، بیانگر این موضوع است.

[1] - Change
[2] - Requirements

شکل ۴-۹

افزایش هزینه‌ی اِعمال اثرِ تغییرات در طول زمان



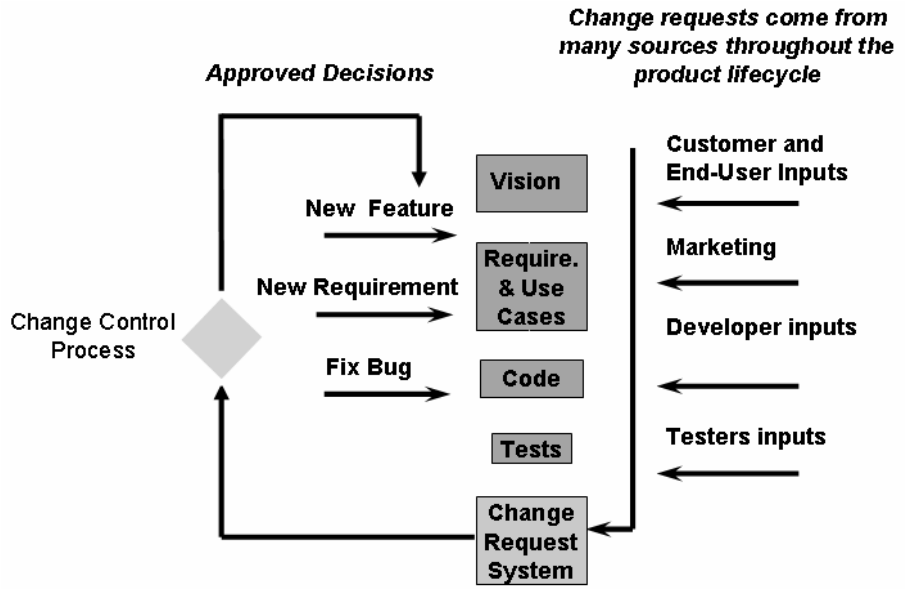
آر.یو.پی مفهوم مدیریتِ یکپارچه‌ی تغییرات[1] را معرفی می‌نماید. با کمک این مکانیزم، نه تنها مدیریت تغییرات اِمکان‌پذیر می‌گردد بلکه به صورتی واحد و از یک کانالِ مشخص، این کار انجام می‌شود.

فازهای آر.یو.پی طوری برنامه‌ریزی شده‌اند که هزینه‌ی کلی تغییرات را کاهش داده و در عین حال، اِمکان دادنِ اجازه برای رخ دادن تغییر را افزایش می‌دهند. به همین علت است که آر.یو.پی، تیم تولید را به توافق روی چشم‌انداز[2] در انتهای فاز آغازین[3] (شناخت)، یک معماری تثبیت شده[4] در انتهای فاز تشریح[5] (معماری)، و بسته‌شدنِ قابلیت‌ها و ویژگی‌های سیستم در انتهای فاز ساخت[6]، ملزم می‌نماید.

[1] - Unified Change Management or UCM
[2] - Vision
[3] - Inception
[4] - Baseline-Architecture
[5] - Elaboration
[6] - Construction

شکل ۴- ۱۰

مدلی از مفهوم و مکانیزمِ مدیریت یکپارچه‌ی تغییرات



نکته‌ی پایانی اینکه، استفاده از ابزارهای مناسب برای مدیریت تغییرات و کاهش هزینه‌ها بسیار مفید و در پروژه‌های متوسط و بزرگ، بسیار ضروری می‌باشد.

اصلِ پنجم. اولویت‌دهی به مبنا قرار دادن[1] یک معماری قابل اجرا[2] و اثبات آن

بسیاری از ریسک‌های پروژه مرتبط با معماری می‌باشند؛ خصوصاً در مواردی که با پروژه‌ای برای تولیدِ اولین نسل[3] از یک سیستمِ کاربردی روبرو هستیم. به همین دلیل، باید دقت زیادی در انتخاب معماری صرف شود. آنچه امروز به عنوان یک اصل مهم پذیرفته شده، اینست که توانایی در تثبیت هر چه زودترِ یک معماری دارای کارایی مطلوب، یعنی طراحی، پیاده‌سازی، و تست آن برای موفقیت یک پروژه بسیار ضروری و حیاتی است.

معماری با فراهم نمودن یک اسکلت از سیستم نهایی، به طور معمول دربرگیرنده‌ی ۱۰ تا ۲۰ درصد از کدهای نهایی سیستم می‌باشد. شناسایی مکانیزم‌های معماری و نیز الگوهای طراحی که راهکارهایی برای

[1] - Baseline
[2] - Executable Architecture
[3] - First Generation

یکسری مسائل و معضلات متداول و مشترک میان مؤلفه‌های مختلف سیستم می‌باشد، یکی از مهم‌ترین ملاحظات مرتبط با معماری است.

داشتن یک معماری که در آن مؤلفه‌های[1] ساختاری شناسایی شده و به خوبی در جایگاه خود قرار گرفته‌اند، اِمکان درک بهتر سیستم را فراهم می‌نماید. با کمک رویکردِ تکرار شونده در آر.یو.پی و این نکته که یک فاز به ملاحظات معماری اختصاص یافته، تیمِ تولید تجربه‌ی خوبی از تحلیل، طراحی، و پیاده‌سازی کسب نموده و بنابراین، حجم کارها و زمان لازم برای تکمیل ساخت و ساز سیستم دقیق‌تر مشخص می‌گردد. وقتی یک معماری تثبیت می‌گردد، بسیاری از ریسک‌های عمده‌ی پروژه، حل شده‌اند. اِمکان هماهنگی تیم‌های مختلف نیز با داشتن چنین معماری فراهم شده و تولید سیستم آسان‌تر می‌گردد.

در صورت توجه مناسب به معماری، تلاش برای تثبیتِ آن، و تمرکز بر قابل اِجرا نمودن معماری، اِمکان استفاده‌ی مجدد از مؤلفه‌ها نیز فراهم می‌گردد. بنابراین، آر.یو.پی برای بهره‌گیری از مزیت‌های تمرکز بر معماری، تثبیتِ آن را در اولویت قرار می‌دهد؛ تا معماری ثابت نشده، به مرحله‌ی تولید وارد نخواهیم شد.

از آنجایی که به طور متوسط، رسیدن به معماری تثبیت شده، مستلزم صرفِ حدودِ یک سوم از زمان کل پروژه می‌باشد، با تثبیت معماری می‌توان درباره‌ی بهینه کردنِ دو سومِ باقی‌مانده تصمیم‌گیری‌های مناسبی اتخاذ نمود. چه بسا تولید سیستم به لحاظ جمیع شرایط و ملاحظاتِ فنی اِمکان‌پذیر نباشد و چه بهتر که این موضوع با حداکثر یک سوم هزینه‌ی زمانی و مالی، اثبات شود.

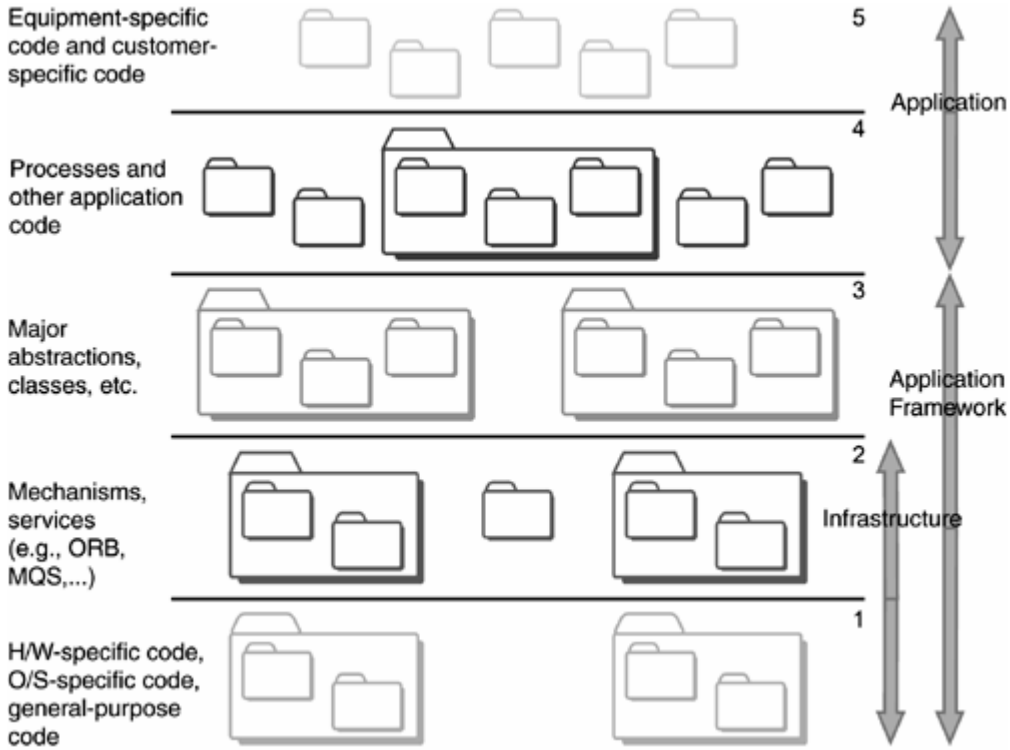
رسیدن به یک معماری مناسب و تثبیت آن، اغلب بسیار مشکل می‌باشد؛ لذا باید بهترین و ماهرترین افراد را به آن معطوف نمود. برای رسیدن به یک معماری مستحکم و تثبیت شده، باید معماری شکلِ قابلِ اِجرا[2] به خود بگیرد؛ به عبارت دیگر باید لایه‌های زیرساختی معماری، پیاده‌سازی شده و مورد آزمایش قرار گیرند. با داشتن چنین معماری می‌توان تیم را بزرگ‌تر نمود و به مرحله‌ی تولید، یعنی فازهای ساخت و انتقال، وارد شد.

---

[1] - Components
[2] - Executable

شکل ۴- ۱۱

لایه‌های زیرین معماری (لایه‌های زیرساختی) در معماری قابل اجرا، پیاده‌سازی می‌شوند.



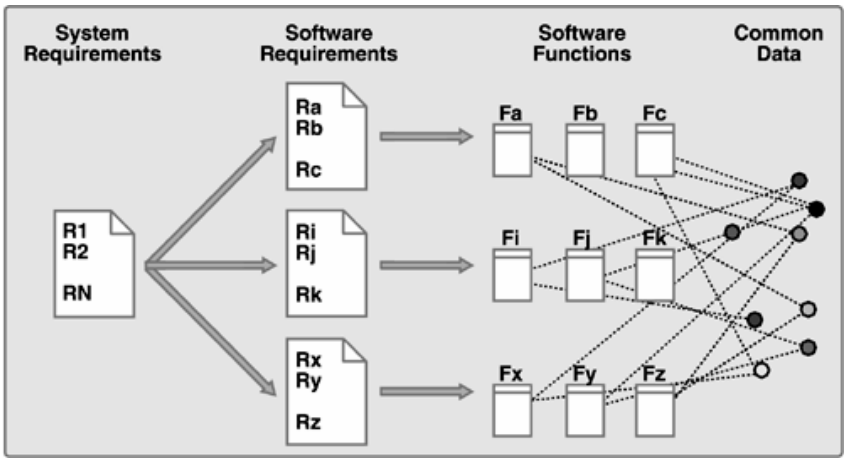
اصلِ ششم. ایجادِ سیستم با استفاده از مؤلفه‌ها[1]

در روش و متدولوژیِ ساخت‌یافته[2] (خصوصاً روش تجزیه‌ی تابعی[3]) داده‌ها[4] از توابع جدا می‌شوند. یکی از تبعات منفی این امر، بالا بودن هزینه‌ی نگهداری و تصحیح سیستم می‌باشد. برای مثال، تغییر در نحوه‌ی ذخیره‌سازی یک داده، ممکن است تعداد نامشخصی از توابع را تحت تأثیر قرار دهد و معمولاً دانستن اینکه چه تابعی در کلِّ سیستم باید تغییر نماید، بسیار مشکل است. همین موضوع، مهم‌ترین دلیل رخ دادنِ مشکل سالِ ۲۰۰۰ بود.

[1] - Components
[2] - Structured Methodology
[3] - Functional Decomposition
[4] - Data

شکل ۴- ۱۲

راهکارِ روش ساخت‌یافته در تجزیه، تحلیل، و ساختِ سیستم



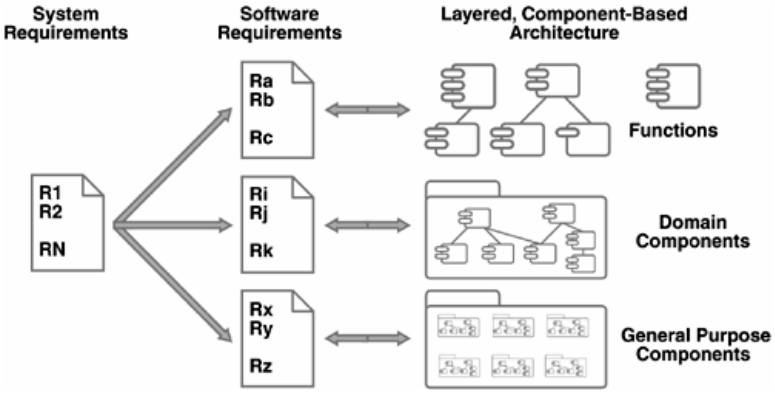
در مقابل، در روشِ توسعه مبتنی بر مؤلفه[1]، اِمکان پنهان‌سازی[2] داده‌ها و توابع مرتبط با آنها در قالبِ مفهومِ مؤلفه فراهم می‌گردد. در صورتی‌که تغییری در یک داده یا در تابع مربوط به آن اِعمال شود، این تغییر از دیدِ سایرِ مؤلفه‌ها پنهان می‌ماند و بدین ترتیب، سیستم نسبت به تغییر اِنعطاف‌پذیر می‌شود. مؤلفه‌ها را هم می‌توان با تکنیک‌های شیءگرا پیاده‌سازی نمود و هم با تکنیک‌های ساخت‌یافته. هر چند که امروزه رویکرد و متدولوژیِ شیءگرا به عنوان یک رویکرد موفق‌تر و مهندسی‌تر نسبت به رویکردِ ساخت‌یافته مطرح می‌باشد.

یک مؤلفه[3]، همانند یک جعبه‌ی سیاه با یک یا چند واسط[4] یا قراردادِ مشخص می‌باشد. بنابراین برای بهره‌گیری از قابلیت‌های فراهم شده به وسیله‌ی یک مؤلفه لازم نیست از جزئیات داخلی آن آگاه باشیم، بلکه دانستن مشخصاتِ واسط یا واسط‌های فراهم شده، کافی می‌باشد. بدین ترتیب، اِمکان استفاده‌ی مجدد از مؤلفه‌ها تسهیل می‌گردد؛ یک مؤلفه را می‌توان با یک گونه‌ی دیگر جایگزین نمود و مادامی که واسط آن تغییری ننماید، هیچ یک از مؤلفه‌های دیگرِ سیستم، متوجه این تغییر نخواهند شد.

[1] - Component
[2] - Encapsulation
[3] - Component
[4] - Interface

شکل ۴- ۱۳

طراحی و تولیدِ سیستم به کمک مؤلفه‌های نرم‌افزاری



توجه داشته باشید که امروزه، توسعه مبتنی بر مؤلفه، نه یک انتخاب، بلکه یک ضرورت اجتناب ناپذیر است. بسیاری از کارشناسان معتقدند که معماری‌هایی که اکنون باید برای رسیدن به آنها تلاش شود، معماری‌های مبتنی بر سرویس[1] می‌باشد. برای تحقق این دسته از معماری‌ها، توسعه‌ی مبتنی بر مؤلفه، یک مفهوم کاملاً بنیادی و زیرساختی است.

اصلِ هفتم. فعالیت در قالبِ یک و تنها یک تیم

مشخصاً مهم‌ترین دارایی و ثروت یک سازمان، نیروی انسانی و سرمایه‌های مبتنی بر دانش در سازمان می‌باشد. امروزه، تولید نرم‌افزار به مانند یک مسابقه‌ی ورزشی تیمی است. البته، دقت داشته باشید که صحبت از انجام دادن یا انجام ندادن کار تیمی نیست؛ در واقع، دیگر انتخاب مدلی غیر از کار تیمی و یا شکل‌های تکامل یافته‌ی آن، نمی‌تواند کارساز باشد. تجربه‌ی پروژه‌های موفق نشان داده است که نه تنها باید کار را به صورت تیمی انجام داد، بلکه باید به صورت یک و تنها یک تیم فعالیت نماییم.

بسیاری از سازمان‌ها دارای ساختاری وظیفه‌گرا[2] می‌باشند. در چنین سازمان‌هایی، بخش تحلیل، بخش طراحی، بخش تست، بخش تولید، و مانند آن را شاهد هستیم. در واقع ساختار سازمان بر حسب وظایف مورد نیاز در انجام مأموریت‌های سازمان، شکل گرفته است. در این صورت، حتی ممکن است گروه‌ها یا بخش‌های

[1] - Service Oriented Architecture or SOA
[2] - Functional Structure

مختلف، در محل‌های مختلفی مستقر باشند. متأسفانه، بسیاری از سازمان‌های تولیدکننده‌ی نرم‌افزار (یا سیستم) از چنین الگویی تبعیت می‌نمایند.

مزیت چنین سازمان‌هایی، تخصص شدن و ایجاد بخش‌های تخصصی خاص می‌باشد. اما از آنجایی که تولیدِ بسیاری از فراورده‌های جدید مستلزمِ یک فعالیت میان رشته‌ای[۱] و چندتخصصی[۲] است، این ساختار سازمانی دیگر پاسخ‌گو نیست. یکی دیگر از مشکلات عمده در چنین سازمان‌هایی، عدمِ وجود ارتباطات مؤثر میان گروه‌های مختلف می‌باشد. در این سازمان‌ها، عملاً با دوباره‌کاری‌های زیاد، عدمِ درک مناسب و در بسیاری از موارد، تأخیر در دستیابی به اهدافِ از پیش تعیین شده مواجه می‌شویم. یکی دیگر از معضلات این قبیل سازمان‌ها اینست که افراد از دیدِ مدیریت به صورت منابع قابل جایگزین دیده می‌شوند؛ به گونه‌ای که هیچ یک از افراد خود را جزء کاملی از یک پروژه نمی‌داند و بنابراین همواره می‌توان افراد را تعویض نمود.

در پروژهایی که فرایندشان آبشاری[۳] و مدت زمان اجرای طولانی مدّت دارند، که البته عملاً در تولید نرم‌افزار و نیز سایر فراورده‌های سیستمی معنا ندارد، ممکن است وجود سازمان‌های وظیفه‌گرا قابل قبول باشد، امّا در پروژه‌هایی که رویکرد تکرارشونده[۴] را پی می‌گیرند و در آنها داشتن ارتباطات زیاد میان اعضا ضروری است، و به طور کلی ، هر جا که کار تیمی لازم باشد، ساختار وظیفه‌گرا[۵] کارایی مطلوبی ندارد. بنابراین، سازماندهی باید حولِ تیم‌های چند وظیفه‌ای[۶] انجام شود. در این نوع ساختار، همه‌ی انواع تخصص‌های درگیر در کار تولیدِ فراورده، مشارکت دارند.

مهم‌ترین مزیت این نوع سازماندهی، به غیر از برقراری ارتباطات مؤثر، این است که همه‌ی اعضای تیم، خود را در موفقیتِ کلِّ سیستم مسئول می‌دانند. در این صورت افراد باید به جای اینکه بگویند که مثلاً « من این بخش مربوط به خودم را درست انجام خواهم داد » ، بگویند که « من هر آنچه را که برای ارائه‌ی یک فراورده‌ی با کیفیت مطلوب لازم باشد، انجام خواهم داد ».

---

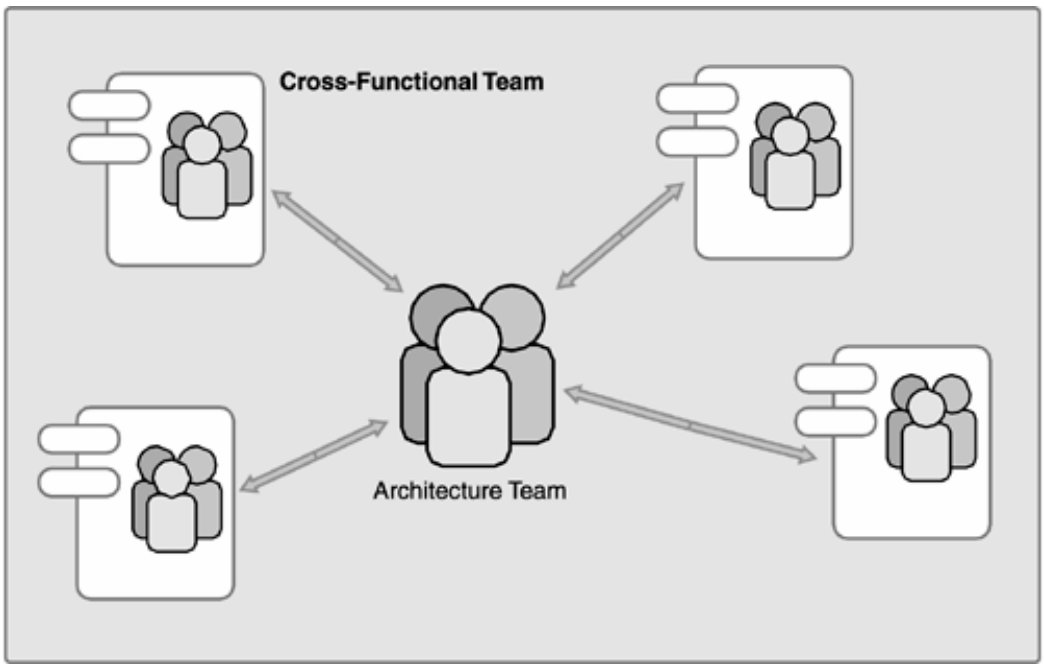
[1] - Inter-diciplinary
[2] - Multi-disciplinary
[3] - Waterfall
[4] - Iterative
[5] - Functional
[6] - Cross-Functional

باز هم بر این نکته تأکید می‌گردد که تنها یک تیم برای هر پروژه باید به کار گرفته شود. سؤالی که ممکن است اینجا به ذهن خطور کند اینست که اگر تعداد افرادِ درگیر در پروژه زیاد باشد، چگونه می‌توان کماکان یک تیم را مسئولِ کل پروژه دانست؟ در این حالت، یک راهکارِ مناسب، سازماندهی تیم‌هایی کوچک حول معماری و با کمک معماری می‌باشد. بدین ترتیب که یک تیم مسئول معماری باشد و بقیه‌ی تیم‌ها هر کدام مسئول یکی از زیر سیستم‌ها (مسئولِ انجامِ کلِ وظایف لازم برای تولید یک زیرسیستم) باشند. به غیر از تیمِ معماری، تمامِ تیم‌های دیگر، چندوظیفه‌ای می‌باشند. در این حالت، تیم معماری مسئولِ تعیین زیرسیستم‌ها و واسطهایشان بوده و هر یک از تیم‌های چندوظیفه‌ای، مسئولیتِ یک یا چند زیر سیستم را بر عهده خواهند داشت.

آر.یو.پی با مرکزیت قرار دادنِ معماری و نیز تعریف نقش‌ها، وظایف مختلف، و دستاوردهای مطلوب، سازماندهی بهتر کار تیمی را ممکن می‌سازد.

شکل ۴- ۱۴

سازماندهی تیم‌های یک پروژه‌ی بزرگ حولِ تیمِ معماری



شایان ذکر است که داشتن یک رویکردِ تکرارشونده، بر چگونگی سازماندهی تیم، چگونگی برقراری ارتباطات میان اعضا، ابزارهای مورد نیاز، و نیز نوع تخصص‌های هر یک از اعضای تیم، تأثیر زیادی دارد.

اصلِ هشتم. درنظر گرفتن کیفیت در بطنِ همه‌ی فعالیت‌ها

یکی از مزایای عمده‌ی رویکرد تکرارشونده اینست که بر خلاف رویکرد آبشاری، تست و آزمونِ نرم‌افزار، از همان ابتدای پروژه امکان‌پذیر است. حتی در اولین فاز، یعنی فازِ آغازین[۱] (شناخت)، تستِ پیش‌الگوی[۲] ایجاد شده، تأثیر زیادی در بدست آوردن بازخوردِ[۳] مناسب نسبت به مواردِ کاربرد[۴] (قابلیت‌ها و کارکردها) دارد. در فاز دوم، یعنی فاز تشریح[۵] (معماری)، یک نسخه‌ی اجرایی از سیستم تولید می‌شود که در آن معماری سیستم پیاده‌سازی شده است. بدین ترتیب، معماری و کلیه‌ی جوانب آن، از جمله کارایی[۶]، امنیت[۷]، بارگذاری[۸]، و توزیع‌شدگی[۹] تست می‌شود. دریافت بازخوردهای مناسب در این مرحله (که با طی آن، تقریباً یکْ سومِ مسیر پروژه طی خواهد شد) امکان بهبود و صرفه‌جویی در زمان و هزینه را در ادامه‌ی مسیر ممکن می‌سازد.

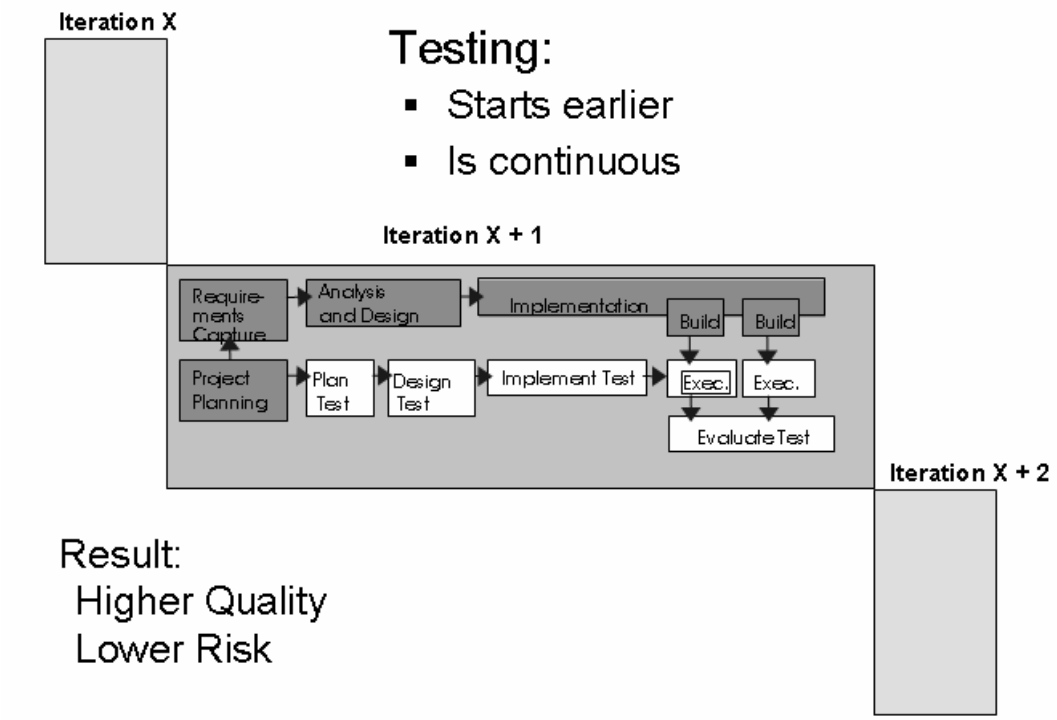
یکی دیگر از نکات مهم در بهبود مستمر کیفیت، عبارتست از مفهومِ کیفیتِ بوسیله‌ی طراحی[۱۰]. این مفهوم به معنای نزدیکی بیش از پیشِ طراحی و تست می‌باشد. در حینِ طراحی سیستم، توجه به اینکه سیستم چگونه باید تست شود، امکان بهبودِ اتوماسیونِ تست و بهره‌گیری مناسب‌تر از ابزارها فراهم می‌گردد. زیرا کدها و رویه‌های تست را می‌توان مستقیماً از روی مدلِ طراحی بدست آورد. این موضوع، نه تنها اِمکان انجام تست‌های بیشتر را فراهم می‌آورد، بلکه صرفه‌جویی قابل ملاحظه‌ای در زمان را نیز در پِی دارد. در نتیجه، با افزایش تعداد تست‌ها، تعدادِ خطاها و نقایص کم‌تر شده و کیفیت افزایش می‌یابد.

---

[1] - Inception
[2] - Prototype
[3] - Feedback
[4] - Use-Case
[5] - Elaboration
[6] - Performance
[7] - Security
[8] - Load
[9] - Distribution
[10] - Quality by Design

شکل ۴-۱۵

شروع زودتر تست و انجام مستمرِ آن، منجر به افزایش سطح کیفی و کاهش ریسک‌ها می‌شود.



به طور کلی، در آر.یو.پی، پیاده‌سازی کارکردها و قابلیت‌های سیستم باید با تست آنها همراه باشد. از آنجایی که مهم‌ترین و کلیدی‌ترین قابلیت‌های سیستم، زودتر و در فازهای اولِ پروژه پیاده‌سازی می‌شوند، با رسیدن به انتهای پروژه، سیستمی در اختیار خواهیم داشت که مدتها در حال کار بوده و قابلیت‌های کلیدی آن، جواب خود را پس داده‌اند. بسیاری از پروژه‌هایی که آر.یو.پی را برای تعریف فرایند خود بکارگرفته‌اند، تصریح نموده‌اند که افزایشِ کیفیتِ فراورده، اولین نتیجه‌ی ملموس ناشی از بهبود فرایند[1] و بکارگیری اصولِ آر.یو.پی بوده است.

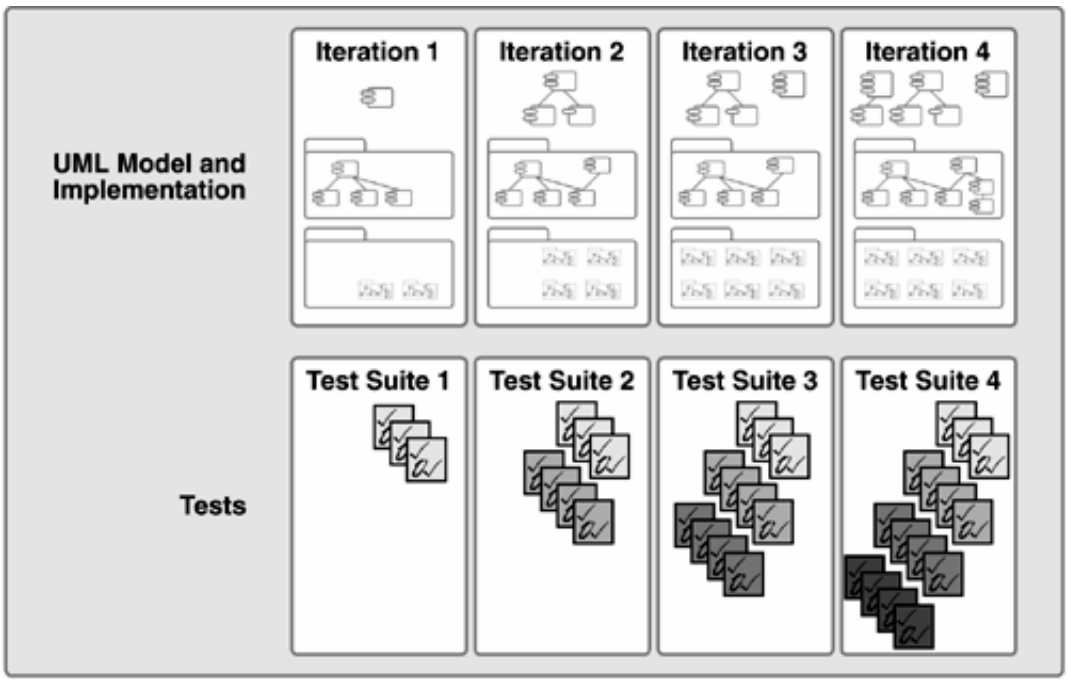
رویکرد تکرارشونده[2] نه تنها اِمکان تست را از همان اوایلِ پروژه فراهم می‌نماید، بلکه تیم تولید را به انجام تست‌های مکرّر و مستمر وادار می‌کند. این فرصت بسیار مهم و ارزنده‌ای است زیرا با تداوم تست

[1] - Process Improvement
[2] - Iterative Development

سیستمِ موجود در تکرارهای[1] مختلف (که به آن تستِ رگرسیون[2] می‌گویند) می‌توانیم همواره اطمینان داشته باشیم که درصد عمده‌ای از خطاهای ممکن را دیده‌ایم.

شکل ۴- ۱۷

تستِ رگرسیون: انجام تست هر بار به طور کامل

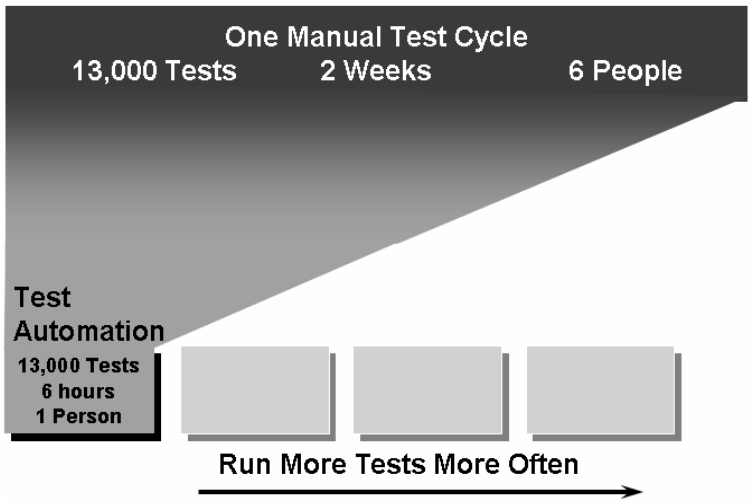


مسلماً استفاده از ابزارهای مناسب برای تست و بکارگیری مناسبِ این ابزارها، نقش بسیار مهمی در ارتقای کیفیت فراورده‌ی نهایی دارد. استفاده نکردن از این ابزارها، نه تنها موجب بالا رفتن هزینه‌های تست می‌شود، بلکه انجام برخی از موارد تست (بخش عمده‌ای از تست‌های غیرِ وظیفه‌مندی[3]) عملاً امکان‌پذیر نخواهد بود.

[1] - Iteration
[2] - Regression Test
[3] - Non-functional Test

شکل ۴- ۱۶

با بهره‌گیری از ابزارها و خودکارسازیِ تست، امکان انجام تست‌های بیشتر فراهم می‌گردد.



همواره به یاد داشته باشید که موضوعِ کیفیت[۱]، دغدغه‌ی و نگرانی همه‌ی اعضای تیم می‌باشد و در همه‌ی قسمت‌های فرایند باید ملاحظات مرتبط با آن وجود داشته باشد؛ بازبینی دستاوردها[۲] به محضِ تولیدِ آنها، فکر کردن درباره‌ی چگونگیِ تستِ نیازمندی‌ها هنگامِ شناسایی آنها، طراحی برای قابل تست بودن[۳]، از جمله ملاحظات ذکر شده می‌باشد.

البته، توجه داشته باشید که موضوع کیفیت تنها به تستِ سیستم برنمی‌گردد. بلکه، همه‌ی نقش‌ها[۴] در تمام دیسیپلین‌ها، مسئول کیفیت فراورده‌ی نهایی می‌باشند. آر.یو.پی با فراهم نمودن چک لیست‌ها[۵] و راهنمایی‌ها[۶]، و نیز قراردادن گام‌هایی[۷] برای بازبینی در بسیار از فعالیت‌ها، نقش‌های مختلف را در ارزیابی به موقع فعالیت‌ها و دستاوردها، یاری می‌دهد.

[1] - Quality
[2] - Artifact
[3] - Design for Testability
[4] - Role
[5] - Checklist
[6] - Guideline
[7] - Step

چکیده‌ی فصل

در این فصل، پس از ذکرِ ویژگی‌های کلیدی آر.یو.پی، اصول و مبانی آن را معرفی نمودیم. مهم‌ترین مباحثِ مطرح شده عبارتند از:

- بررسی مختصرِ هر یک از ویژگی‌های کلیدی آر.یو.پی:
  - توسعه مبتنی بر رویکردِ تکرارشونده و تکاملِ تدریجی
  - متمرکز بودنِ فرایند بر معماری
  - توسعه بر مبنای مواردِ کاربرد (دیدِ مشتری‌مداری)
- بررسی اصولِ هشت‌گانه‌ی روحِ آر.یو.پی
  - از همان ابتدا و به طور مستمر، بر ریسک‌ها (مخاطرات) اصلی و مهمِ پروژه غلبه نمایید، در غیر این صورت، این ریسک‌ها بر شما غلبه خواهند کرد!
  - اطمینان یابید که در طول فرایند، فعالیت‌های شما همواره برای مشتری ارزش افزوده‌ای ایجاد می‌نماید.
  - همواره بر داشتن یک نرم‌افزار قابل اجرا در تمام مقاطع و در طول پروژه (نه فقط در انتهای آن) تاکید داشته باشید.
  - از همان ابتدای پروژه، در اندیشه‌ی راهکار مناسبی برای مدیریت تغییرات باشید و هرگز این کار را به بعد موکول ننمایید.
  - رسیدن به یک چارچوب (معماری) مستحکم و قابل اجرا و مبنا قرار دادن آن را در اولویت قرار دهید.
  - سیستم را با استفاده از مؤلفه‌ها بنا نمایید.
  - در قالب یک و تنها یک تیم فعالیت کنید.
  - کیفیت را در بطنِ همه‌ی فعالیت‌های خود قرار دهید. کیفیت چیزی نیست که بتوان آن‌را در انتهای کارها، پس از پیاده‌سازی سیستم و مثلاً با انجام تست، بدست آورد!

پرسش‌هایی برای مطالعه‌ی بیشتر

۱- درباره‌ی چگونگی معرفی و فرهنگ‌سازی رویکردِ تکرارشونده در یک سازمان، تحقیق نمایید.

۲- با بررسی ویژگی‌ها و انگیزه‌های دیدِ مشتری‌مداری، آن را با مدلِ مواردِ کاربرد[1] مقایسه نمایید.

۳- درباره‌ی نقش معماری نرم‌افزار در سازماندهی تیم‌های تولید تحقیق نمایید.

۴- تمرکز فرایند بر معماری، چگونه می‌تواند بر جلوگیری از دوباره‌کاری تأثیر داشته باشد؟

۵- درباره‌ی انواعِ مکانیزم‌های معماری تحقیق نمایید.

۶- یکی از تکنیک‌های مدیریت نیازمندی‌ها در صنعت، تکنیکِ QFD می‌باشد. این تکنیک را با مدل‌سازی مواردِ کاربرد (Use-Csse Modeling) مقایسه نمایید.

۷- در رابطه با رابطه‌ی میان اصول و روحِ آر.یو.پی با راهکارهای موفق، تحقیق نمایید.

[1] - Use-Case Model

منابع و مراجع

[1]. Ivar Jacobson, Grady Booch, James Rumbaugh, (1999). *The Unified Software Development Process*, Reading, MA: Addison-Wesley

[2]. Philippe Kruchten, (2003). *The Rational Unified Process: An Introduction*, Reading, MA: Addison-Wesley.

[3]. Per Kroll, Philippe Kruchten, (2003). *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*, Reading, MA: Addison-Wesley.

[4]. Rational Software Corporation. (2003) "Rational Unified Process 2003.06," Available at: http://www.rational.com/

[5]. Craig Larman, (1998). *Applying UML and Patterns: An Introduction to OOA/D and the Unified Process*, Reading, NJ: Prentice Hall PTR.

[6]. Walker Royce, (1998). *Software Project Management: A Unified Framework*. Reading, MA: Addison-Wesley.

[7]. Software Academy. (2006) "Unified Process Knowledge Base," Available at: http://www.unifiedProcess.info/

[8]. Scott W. Ambler, (2000). *The Unified Process Inception Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[9]. Scott W. Ambler, (2000). *The Unified Process Elaboration Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[10]. Scott W. Ambler, (2000). *The Unified Process Construction Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[11]. Scott W. Ambler and Larry L. Constantine, (2001). *The Unified Process Transition Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

بخش سوم

فصل پنجم: چرخه‌ی تولیدِ فراورده

فصل ششم: فازِ آغازین (شناخت)

فصل هفتم: فازِ تشریح (معماری)

فصل هشتم: فازِ ساخت

فصل نهم: فازِ اِنتقال

فصل پنجم

چرخه‌ی تولیدِ فراورده

مهم‌ترین موضوعات مورد بررسی در این فصل، عبارتند از:

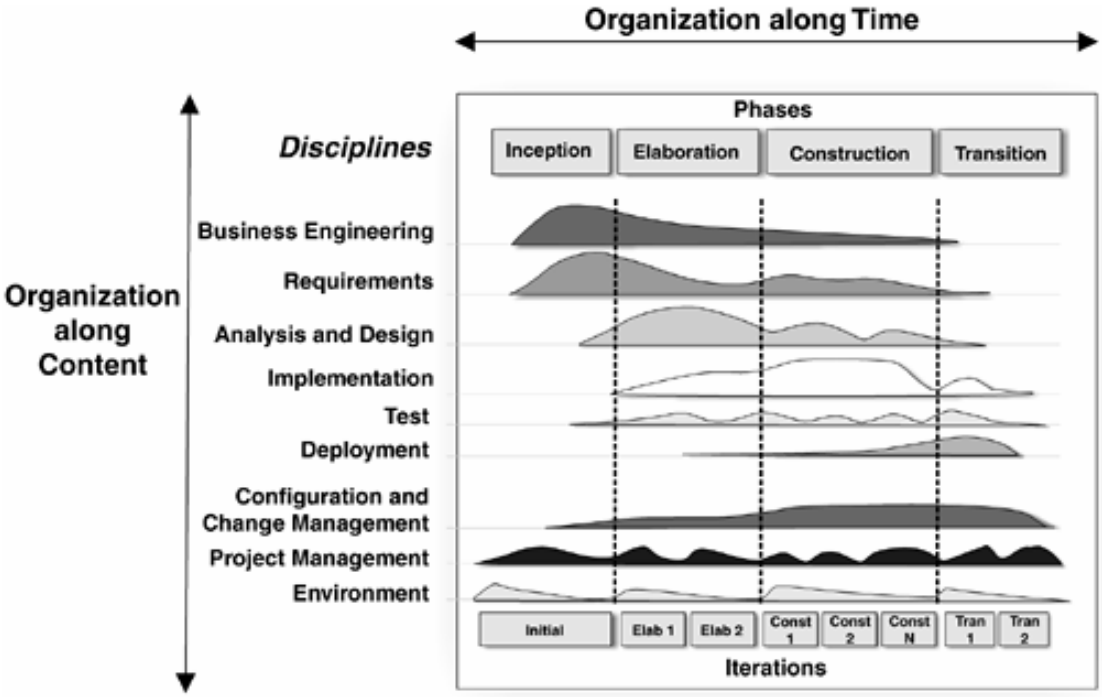
- معرفی ساختارِ متعامدِ آر.یو.پی
- بررسی نکاتی پیرامونِ ساختارِ پویا یا دینامیکِ آر.یو.پی

۵ چرخه‌ی تولیدِ فراورده

در این فصل، مشخصاتِ کلّی چرخه‌ی تولید فراورده را در فرایند آر.یو.پی مورد بررسی قرار می‌دهیم. همان‌گونه که پیش از این نیز اشاره گردید، آر.یو.پی دارای دو بُعد پویا و ایستا می‌باشد. چرخه‌ی تولیدِ در آر.یو.پی، متناظر با بُعدِ پویا و دینامیک آن است. در شکل ۵-۱، ساختار آر.یو.پی از دو منظر پویا و ایستا نشان داده شده است. این ساختار را ساختار متعامد[1] آر.یو.پی نیز می‌نامند.

شکل ۵-۱

ساختارِ متعامدِ آر.یو.پی: دو بُعدِ پویا و ایستا



[1] - Orthogonal

در این فصل و چهار فصلِ آینده، جنبه‌ی پویایی آر.یو.پی را که شامل ملاحظات زمانی مرتبط با پروژه از جمله فازها و تکرارها می‌باشد، بررسی خواهیم نمود. در ادامه و در بخش سومِ این کتاب، ساختار محتوایی، یعنی جنبه‌ی ایستای آر.یو.پی که شامل مؤلفه‌هایی مانند دیسیپلین‌ها، نقش‌ها، فعالیت‌ها، و دستاوردها می‌باشد، بررسی می‌گردد.

آر.یو.پی چرخه‌ی تولیدِ محصول را به چهار بازه‌ی زمانی به نامِ فاز[۱] تقسیم می‌نماید: فازِ آغازین[۲] (شناخت)، فازِ تشریح[۳] (معماری)، فازِ ساخت[۴]، و فازِ انتقال[۵]. این چرخه با انتشارِ یک فراورده‌ی نرم‌افزاریِ[۶] کامل پایان می‌پذیرد. باید ببینیم که در هر یک از فازهای آر.یو.پی چه کارهایی باید انجام شود؟ چه اتفاقاتی در هر یک از این فازها خواهد افتاد؟ در هر فاز چه اهدافی را دنبال خواهیم کرد؟ چه دستاوردهایی تولید می‌شود؟ چه فعالیت‌هایی باید انجام شود؟ در هر فاز چند تکرار خواهیم داشت؟ چه تصمیم‌گیری‌هایی باید انجام شود؟ چه معیارهایی طول زمانی و نوع فعالیت‌های هر فاز را تعیین می‌کند؟ چه تفاوت‌هایی میان مفهوم این فازها و متناظرشان در فرایند آبشاری[۷] وجود دارد؟

در چهار فصل آینده به دنبال پاسخ‌گویی به این سؤالات و بسیاری سؤالات دیگر، پویایی یک پروژه‌ی مبتنی بر فرایندِ آر.یو.پی را بررسی خواهیم نمود. در هر یک از فصل‌های آتی، یکی از فازهای آر.یو.پی تشریح می‌شود. امّا ضروری است پیش از پرداختن به جزئیات مربوط به فازها، نکات مهمّی در رابطه با ساختارِ پویای آر.یو.پی مورد توجه قرار گیرد.

---

[1] - Phase
[2] - Inception
[3] - Elaboration
[4] - Construction
[5] - Transition
[6] - Software Product
[7] - Waterfall

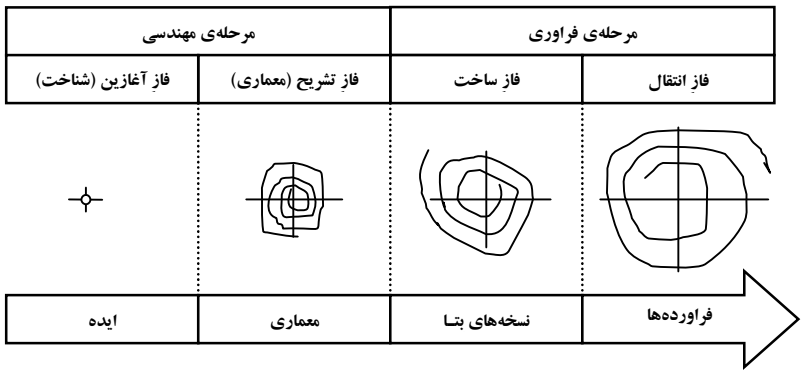
پیش از بررسی دقیق‌ترِ فازهای مختلف و ملاحظات مرتبط با هر یک، لازم است به این نکته توجه نماییم که به طور کلی چرخه‌ی توسعه[1] یا تولید فراورده از منظر چارچوبِ فرایند آر.یو.پی، دارای دو مرحله‌ی[2] اصلی می‌باشد. این مراحل عبارتند از:

۱. مرحله‌ی مهندسی[3]، شامل فازهای آغازین (شناخت) و تشریح (معماری)
۲. مرحله‌ی فرآوری[4]، شامل فازهای ساخت و انتقال

در شکل ۵-۲ نمای کلی این مراحل نشان داده شده است.

شکل ۵-۲

مراحل مهندسی و فراوری در چرخه‌ی تولید فراورده



برخی تصوّراتِ اشتباه

باوجودی‌که چهار فاز آر.یو.پی (آغازین، تشریح، ساخت، و انتقال) به صورت متوالی[5] و پشت سرِ هم اجرا می‌شوند، هیچگاه نباید فراموش کرد که آر.یو.پی اساساً رویکردی تکرارشونده[6] و مبتنی بر ریسک[7] می‌باشد. بسیاری از کسانی که برای اوّلین بار با آر.یو.پی آشنا می‌شوند با یک تصوّر اشتباه و بسیار خطرناک دست به گریبان هستند و آن اینست که این فازها را صرفاً تغییرِ نامِ همان فازهای فرایندهای مبتنی بر رویکردِ آبشاری

[1] - Development Cycle
[2] - Stage
[3] - Engineering Stage
[4] - Production Stage
[5] - Sequential
[6] - Iterative
[7] - Risk-driven

تلقی می‌کنند. بنابراین فاز آغازین را فازی می‌دانند که در آن همه‌ی نیازمندی‌های[1] سیستم مشخص می‌شود، در فاز تشریح، یک طراحی سطح بالا انجام می‌دهند، در فاز ساخت، برنامه‌نویسی و پیاده‌سازی سیستم را انجام داده و نهایتاً در فاز انتقال، تست سیستم را انجام می‌دهند! هر چند ممکن است که در ظاهر از تکرارشونده بودن فرآیند صحبت شود ولی در عمل می‌توانیم روح آبشاری را در برنامه‌ها و فعالیت‌های ایشان شاهد باشیم! این موضوع در بسیاری از پروژه‌های بزرگ و متوسط در کشور که ظاهراً فرآیندی مبتنی آر.یو.پی را دنبال می‌کنند ولی در واقع تنها خروجی‌ها و دستاوردهای آر.یو.پی را پُر می‌کنند! به وفور دیده می‌شود. این موضوع سبب شکست بسیاری از پروژه‌ها بوده است. در تلاش برای بکارگیری آر.یو.پی در یک پروژه، نباید توسعه‌ی تکرارشونده به فراموشی سپرده شود.

درست است که در هفته‌ها یا ماه‌های اوّلِ یک پروژه، تأکید بیشتری بر شناخت نیازمندی‌ها داریم و در طی هفته‌ها یا ماه‌های آخر، تست و رفع نواقص در اولویت قرار می‌گیرد، اما این موضوع منطقاً به دلیل ماهیت نیازها و ریسک‌های موجود در یک پروژه می‌باشد. مهمترین ریسک در شروع یک پروژه، درک و شناخت اَبعاد و ماهیّت مسأله و راه‌حل یا راه‌حل‌های ممکن می‌باشد. قسمت عمده‌ای از این شناخت و درک از طریق تمرکز بر نیازمندی‌ها بدست می‌آید اّما ممکن است بنا به شرایط مسأله، حصول اطمینان از درک کامل ماهیّت مسأله و راه‌کارهای مطلوبِ آن و نیز اثباتِ به‌صرفه‌بودن انجام آن، مستلزم انجام فعالیت‌های متنوع دیگری نیز باشد و لذا در فاز آغازین ممکن است هر یک از فعالیت‌های مختلف تولید نرم‌افزار، اَعَم از تحلیل و طراحی، پیاده‌سازی، تست، و حتی استقرار انجام شود.

بعد از اثبات صحّتِ شناخت و مقرون به صرفه بودن تولید فراورده، مدیریت ریسک‌های فنی در اولویت قرار می‌گیرد. در فاز تشریح (معماری) که فاز دوم از فرآیند آر.یو.پی می‌باشد، عمده‌ی ریسک‌های فنی به کمک طراحی، پیاده‌سازی، و تستِ معماری رفع می‌شوند. در این فاز، بطور معمول فعالیت‌های تحلیل و طراحی درصد بیشتری از فعالیت‌ها را به خود اختصاص می‌دهند ولی کماکان فعالیت‌های مرتبط با شناخت نیازمندی‌ها، پیاده‌سازی، تست، و استقرار هم انجام می‌شوند.

[1] - Requirements

پس از اینکه معماری تثبیت شد و کارایی آن به اثبات رسید، نوبت به ساخت و بنا نمودنِ سیستم است. همانگونه که در ساخت یک ساختمان بطور معمول فعالیت‌های بنّایی حجم زیادی از مرحله‌ی ساخت را به خود اختصاص می‌دهند، مسلماً بیشتر فعالیت‌های ساختِ یک نرم‌افزار نیز به پیاده‌سازی و تست اختصاص دارد. اما سایر فعالیت‌های لازم برای تکمیل ساخت یک نرم‌افزار مانند تکمیل نیازمندی‌ها، تحلیل، طراحی، و استقرار نیز انجام می‌شود. بنابراین در فاز سوم که کم‌کم به انتهای پروژه نزدیک می‌شویم، حجم فعالیتهای پیاده‌سازی و تست بیشتر خواهد شد.

در فاز نهایی که فاز اِنتقال نام دارد، برای اِنتقال سیستمِ آماده شده به محیط مشتری، تدابیر لازم اتخاذ می‌شود. در طولِ این فاز، برای اطمینان از بی‌نقص بودن و تحویل کامل نرم‌افزار به مشتری، آخرین تست‌ها و بررسی‌های لازم روی نرم‌افزار انجام می‌شود. در این فاز هم ممکن است که کماکان پیاده‌سازی مختصری داشته باشیم. حتی، فعالیت‌های مربوط به نیازمندی‌ها، تحلیل، و طراحی نیز ممکن است انجام شود. اما به طور منطقی فعالیت‌های مرتبط با تست و استقرار[1] بیشترین حجم را به خود اختصاص می‌دهند.

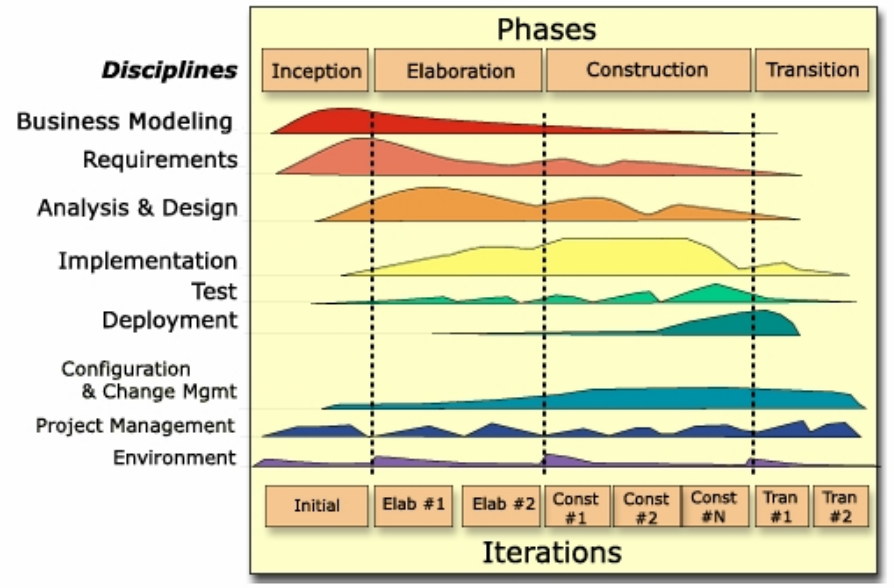
در شکل ۵-۳ که بیانگر شِمایی نمادین از ساختار فرایند است، برآمدگی‌های کوه‌مانند در فازهای مختلف، به همین موضوع، یعنی میزان توجه به دسته فعالیت‌های مختلف (که آن‌را دیسیپلین[2] می‌نامند) اشاره دارد. همان‌گونه که مشاهده می‌شود، سطح برآمدگی‌ها در طول فازهای مختلف متفاوت است. به طور معمول، در داخل هر فاز، تعدادی تکرار[3] وجود دارد و هریک از این تکرارها شامل فعالیت‌های مختلف توسعه‌ی نرم‌افزار، برای تولید یک نسخه‌ی[4] تست شده داخلی یا قابل ارائه به خارج می‌باشد.

---

[1] - Deployment
[2] - Discipline
[3] - Iterations
[4] - Release

شکل ۵-۳

میزان انجام فعالیت‌های مرتبط با دیسیپلین‌ها در فازهای مختلف آر.یو.پی



گام‌های اصلی[1] یا نقاط تصمیم‌گیری سازمانی در فرایند

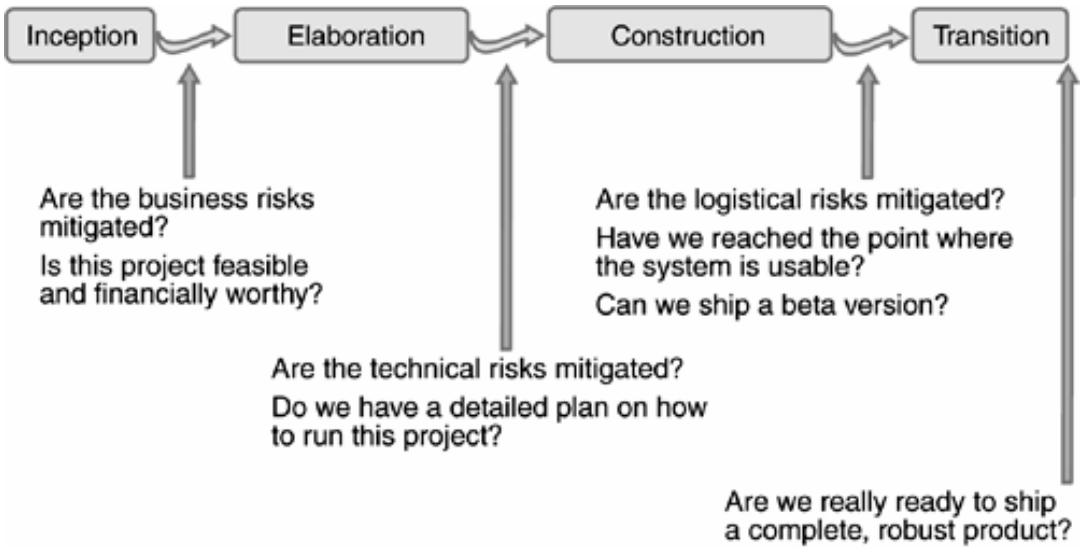
برخلاف رویکرد آبشاری، فازهای آر.یو.پی بر اساس تقسیم‌بندی فعالیت‌ها بر حسب نوع‌شان طرح‌ریزی نشده است. با این توصیف، در آر.یو.پی، فازِ خاصی به صرفاً تحلیل یا تست اختصاص نیافته است؛ بلکه در عوض، در هر فاز که ماهیتی زمانی و مبتنی بر تصمیم‌گیری‌هایی مرتبط با ریسک‌های عمده‌ی مطرح در پروژه دارند، برای دستیابی به اهداف تعیین شده، به اندازه‌ی کافی انواع فعالیت‌های مختلف انجام می‌شود و پس از اینکه نتایج مورد نظر در زمان رسیدن به یک نقطه‌ی تصمیم‌گیری بررسی گردید، تصمیم‌های لازم در باره‌ی خاتمه‌ی آن فاز و یا تمدید زمان آن اتخاذ می‌شود.

آنچه باید در هر فاز بدان دست یابیم، به وسیله‌ی ماهیّت ریسک‌های عمده‌ی مطرح در آن فاز و به منظور رفع این ریسک‌ها تعیین می‌گردد. به عبارت دیگر، فازها در فرآیندی مبتنی بر آر.یو.پی، وضعیت‌های مختلف پروژه را تعریف می‌نمایند و این وضعیت‌ها، خود از منظر ریسک‌های خاصی که باید بر آنها غلبه کنیم یا سؤالاتی که به دنبال پاسخ‌شان می‌گردیم، تعریف می‌شوند.

[1] - Major Milestones

شکل ۵-۴

ریسک‌های مرتبط با هر یک از فازهای آر.یو.پی



بر اساس شکل ۵-۴، هر یک از فازهای آر.یو.پی، به تسکین[۱] ریسک‌های عمده‌ی پروژه بر اساسِ اولویت‌بندی‌شان اختصای دارد. به عبارت دیگر، در هر فاز با ریسک‌های زیر مواجه هستیم:

- در فازِ آغازین[۲] (شناخت) تمرکز اصلی بر مدیریت ریسک‌های مربوط به امکان‌پذیر بودن[۳] و مقرون به صرفه بودن انجام پروژه یا به اصطلاح " قالبِ کسب و کار[۴] " می‌باشد. مهم‌ترین سؤالاتی که در این فاز مطرح‌اند، عبارتند از: آیا انجام پروژه امکان پذیر است؟ آیا پروژه مقرون به صرفه است؟
- در فازِ تشریح[۵] (معماری)، با شناخت کامل‌تر نیازمندی‌ها، تمرکز اصلی بر ریسک‌های فنی، ریسک‌های مربوط به معماری، و احتمالاً بررسی مجدد اَبعاد و محدوده‌ی[۶] پروژه قرار می‌گیرد.
- در فازِ ساخت[۷]، ریسک‌های به اصطلاح لُجستیکیِ[۸] پروژه مورد توجه قرار می‌گیرند. به طور معمول، حجمِ انبوه کار در این فاز انجام شده و بیشترین نیروهای پروژه نیز در طی آن به فعالیت می‌پردازند.
- در فازِ انتقال[۱]، ریسک‌های مرتبط با اِنتقال لُجستیکیِ فراورده به محیط کاربران آن مدیریت می‌شود.

[1] - Mitigation
[2] - Inception
[3] - Feasibility
[4] - Business Case
[5] - Elaboration
[6] - Project Scope
[7] - Construction
[8] - Logistical

گام‌های اصلی که در پایانِ هر فاز قرار دارند، نقاط تصمیم‌گیری سازمانی[۲] نامیده می‌شوند؛ یعنی نقاطی که تصمیم‌های کلیدی در رابطه با ادامه‌ی جریان پروژه، محدوده‌ی پروژه، بودجه[۳]، استراتژی[۴]، تحویل[۵]، و برنامه‌ی زمانی[۶] اتخاذ می‌شود.

همچنین از آنجایی که در یک فاز، نقش‌هایی از انواع مختلف با هم همکاری دارند، بر خلاف رویکرد آبشاری، در یک پروژه‌ی مبتنی بر آر.یو.پی، مانند یک خط لوله که در آن یک گروه از تحلیل‌گران، نیازمندی‌های سیستم را روی یک تاقچه گذاشته و تیمی طراحان آنرا برداشته و سپس طراحی‌ها را به برنامه‌نویس‌ها بدهند و آنها نیز کدهایشان را به مسئولین تست تحویل دهند، اجرا نخواهد شد؛ در یک پروژه‌ی مبتنی بر آر.یو.پی، همه‌ی نقش‌ها در سرتاسر چرخه‌ی تولید فراورده مشارکت دارند (به جز در ابتدای یک پروژه‌ی کاملاً جدید).

نکاتی درباره‌ی جریان‌های کار[۷] در آر.یو.پی

چیزی که در همه‌ی پروژه‌های مبتنی بر آر.یو.پی بدون تغییر باقی می‌ماند، گام‌های اصلی[۸] یا نقاط تصمیم‌گیری سازمانی می‌باشد. آنچه که باید در انتهای هر فاز بدست آید، مهم‌ترین نگرانی همه‌ی اعضای تیم است. البته، این بدان معنا نیست که در آر.یو.پی جریان‌های ثابت کار[۹]، یک نسخه‌ی ثابت یا یک مجموعه فعالیت‌های از پیش تعریف شده‌ی لازم‌الاجرا وجود دارد؛ بلکه آر.یو.پی طیف وسیعی از احتمالات، روش‌ها، و استراتژی‌های ممکن را پیشنهاد می‌نماید و این مشخصات و ویژگی‌های شما و پروژه‌ی شماست که تعیین می‌کند که در طی فازهای مختلف و برای دست‌یابی به اهداف هر فاز، چه کارهایی در عمل باید انجام شود.

از یک چرخه‌ی توسعه به چرخه‌ی توسعه‌ی دیگر یا از یک پروژه به پروژه‌ی دیگر، شرایط متفاوتی حاکم می‌باشد و با ریسک‌های متفاوتی دست و پنجه نرم می‌کنیم. بنابراین، با توجّه به نقطه‌ی شروع پروژه،

---

[1] - Transition
[2] - Business Decision Point
[3] - Funding
[4] - Strategy
[5] - Delivery
[6] - Schedule
[7] - Workflows
[8] - Major Milestones
[9] - Fixed Workflow

اندازه‌ی پروژه، طول زمانی پروژه، و تعداد ذینفعان[۱]، فعالیت‌هایی که حتماً بایستی اجرا شده و دستاوردهایی[۲] که باید تولید شوند، تعیین می‌گردد.

توجه داشته باشید که آر.یو.پی دارای چندین قطعه‌ی مشترکِ فرآیندی[۳] می‌باشد که در طول چرخه‌ی توسعه تکرار می‌شوند. این قطعات مشترک عبارتند از:

- فعالیت‌های شروع و پایان یک پروژه، یک فاز، یا یک تکرار[۴] ، و نیز بازبینی‌ها[۵]
- فعالیت‌های مرتبط با طراحی تفصیلی[۶]، کدنویسی[۷]، تست، و یکپارچه‌سازی[۸] نرم‌افزار
- فعالیت‌های مرتبط با مدیریت پیکربندی[۹]، تولید نسخه‌های[۱۰] مختلف، و مدیریت تغییرات

با این وجود، این موارد نسبت به آنچه باید در انتهای هر یک از فازهای پروژه بدست آید، نکاتی فرعی محسوب می‌شوند. اصلی‌ترین مسأله، دستیابی به اهداف و مقصودهای هر فاز می‌باشد.

یکی از وحشتناک‌ترین حالت‌ها در بکارگیری آر.یو.پی، وقتی است که یک تیم سعی می‌نماید تمام آر.یو.پی را بکارگرفته، همه‌ی فعالیت‌های تعریف شده در آر.یو.پی را انجام دهد و تمام دستاوردهای موجود در آر.یو.پی را تولید کند! باور کنید حتی تصور این موضوع هم وحشتناک است. در اینجا باز هم خاطرنشان می‌کنیم که آر.یو.پی یک فرایند نیست؛ بلکه چارچوبی است برای تعریف فرایندهای تولیدِ نرم‌افزار (یا به طور کلی فرایندِ تولیدِ سیستم). بنابراین لازم است آر.یو.پی با توجه به شرایط خاصِّ هر مسأله، سفارشی‌سازی[۱۱]، پیکربندی[۱۲]، و مناسب‌سازی[۱] شود. ما به فرایندی نیاز داریم که با حداقلِ تشریفات ممکن، برای شرایط پروژه‌ای که در دست داریم، ساده‌تر، مؤثرتر و کاراتر باشد.

---

1 - Stakeholder
2 - Artifact
3 - Common Process Fragments
4 - Iteration
5 - Reviews
6 - Detailed Design
7 - Coding
8 - Integration
9 - Configuration Management
10 - Releases
11 - Customization
12 - Configuration

نکاتی درباره‌ی دستاوردها[2]

در مواردی زیادی خصوصاً در پروژه‌های تعریف شده در کشورمان، دیده شده است که برخی از تیم‌های تولید سعی می‌نمایند که به منظور تسهیل در برنامه‌ریزی و نیز ایجاد احساس پیشرفت و موفقیت در انجام کارها، در طی یک فاز یا حتی در طی یک تکرار، به صورت یکباره، یک دستاوردِ آر.یو.پی (مانند یک مدل، سند، و یا کد) را کامل کرده و آن را غیرقابلِ تغییر[3] تلقی نموده و بایگانی نمایند. در این تیم‌ها با چنین جملاتی برخورد می‌کنیم: "اجازه بدهید نیازمندی‌های را فعلاً کامل کرده و امضاء بگیریم و بعد که خیال‌مان راحت شد، کار را از روی آن‌ها ادامه دهیم" و یا "حالا دیگر طراحی کامل شد".

این هیچ اشکالی ندارد که یک کاری را خوب و کامل انجام داده و مجبور نباشیم دستاورد آن را دوباره بازبینی نماییم. و نیز بسیار خوب و حتی ضروری است که برخی از دستاوردها، یک جایی در طول فرایند تثبیت[4] شوند. اما در فازهای آر.یو.پی در پیِ کامل کردنِ یک یا چند دستاوردِ خاص نیستیم؛ آنچه اهمیت دارد اینست که یک دستاورد را به آن حد از بلوغ[5] برسانیم که قادر باشیم درباره‌ی آن و نیز به کمک آن، تصمیم درستی اتخاذ نماییم؛ تصمیمی که به حال و هوای آن فاز و ریسک‌ها موجود بستگی دارد. در حینِ تکاملِ پروژه و درک اهداف آن و با کشف مشکلات و موانع و نیز مطرح شدن تغییرات بیرونی، دستاوردها را باید مورد بازبینی، به‌روز رسانی، و تصحیح قرار داد. بنابراین فعالیت‌هایی که قبلاً انجام شده بودند، باید دوباره انجام شوند. بالطبع ،اگر که زودتر از موقع مناسب، سعی در آراستن و کامل‌تر کردن یک دستاورد داشته‌باشیم، به احتمال زیاد منجر به دوباره‌کاری بیشتری خواهد شد.

اسنادِ چشم‌انداز[6] و قالبِ کسب و کار[7] که در طی فازِ آغازین (شناخت) تولید می‌شوند، در انتهای فازِ تشریح (معماری) تا حد زیادی کامل‌تر شده و البته در انتهای پروژه تثبیت می‌شوند. نیازمندی‌ها تدریجاً در فازهای آغازین و تشریح (معماری) ساخته و پرداخته شده و باید در انتهای فاز تشریح (معماری) تا حد زیادی

---

[1] - Tailoring
[2] - Artifacts
[3] - Freeze
[4] - Stable
[5] - Maturity
[6] - Vision
[7] - Business Case

کامل شوند. معماری در طول فازِ تشریح (معماری) طراحی و تثبیت می‌شود. اما هر یک از این دستاوردهای اشاره شده ممکن است در طول چرخه با تغییراتی مواجه شوند؛ شما ممکن است چشم‌انداز را تغییر دهید، نیازمندی‌ها را اصلاح کنید، یا حتی طراحی معماری را در فازهای بعد تغییر دهید. اما بدیهی است در صورتی که تغییرات جدید باعث شوند که عملاً تغییری در اطلاعاتِ مبنای تصمیم‌گیری انجام شده در فازهای قبلی بوجود آید، در این صورت احتمالاً باید در برنامه‌ی زمانبندی، هزینه‌ها، محدوده‌ی پروژه، و سایر عوامل مرتبط، تغییراتی اِعمال شود.

با وجودی‌که هیچ یک از دستاوردهای آر.یو.پی اضافی و بی‌مصرف نمی‌باشند و هر یک نقش خاصی در پروژه ایفا می‌نمایند، لازم نیست همه‌ی آن‌ها در یک پروژه تولید نمایید. علاوه‌براین، برای برخی دستاوردهای خاص مانند موارد کاربرد[1]، شما ممکن است برخی از مواردِ کاربرد را به علت حساس بودن و داشتن ریسک‌های زیاد، بطور کامل توصیف نمایید ولی بقیه را به صورت یک توصیف مختصر باقی بگذارید.

یکی دیگر از نکات مهم درباره‌ی دستاوردهای آر.یو.پی اینست که همانگونه که اشاره گردید نه تنها از بین مجموعه‌ی دستاوردهای تعریف شده باید دستاوردهای ضروری و دارای ارزش افزوده را انتخاب کرد، بلکه هریک از دستاوردها نیز باید با توجه به مشخصات و ضروریات پروژه، مناسب‌سازی[2] شوند. برای مثال، سند چشم‌انداز برای همه‌ی پروژه‌ها ضروری است. اما مسلماً سند چشم‌انداز برای یک پروژه کوچکِ سه ماهه با سند چشم‌انداز در یک پروژه‌ی دو ساله متفاوت می‌باشد.

قبل از رفتن به فصل‌های بعدی و آشنایی بیشتر با فازهای فرایند، همواره به یاد داشته باشید که تمرکز هر فاز در رسیدن به یک گام اصلی[3] است. این گام اصلی بیش از هر چیزی با مسأله‌ی تسکین[4] ریسک‌ها، دستیابی به یک‌سری اهداف و مقاصد تعیین شده، و اتخاذ تصمیم‌های کلیدی سر و کار دارد. در نتیجه‌ی فعالیت‌هایی که برای دست‌یابی به اهداف هر فاز انجام می‌شود یک‌سری دستاورد تولید می‌گردد. در همه‌ی

[1] - Use Cases
[2] - Tailoring
[3] - Major Milestone
[4] - Mitigation

فازها باید به یاد داشته باشیم که هدف اصلی، تحویل یک فراورده‌ی نرم‌افزاری دارای کیفیت مناسب برای کاربران و مشتریان است.

منابع و مراجع

[1]. Ivar Jacobson, Grady Booch, James Rumbaugh, (1999). *The Unified Software Development Process*, Reading, MA: Addison-Wesley

[2]. Philippe Kruchten, (2003). *The Rational Unified Process: An Introduction*, Reading, MA: Addison-Wesley.

[3]. Per Kroll, Philippe Kruchten, (2003). *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*, Reading, MA: Addison-Wesley.

[4]. Rational Software Corporation. (2003) "Rational Unified Process 2003.06," Available at: http://www.rational.com/

[5]. Scott W. Ambler, (2000). *The Unified Process Inception Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[6]. Scott W. Ambler, (2000). *The Unified Process Elaboration Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[7]. Scott W. Ambler, (2000). *The Unified Process Construction Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[8]. Software Academy. (2006) "Unified Process Knowledge Base," Available at: http://www.unifiedProcess.info/

[9]. Scott W. Ambler and Larry L. Constantine, (2001). *The Unified Process Transition Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

فصل ششم

فازِ آغازین (شناخت)

مهم‌ترین موضوعات مورد بررسی در این فصل، عبارتند از:

- آشنایی با فازِ آغازین (شناخت)
- اهداف فازِ آغازین (شناخت)
- تکرارها در فازِ آغازین

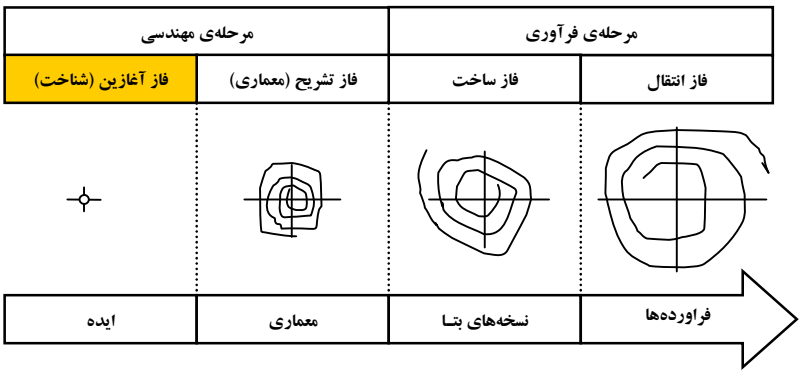
فازِ آغازین (شناخت)

۶

در این فصل، اولین فاز از چرخه‌ی تولید، یعنی فازِ آغازین (شناخت) را معرفی خواهیم نمود. این فاز، اولین فاز از مرحله‌ی مهندسی در چرخه‌ی تولید و نقطه‌ی شروع پروژه می‌باشد.

شکل ۶-۱

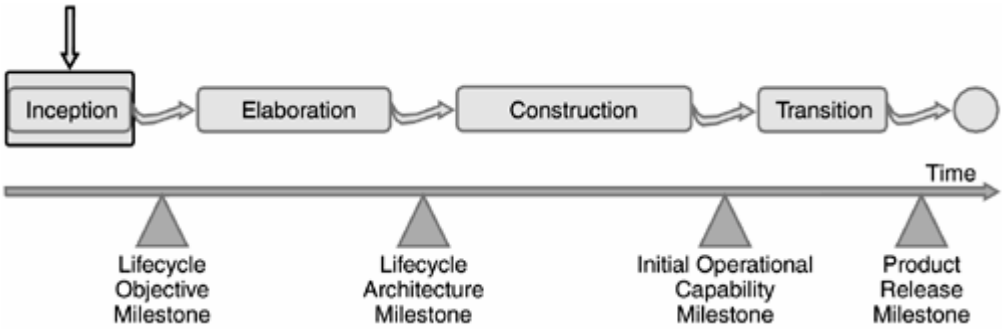
مراحل مهندسی و فرآوری در چرخه‌ی تولید



در این فصل با اهداف و مقصودهای اوّلین فازِ فرایند آشنا خواهیم شد. در بسیاری از محافل آکادمیک و صنعتی در کشور، این فاز را فازِ شناخت نامیده‌اند؛ هر چند این نام‌گذاری اشتباه نیست، اما ما ترجیح دادیم برای جلوگیری از اشتباه‌شدنِ آن با اولین فازِ رویکردِ آبشاری، از واژه‌ی آغازین در کنارِ واژه‌ی شناخت استفاده نماییم.

شکل ۶-۲

فاز آغازین (شناخت)، اولین فاز از چرخه‌ی تولید



بسیاری از کسانی که برای اولین بار با آر.یو.پی آشنا می‌شوند در میان حجم زیاد فعالیت‌ها[1] و راهنمایی‌هایی[2] که به وسیله‌ی آر.یو.پی فراهم شده است، سردرگم می‌شوند. در حالی‌که رویکرد آر.یو.پی بسیار ساده است؛ تنها کافیست تصویر روشنی از اهداف و مقصودهای هر فاز داشته باشیم و بتوانیم نمودِ مناسبی از این اهداف در موقعیت پروژه‌ی خودمان فراهم آوریم.

مسلماً، درک اینکه در یک فاز به چه چیزی می‌خواهیم دست پیدا کنیم، تأثیر بسزایی در بکارگیری هر چه کاراتر رویکرد آر.یو.پی در پروژه دارد. بدین ترتیب قادر خواهیم بود تنها فعالیت‌هایی را برگزینیم که در دستیابی به اهداف مطلوب در پروژه‌ی خاص ما تأثیرگذار باشند.

[1] - Activities
[2] - Guidelines

اهداف فازِ آغازین (شناخت)

فاز آغازین (شناخت)، اولین فاز از فازهای چهارگانه‌ی چرخه‌ی تولید در چارچوبِ فرایندِ آر.یو.پی می‌باشد. بطور کلی، هدفِ اصلی این فاز، رفع و تسکینِ ریسک‌های کلانِ سازمانی[1]، ریسک‌های مربوط به درک اهداف و محدوده‌ی پروژه[2]، و نیز بدست آوردن اطلاعات کافی برای اطمینان از ادامه‌ی پروژه و یا توقف آن می‌باشد.

پنج هدفِ اصلی فازِ آغازین (شناخت)، عبارتند از:

۱. رسیدن به درک مناسبی از راهکاری که باید در طولِ فرایندِ تولید ایجاد شود. تعیین چشم‌انداز[3] و محدوده و مرز سیستم، بایدها و نبایدهای سیستم، آنچه در درون سیستم قرار می‌گیرد و آنچه در بیرون آن، شناسایی کسانی که سیستم را می‌خواهند و اینکه سیستم چه ارزش و جایگاهی برای آنان دارد.
۲. شناسایی وظیفه‌مندی‌های کلیدی[4] سیستم.
۳. تعیین حداقل یک راهکار ممکن. مشخص کردن حداقل یک معماری کاندید[5]
۴. درک هزینه، زمان، و ریسک‌های مرتبط با پروژه.
۵. تصمیم‌گیری درباره‌ی فرایند مناسب و نیز ابزارهای مورد نیاز (برخی از جزئیات مرتبط با فرایند، از جمله فعالیت‌ها و دستاوردها در این فاز مشخص می‌شود).

می‌دانیم که درک کاملِ ابعاد، اهداف، و چیستیِ مسأله، معادلِ حلِ نیمی از آن می‌باشد. تیم تولید در فازِ آغازینِ آر.یو.پی، تمام نیرو و توان خود را صرف درک مسأله می‌نماید. ممکن است در یک پروژه‌ی کوچک، این کار با یک مشاهده و مصاحبه‌ی ساده، انجام شود ولی در پروژه‌های بزرگ به اقداماتی فَرایِ شناختِ نیازمندی‌ها نیاز داریم. در یک پروژه‌ی بزرگ ممکن است فازِ آغازین چندین ماه به طول بیانجامد. با این

[1] - Business Risks
[2] - Scope
[3] - Vision
[4] - Key functionality
[5] - Candidate architecture

توصیف، فاز آغازین در آر.یو.پی، تفاوت‌های بسیاری با فازِ اولِ رویکردِ آبشاری (یعنی همان به اصطلاح فازِ شناخت) دارد.

فازِ آغازین (شناخت) و تکرارها[1]

بسیاری از پروژه‌ها در فازِ شناخت تنها یک تکرار دارند. اما برخی پروژه‌ها برای دستیابی به اهداف مطرح شده‌ی این فاز، باید بیش از یک تکرار داشته باشند.

برخی از مهم‌ترین شرایطی که داشتن چند تکرار را در فازِ آغازین (شناخت) ایجاب می‌نمایند، عبارتند از:

- پروژه دارای ابعاد و اندازه‌ی بزرگی باشد، به‌گونه‌ای که تیم پروژه در تعیین محدوده‌ی سیستم با مشکل روبرو شود.
- سابقه و تجربه‌ی قبلی درباره‌ی سیستم مورد نظر وجود نداشته باشد، به‌گونه‌ای که تعیین دقیق کارهایی که سیستم باید انجام شود، مشکل باشد.
- پروژه دارای ذینفعان متعددی بوده و ارتباطات پیچیده‌ای بین آن‌ها وجود داشته باشد.
- تصمیم‌گیری درباره‌ی ملاحظات سازمانی پروژه، از جمله مسائل مربوط به سود، زیان، و به‌صرفه بودن یا نبودنِ انجام آن با مشکلاتی روبرو باشد. این مورد در رابطه با پروژه‌های جدیدِ تجاری مصداق بیشتری دارد.
- ریسک‌های فنی عمده‌ای وجود داشته باشد که باید بوسیله‌ی یک مدل پیش‌الگو[2] حل و فصل شوند و یا اینکه برای جلب حمایت افراد ذینفع[3]، نیاز به اثباتِ مفهومیِ موضوع[4] باشد. خصوصاً در مواقعی که باید از معماری کاندید، یک مدل پیش‌الگو برای درک بهتر کارایی، هزینه، و دیگر خصیصه‌ها ایجاد شود.

[1] - Iteration
[2] - Prototype
[3] - Stakeholder
[4] - Proof-of-concept

درصورتی‌که بیش از یک تکرار در فازِ آغازین (شناخت) داشته باشیم، تکرار اول عمدتاً بر اهـداف ۱ تـا ۳ ("چیستی") متمرکز شده و بقیه‌ی تکرارها بر اهداف ۴ و ۵ ("چگونگی") تمرکز خواهند داشت.

در ادامه، هر یک از اهدافِ کلیدی فازِ آغازین (شناخت) را بررسی خواهیم نمود. قبل از بررسی اهدافِ کلیدی این فاز، لازم است به این نکته توجه داشته باشید که هیچ‌گونه ترتیبی میان این اهداف وجود ندارد و نیز این اهداف، معادلِ فعالیت‌ها[1]، نمی‌باشند. در واقع، اهدافِ مورد بررسی، حال‌وهوا و مقصودِ تمامیِ فعالیت‌های هر یک از تکرارهای[2] مختلفِ این فاز را بیان می‌نمایند؛ هر فعالیتی که انجام می‌شود، حتماً باید در راستای دستیابی به حداقل یکی از این اهداف باشد، در غیر اینصورت، فعالیت زائدی محسوب می‌گردد.

هدف ۱. درک چیستیِ محصول، (چه چیزی را باید بسازیم؟)

شاید کمی عجیب به نظر برسد ولی علّت عدمِ موفقیتِ بسیاری از پروژه‌ها، نداشتن یک درک مشترک از آنچه نیاز است که ساخته شود، می‌باشد. با وجودی‌که همه‌ی اعضای تیم ممکن است فکر کنند که آنها کاملاً نسبت به موضوعِ پروژه اِشراف دارند، اغلب هر یک درک متفاوتی نسبت به دیگران دارند. اگر می‌خواهید که موفق شوید، همه‌ی ذینفعان باید تعریف مشترکی از موفقیت داشته باشند. ما باید مطمئن شویم که مشتریان، مدیریت، تحلیل‌گران، برنامه‌نویسان، طراحان، معمارها، تست‌کننده‌ها، و دیگر افراد کلیدی، همگی در آنچه که سیستم باید باشد، با هم هم‌عقیده بوده و اِجماع داشته باشند.

برای دستیابی به اطمینان از یک درکِ مشترک باید:

۱. نسبت به یک چشم‌انداز[3] سطح بالا توافق حاصل گردد.
۲. توصیفی جامع و در عین حال سطحی[4] از سیستم ارائه گردد. بطور مختصر کارهایی که سیستم باید انجام دهد در این توصیف بیان شود و از پرداختن به جزئیات پرهیز شود.

---

[1] - Activity
[2] - Iteration
[3] - Vision
[4] - Mile-wide, inch-depth

۳. برخی اَکتورها و مواردِکاربرد[1] کلیدی را تشریح نماییم به‌گونه‌ای که همه‌ی ذینفعان به سهولت آن را درک کرده و اعضای تیم بتوانند از آن به‌عنوان ورودی کارشان استفاده نمایند.

در ادامه‌ی این بحث، برخی اقدامات و فعالیت‌های لازم را برای دستیابی به درک مشترک درباره‌ی سیستم، تشریح می‌نماییم.

تولید یک سندِ چشم‌انداز[2]

برای توافق نسبت به یک چشم‌اندازِ و دیدگاهِ سطح‌بالا، لازم است یک سندِ چشم‌انداز برای پروژه تهیه شود. برای پروژه‌های خیلی کوچک، این سند می‌تواند شکل غیر رسمی داشته باشد. برای مثال، سندِ چشم‌انداز ممکن است به صورت یک نامه‌ی الکترونیکی و یا یک برگه‌ی ساده باشد. برای پروژه‌های متوسط، عموماً لازم است چند صفحه‌ای نوشته شود. صرف نظر از شکل و قالب خاص، یک سند چشم‌انداز باید برای همه‌ی ذینفعانِ[3] پروژه، موارد زیر را تبیین نماید:

- مزایا و فرصت‌هایی که با ایجاد سیستم و فعّال کردن آن، عایدِ ذینفعان می‌شود.
- مسائل یا مشکلاتی که به وسیله‌ی سیستم مورد نظر حل می‌شوند.
- کاربران نهایی[4] و انتظاراتشان از سیستم.
- ارائه‌ی یک توصیف سطح بالا از ویژگی‌ها و سرویس‌های قابلِ ارائه به وسیله‌ی فراورده‌ی نهایی.
- بیان برخی از مهم‌ترین و کلیدی‌ترین نیازمندی‌های غیر وظیفه‌مندی[5] مانند سیستم‌عامل، بانک‌اطلاعاتی مورد نیاز، کیفیت، لیسانس[6] مورد نیاز، قیمت‌گذاری، مقیاس‌پذیری[7].

سند چشم‌انداز زمینه‌ی مناسبی برای درک مشترک انگیزه‌های ایجادِ سیستم و نیز تعریفی سطح بالا از آن فراهم می‌آورد. این سند باید تا پایان فاز آغازین (شناخت) تهیه شده و مورد توافق ذینفعان قرار کیرد. امّا،

---

[1] - Use-Case
[2] - Vision Document
[3] - Stakeholder
[4] - End-user
[5] - Non-Functional Requirements
[6] - License
[7] - Scalability

مسلماً برخی از ملاحظات آن، خصوصاً ملاحظات فنی، در فاز تشریح[1] (معماری) بازبینی و پالایش می‌گردد. حتی ممکن است بخش کوچکی از این سند در ابتدای فازِ ساخت، دچار تحولاتی شود. با این وجود، معمولاً این سند در پایان فاز دوم، یعنی فاز تشریح (معماری)، تثبیت شده و از آن به بعد، دچار کمترین تغییر و تحوّل خواهد شد. نکته‌ی حائز اهمیت اینکه، سند چشم‌انداز یک سند عمومی[2] و قابل دسترس برای کلیه‌ی ذینفعان می‌باشد و لذا هیچ یک از ذینفعان نباید درباره‌ی آن بی‌اطلاع باشند.

ارائه‌ی یک توصیف گسترده و در عین حال سطحی[3]

در فازِ آغازین (شناخت)، لازم است بدون وارد شدن به جزئیات، توصیف خوبی از محدوده‌ی سیستم ارائه گردد. در این توصیف، موارد ذیل حتماً باید لحاظ شود:

- شناسایی و توصیف مختصرِ **اَکتورها**[4] (نهادها و مؤلفه‌های بیرون سیستم که با آن در تعامل هستند، اَعَم از کاربران نهایی سیستم، سازمان‌ها، و یا سایر سیستم‌ها). کاربران نوعیِ سیستم را شناسایی کنید و آن‌ها را بر حسب کاری که انجام می‌دهند و خدماتی که انتظار دارند، دسته‌بندی نمایید. سایر سیستم‌هایی را که با سیستم شما در ارتباط می‌باشند، شناسایی نمایید. این سیستم‌ها نیز اَکتورهایی برای سیستم شما محسوب می‌گردند.
- شناسایی و توصیفِ مختصرِ **مواردِکاربرد**[5] (سرویس‌ها و وظیفه‌مندی‌های سیستم از دیدِ اَکتورهای آن). چگونگیِ تعامل اَکتورها با سیستم را شناسایی و توصیف نمایید. در صورتی‌که اَکتور یک انسان است، روش‌های بکارگیری سیستم توسط اَکتور را توصیف نمایید. توجه داشته باشید که توصیفِ تعامل‌های یک اَکتور با سیستم، موردِکاربرد نامیده می‌شود.

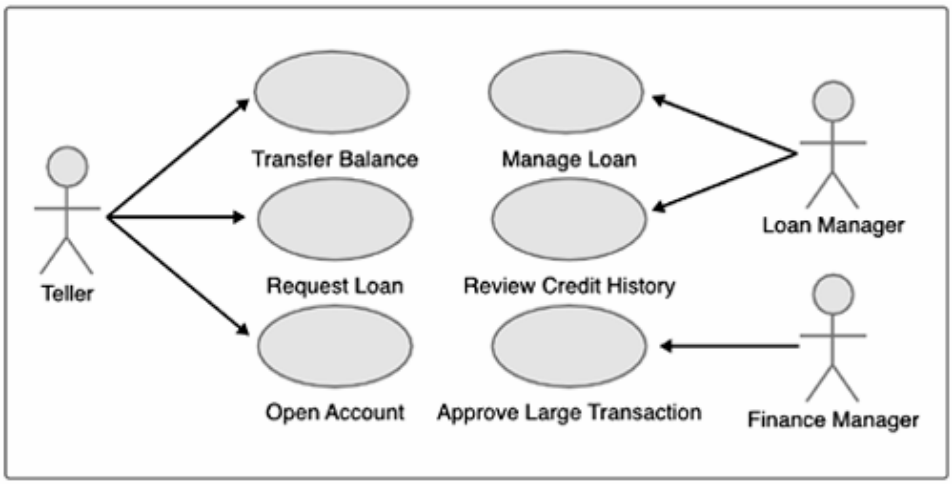
---

[1] - Elaboration
[2] - Public
[3] - Mile-Wide, Inch-Deep
[4] - Actor
[5] - Use-Case

در این مرحله وارد شدن به جزئیاتِ زیاد، ضرورتی ندارد. برای بیشتر مواردِکاربرد، داشتن توصیفی در حد یک یا دو پاراگراف کافی است. البته برای درک کامل‌ترِ مواردِکاربردِ کلیدی و حیاتی[1]، که کمتر از حدودِ بیست درصد از کلِ مواردِکاربرد را به خود اختصاص می‌دهند، باید نسبتاً زمان بیشتری صرف شود و حتی بهتر است که نتایج به صورتِ بصری[2] توصیف گردد.

شکل ۶- ۳

نمونه‌ای از یک دیاگرامِ مواردِکاربرد



برگزاری یک کارگاه[3] یا جلسه‌ی طوفانِ‌مغزها[4]

حال سوالی که ممکن است پیش بیاید این است که چگونه می‌توانیم به بهترین شیوه‌ی ممکن، توصیفی گسترده و در عین حال سطحی از سیستم داشته باشیم؟

در پروژه‌های کوچک، می‌توان یک جلسه‌ی چند ساعته‌ی طوفان‌مغزی با شرکت تیمِ توسعه، مشتری، و نیز احتمالاً سایرِ ذینفعان تشکیل داد. در پروژه‌های بزرگ، می‌توان یک کارگاهِ یک یا دو روزه با مشارکتِ همه‌ی ذینفعان اعم از مدیریت پروژه، معمار، برنامه‌نویسانِ ارشد، مشتری، و تحلیل‌گران برگزار نمود. درباره‌ی جزئیات مرتبط با این جلسه به آر.یو.پی مراجعه نمایید.

---

[1] - Critical Use Cases
[2] - Visual
[3] - Workshop
[4] - Brainstorming Session

تفصیلِ اَکتورها و مواردِکاربرد

یکی دیگر از گام‌های لازم برای درکِ کامل‌تر سیستمی که باید ساخته شود، پالایش برخی از مواردِکاربرد می‌باشد. در پایان جلسه‌ی طوفانِ‌مغزها یا کارگاه برگزار شده، چند موردِکاربردِ اصلی سیستم را برای تشریح جزئیات بیشتر، به تحلیل‌گران می‌دهیم. به موازات تفصیل برخی جزئیات مواردِکاربردِ اصلی، ممکن است لازم باشد که پیش‌الگوی واسطِ کاربر[1] را نیز ایجاد نماییم. به کمک این پیش‌الگو، قادر خواهیم بود تصویری از جریان وقایع[2] در مواردِکاربرد را به مشتری و کاربر نهایی ارائه داده و بدینوسیله به کاربر، تیم توسعه، و دیگر ذینفعان اطمینان دهیم که نسبت به جریان وقایع و روند عملیاتی سیستم درکِ درستی داریم و روی آن توافق نماییم.

البته باید توجه داشته باشیم که تشریحِ جزئیات مواردِکاربرد را در یک بازه‌ی زمانی، محدود کرده و به سراغ جزئیات غیر ضروری در این فاز نرویم. بالطبع، میزان رسمی بودنِ توصیف مواردِکاربرد، با توجه به خصوصیات پروژه‌های مختلف، متفاوت خواهد بود. در این فاز، تنها بایستی به شناسایی و توصیفِ جریان وقایعِ ضروری از مواردِکاربرد توجّه داشته باشیم. توجّه به وجود یا عدم وجودِ آلترناتیوها، برای درک پیچیدگی مواردِکاربرد و تخمینِ بهتر میزان و حجم کار، بسیار مفید می‌باشد.

---

[1] - User-Interface Prototype
[2] - Flow of Events

هدف ۲. شناسایی کارکِردها و وظیفه‌مندی‌های[1] کلیدی سیستم

دومین هدفِ فازِ آغازین (شناخت) مربوط به شناسایی وظیفه‌مندی‌ها و کارکِردهای سیستم می‌باشد. این کار در حینِ شناسایی مواردِکاربرد[2] انجام می‌شود. تشخیص مواردِکاربردِ ضروری و یا آن‌هایی که به لحاظ معماری دارای اهمیت می‌باشند، بسیار مهم است.

مدیر پروژه[3] و معمارِ[4] سیستم باید دوش به دوش هم و ضمن همکاری با سایر ذینفعان[5] (مانند مشتری و کاربران نهایی)، مواردِکاربردِ حیاتی سیستم را شناسایی نمایند. درنظر گرفتن نکات ذیل برای شناسایی دقیق‌ترِ مواردِکاربرد مهمِّ سیستم، مفید می‌باشد:

الف – توجه به وظیفه‌مندی‌ها و کارکِردهایی که جزء بنیادی و کلیدی در سیستم بوده و با واسطهای اصلی سیستم در ارتباط می‌باشند. این قبیل وظیفه‌مندی‌ها، اثرِ مهمی بر شکل‌گیری معماری سیستم دارد. معمارِ سیستم می‌تواند با استفاده از تکنیک‌هایی نظیر استراتژی‌های مدیریت افزونگی[6]، مخاطرات درگیری منابع[7]، مخاطرات مربوط به کارایی[8] ، و استراتژی‌های امنیت داده[9]، این دسته از مواردِکاربرد را شناسایی نماید. برای مثال، در یک سیستم فروش الکترونیکی، پرداخت و در یک سیستم حقوق و دستمزد، محاسبه‌ی حقوق، از مواردِکاربرد اصلی می‌باشند که نقشی تعیین کننده در شکل‌گیریِ معماری سیستم ایفا می‌نمایند.

ب – توجه به وظیفه‌مندی‌ها و کارکِردهایی که بدون آنها سیستم معنایی نخواهد داشت و یا قابل ارائه و تحویل نخواهد بود. معمولاً افراد خبره‌ی[10] آشنا با حوزه‌ی بکارگیری سیستم، قادر به تشخیص این دسته از وظیفه‌مندی‌های سیستم می‌باشند. ملاحظاتی از قبیل رفتارهای اولیه‌ی سیستم، حداکثرِ تراکنش‌های

---

[1] - Functionality
[2] - Use-Case
[3] - Project Manager
[4] - Architect
[5] - Stakeholders
[6] - Redundancy Management Strategies
[7] - Resource Contention Risks
[8] - Performance Risks
[9] - Data Security Strategies
[10] - Domain Experts

داده‌ای و تراکنش‌های کنترلیِ حیاتی، در شناسایی این نوع وظیفه‌مندی‌ها مفید می‌باشد. برای مثال یک سیستم مدیریت سفارش‌ها را بدون قابلیت واردسازی سفارش‌ها نمی‌توان تحویل مشتری داد!

ج ـ درنظر گرفتن وظیفه‌مندی‌هایی که بخشی از معماری را می‌پوشانند ولی در قالب هیچ یک از مواردِکاربردِ[۱] حیاتی[۲] شناسایی شده، قرار نمی‌گیرند.

در یک سیستم که حدود بیست موردِکاربرد دارد، بطور متوسط سه یا چهار موردِکاربردِ حیاتی دارد. در طولِ فازِ آغازین[۳] (شناخت)، درک مواردِکابردِ حیاتی سیستم، بسیار ضروری است. بنابراین، تا آنجا که اِمکان دارد باید به توصیف جزئیات کافی از این مواردِکاربرد بپردازیم. البته همانگونه که پیش از این نیز بیان گردید، توصیف کاملِ جزئیات و از جمله تشریح آلترناتیوها[۴] را به فازهای بعدی موکول می‌نماییم.

مواردِکاربردِ حیاتی را در سندِ معماری نرم‌افزار[۵] که از اسناد و دستاوردهای[۶] مهم پروژه می‌باشد، ثبت می‌نماییم.

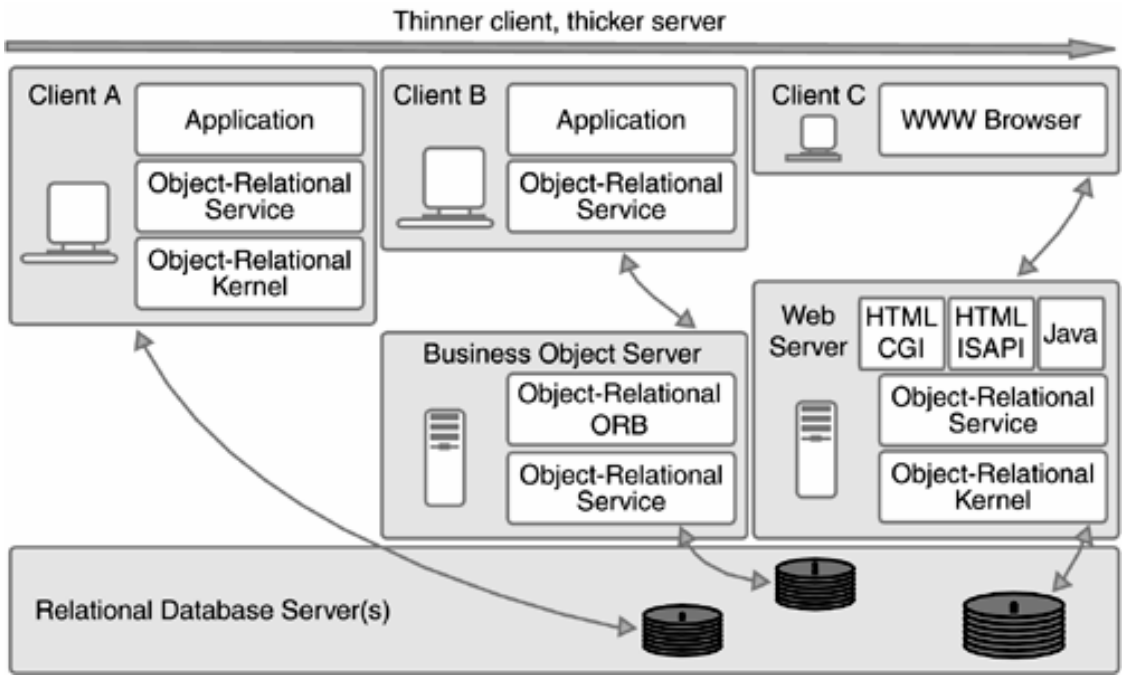
[1] - Use-Case
[2] - Critical Use-Cases
[3] - Inception
[4] - Alternative
[5] - Software Architecture Document
[6] - Artifact

هدف ۳. شناسایی حداقلْ یک راهکارِ امکان‌پذیر[1]

از آنجایی که یکی از مهم‌ترین اهداف فازِ آغازین (شناخت)، اثباتِ منطقی بودن ادامه‌ی پروژه می‌باشد، لازم است که حداقل، یک معماری بالقوه در اختیار داشته باشیم که بتوان به وسیله‌ی آن سیستمی با هزینه‌ی منطقی و میزان قابل قبولی از ریسک‌ها ایجاد نمود. به عنوان مثال، ممکن است برای ایجاد یک معماری client/server سه آلترناتیو در نظر داشته باشیم؛ با تحلیل کارکِردها و وظیفه‌مندی‌های مطلوب در نسخه‌ی اول و همچنین نسخه‌های بعدی سیستم، قابلیت تطابق با سایر سیستم‌ها و نیازمندی‌های خاص عملیاتی و نگهداری سیستم، ممکن است هر کدام از این سه آلترناتیو را انتخاب نماییم.

شکل ۶- ۴

نمونه‌ای از سه آلترناتیو در یک معماری با توجه به ملاحظات مختلف



[1] - Determine at Least One Possible Solution

برای اطمینان از این موضوع، طرح پرسش‌های ذیل مهم می‌باشد:

- آیا سیستم‌های مشابه دیگری ساخته شده‌اند، در این صورت این سیستم‌ها کدام‌اند و از چه معماری یا فناوری بهره گرفته‌اند؟ هزینه‌ی شما برای استفاده از هریک از این فناوری‌ها چه مقدار می‌باشد؟
- در ارتباط با یک سیستمِ درحالِ تکامل، آیا معماری فعلی برآورده کننده‌ی نیازهای جدید می‌باشد یا اینکه باید متحوّل شود؟
- چه فناوری‌هایی را باید در توسعه‌ی سیستم بکار گرفت؟ آیا فناوری‌های جدیدی را باید تهیه نمود؟ هزینه و ریسک‌های مرتبط با هریک از انتخاب‌های موجود چیست؟
- چه مؤلفه‌های نرم‌افزاری (بانک اطلاعاتی، میان‌افزارها، و مانند آن) در سیستم لازم است؟ آیا باید آنها را خریداری نمود؟ آیا می‌توان از پروژه‌های موجودِ فعلی، آنها را موردِ استفاده‌ی مجدد قرار داد؟ هزینه‌ی تخمینی چه میزانی است؟ چه ریسک‌هایی در این رابطه وجود خواهد داشت؟

در برخی از موارد، برای درک بهتر آلترناتیوهای موجود و ریسک‌های مرتبط، لازم خواهد بود که بعضی از عناصر کلیدی معماری و یا حتی معماری‌های پیشنهادی مختلف، پیاده‌سازی شود. در پروژه‌هایی که تصوّر محصول نهایی برای ذینفعان آن مشکل است، باید پیش‌الگوهایی[1] که دارای قابلیت‌های عملیاتی هستند، تهیه شود.

در پایانِ فازِ آغازین (شناخت) باید اطلاعات کافی درباره‌ی ریسک‌هایی که با آنها روبرو خواهیم شد، داشته باشیم. خصوصاً در رابطه با ریسک‌های مرتبط با فناوری و دارایی‌های قابل استفاده‌ی مجدد، مانند چارچوب‌های معماری و بسته‌های نرم‌افزاری. در فازِ بعدی که آن‌را فازِ تشریح (معماری) می‌نامند، جزئیات مرتبط با معماری (راهکار) تشریح و تثبیت خواهد شد.

[1] - Prototype

هدف ۴. کسب اطلاعات بیشتری در رابطه با هزینه‌ها[۱] ، زمانبندی[۲] ، و ریسک‌های[۳] پروژه

فهمیدن اینکه چه چیزی را باید بسازیم، مهّم است، امّا تعیین چگونگی و میزانِ هزینه‌ی مورد نیاز برای ایجادِ آن نیز بسیار مهم و سرنوشت‌ساز می‌باشد. برای اینکه مشخص نماییم که آیا باید پروژه را ادامه دهیم یا نه، لازم است اطلاعاتی در رابطه با هزینه‌ی پروژه کسب نماییم. بیشتر هزینه‌ها به منابع مورد نیاز و نیز زمانِ پروژه مربوط می‌باشد. با ترکیب همه‌ی اطلاعات مرتبط با منابع و زمان به علاوه‌ی قابلیت‌های مورد انتظار از فراورده، می‌توان یک قالبِ کسب و کار[۴] برای پروژه ایجاد نمود. در این قالبِ کسب و کار، ارزشِ اقتصادیِ فراورده[۵] به صورت کمّی و مثلاً با نشان دادنِ بازگشت سرمایه[۶]، بیان می‌شود. ریسک‌های رفع نشده‌ی بزرگِ پروژه نیز معیارهایی برای براورد هزینه می‌باشند.

در بسیاری از سازمان‌ها، خصوصاً در سازمان‌هایی که اداره‌ی فناوریِ اطلاعات[۷] در داخل سازمان قرار دارد، بودجه‌ی مشخصی به پروژه اختصاص می‌یابد. در چنین حالت‌هایی باید بررسی شود که به تناسب این بودجه و محدوده‌ی زمانی، چه چیزی را می‌توان ارائه نمود.

---

[1] - Costs
[2] - Schedule
[3] - Risks
[4] - Business Case
[5] - Economic Value of Product
[6] - Return on Investment (ROI)
[7] - Inofmation Technology (IT)

هدف ۵. تصمیم‌گیری درباره‌ی مشخصه‌های فرایند[1] و ابزارهای مورد نیاز

داشتن یک دیدگاهِ مشترک میان همه‌ی افراد تیم در رابطه با چگونگیِ تولید نرم‌افزار، یعنی فرایندِ تولید[2]، بسیار مهم است. باید دقت داشته باشید که فرایندِ تولید، متناسب با مشخصه‌ها و ابعاد پروژه انتخاب شده و کمترین سربارِ اضافی را بر تیم تحمیل نماید. در پروژه‌های کوچک، به راحتی می‌توان در طول مسیر در رابطه با مشخصه‌های فرایند تصمیم‌گیری نمود، اما در پروژه‌های بزرگ لازم است خیلی زودتر و در همان اوایلِ پروژه، انتخاب‌های ممکن را بررسی و فرایندِ مناسبی برای پروژه انتخاب شود.

مهندس فرایند[3]، که یکی از نقش‌های کلیدی در سازمان و پروژه می‌باشد، پیش از آغاز پروژه با اطلاعاتی که از مشخصه‌های پروژه دریافت می‌نماید، شکل و شِمایِ کلّی فرایند را تعریف می‌کند. بدیهی است به هر میزان که اطلاعات بیشتری در رابطه با کمّ و کیفِ پروژه وجود داشته باشد، فرایند متناسب‌تری برای پروژه انتخاب می‌گردد. اما در اکثر پروژه‌ها، این اطلاعات قبل از شروع پروژه یا وجود ندارند و یا در طول پروژه با تغییرات بسیاری مواجه خواهند شد و در واقع، تنها با وارد شدن به چرخه‌ی تولید، اِمکان دسترسی به اطلاعات و ملاحظات مرتبط با شرایط پروژه فراهم می‌گردد. آر.یو.پی، این امکان را برای شما فراهم نموده است که بتوانید در طولِ چرخه‌ی تولید[4]، فرایند را با ویژگی‌های مطلوبِ پروژه تطبیق دهید. این کار عمدتاً در فازهای اول و دوم انجام می‌شود. البته، در صورتی‌که مشتری یا کارفرمای پروژه، نسبت به این موضوع توجیه نشده باشد، ممکن است چالش‌ها و مشکلاتی به وجود آید که برای جلوگیری از آنها باید این موضوع که فرایند خود مفهومی است پویا و قابل تطبیق، به روش مناسبی مورد توافق قرار گیرد.

تصمیم‌گیری در رابطه با بکارگیری برخی از ابزارها نیز می‌تواند در همین فاز انجام شود. البته این تصمیم‌گیری پس از انتخاب فرایندِ مناسب انجام می‌شود. در برخی از موارد، ابزارهای مورد نیاز، از قبل در سازمان وجود دارد. البته ممکن است با توجه به ملاحظات خاصِ یک پروژه، یک یا چند ابزار جدید هم به

---

[1] - Process
[2] - Development Process
[3] - Process Engineer
[4] - Development Cycle

محیطِ تولید[1]، اضافه شود. در برخی از موارد ممکن است که ابزارهای انتخاب شده نیاز به آموزش، سفارشی‌سازی، و پیکربندی[2] داشته باشند. برخی از مهم‌ترین ابزارهای مورد نیاز عبارتند از: ابزارهای مدیریت نیازمندی‌ها، مدل‌سازی بصری، مدیریت پیکربندی و تغییرات، و ابزارهای تست.

توجه داشته باشید با وجودی که آر.یو.پی ابزارهای خاصِ شرکت رَشنال[3] را معرفی نموده است، هیچ لزومی ندارد که از این ابزارها استفاده کنید. در انتخاب ابزارِ مناسب، بعد از درکِ جایگاه و ضرورتِ آن در فرایند، باید نکات خاصی مانند مجموعِ هزینه‌های مالکیت[4]، یادگیری، به‌روز رسانی، و نگهداری را لحاظ نمایید.

بازبینی پروژه: نقطه‌ی تصمیم‌گیری[5] شناخت اهداف چرخه‌ی حیات[6]

در پایانِ فازِ آغازین، به اولین نقطه‌ی تصمیم‌گیری کلیدی یا سازمانی می‌رسیم؛ این نقطه، اصطلاحاً نقطه‌ی تصمیم‌گیری یا گامِ اصلیِ اهدافِ چرخه‌ی حیات فراورده نام دارد. در این نقطه، اهدافِ کلیدی فراورده و پروژه مورد بررسی قرار می‌گیرد. اگر پروژه نتواند به اهداف این نقطه دست یابد، یا باید متوقف گردد یا اینکه مورد بازنگری مجدد واقع شود. اگر پروژه‌ی شما واقعاً دارای مشکل و ابهام است، بهتر است که این موضوع هرچه زودتر مشخص شود.

در بازبینی اهدافِ چرخه‌ی حیاتِ فراورده (یا پروژه)، که در بسیاری از موارد، یک یا چند جلسه‌ی مشترک با همه‌ی ذینفعان می‌باشد، معیارهای ارزیابی زیر در نظر گرفته می‌شود:

- توافق ذینفعان در رابطه با تعریفِ محدوده و قلمرو پروژه و نیز یک تخمین اولیه از هزینه و زمان (که در فازهای بعد پالایش می‌شود)

[1] - Development Environment
[2] - Configuration
[3] - Rational
[4] - Total Cost of Ownership or TCO
[5] - Milestone
[6] - Lifecycle Objectives

- توافق و تفاهم در رابطه با اینکه مجموعه‌ی درست و مناسبی از نیازمندی‌ها اخذ شده و درک مشترکی از این نیازمندی‌ها وجود دارد.
- تفاهم روی اینکه تخمین هزینه و زمان، اولویت‌ها، ریسک‌ها، و نیز فرایند تولید، مناسب و بجا هستند. دقت کنید که صرفِ درنظر گرفتنِ آر.یو.پی به عنوان فرایند به هیچ وجه کافی نیست، در واقع همان‌گونه که پیش از این نیز بارها تأکید شد، آر.یو.پی یک چارچوب برای تعریف فرایند است، فرایند مناسبِ هر پروژه را باید با توجه به این چارچوب، تدوین نمود.
- تفاهم روی اینکه ریسک‌های اولیه شناخته شده‌اند و یک استراتژی مناسب برای تسکین هر یک از آنها وجود دارد.

درصورتی‌که پروژه‌ی شما نتواند در رابطه با معیارهای بازبینی، موفق باشد، یا باید متوقف گردد، یا اینکه تجدیدِ نظر کلی در رابطه با آن انجام شود. توجه داشته باشید که در این صورت به هیچ وجه نباید وارد فاز بعد یعنی تشریح (معماری) شوید، تجدیدِ نظر ممکن است با اضافه‌کردن یک تکرار به فازِ آغازین تحقّق یابد.

چکیده‌ی فصل

این فصل به معرفیِ فازِ آغازین (شناخت) و بررسی برخی از جزئیات مربوط به آن اختصاص داشت. فاز آغازین[1] (شناخت) اولین فاز از فازهای چهارگانه‌ی چرخه‌ی فرآیند تولیدِ یک سیستم می‌باشد. مقصود اصلی این فاز، شناخت مسأله، محدوده و جوانب مختلف آن، و نیز اطمینان از داشتن حداقل یک راه‌حل ممکن و نیز امکان‌پذیری پروژه می‌باشد. در پایانِ این فاز، با یک تصمیم‌گیری مهم در اثبات درکِ مسأله مواجه می‌باشیم. در این فصل، ملاحظات بیان شده را در قالب پنج هدفِ اصلی برای فاز آغازین (شناخت)، بیان کردیم؛ این اهداف عبارتند از:

- درک چیزی که باید ساخته شود.
- شناسایی وظیفه‌مندی‌های کلیدی سیستم.
- تعیین حداقل یک راهکار ممکن.
- درک بیشتر هزینه‌ها، زمان، و ریسک‌های مرتبط با پروژه.
- تصمیم‌گیری درباره‌ی فرایند و ابزارهای لازم.

تمرکز بر این اهداف، مانع از گُم شدن در میان حجم زیاد فعالیت‌ها و رهنمودهای فراهم شده بوسیله‌ی آر.یوپی می‌شود. از رهنمودهای فراهم شده برای دستیابی بهینه‌تر و سریع‌تر به اهداف ذکر شده استفاده نمایید. توجه داشته باشید که هیچ‌گاه نباید سعی کنیم که در یک پروژه، تمامِ دستاوردهای آمده در آر.یو.پی را تولید نماییم؛ تنها دستاوردهایی را باید تولید کنیم که ما را در دستیابی به اهداف مورد نظر در هر فاز یاری دهند.

در پایان فازِ آغازین (شناخت)، با داشتن درکِ خوبی از مسأله و داشتن حداقل یک راهکارِ مناسب، بر مهم‌ترین ریسک‌های پیش‌رو که مربوط به شناخت مسأله می‌باشند، فائق آمده و حال، آماده هستیم که به فاز دوم رفته و با ریسک‌های مربوط به راه‌حل یا همان ریسک‌های فنی رویارو شویم.

[1] - Inception

پرسش‌هایی برای تحقیق و مطالعه‌ی بیشتر

۱. درباره‌ی تفاوت‌های فازِ آغازین در آر.یو.پی و فازِ شناختِ نیازمندی‌ها در رویکرد آبشاری تحقیق نمایید.

۲. در فازِ آغازین، عمدتاً چه نوع ریسک‌هایی مدیریت می‌شوند؟

۳. آیا ممکن است فازِ آغازینِ یک پروژه بسیار کوتاه باشد؟ در صورت مثبت بودن پاسخ، در چه مواردی؟

۴. درباره‌ی جزئیات برگزاری جلسه‌ی طوفانِ‌مغزی تحقیق نمایید.

۵. با بررسی مفهوم معماریِ جامعِ سازمانی[1]، چگونگی توسعه‌ی آن در فاز آغازین را بررسی نمایید.

۶. محتوای سند چشم‌انداز را بررسی نمایید.

[1] - Business Architecture

منابع و مراجع

[1]. Ivar Jacobson, Grady Booch, James Rumbaugh, (1999). *The Unified Software Development Process*, Reading, MA: Addison-Wesley.

[2]. Robert L. Glass, (2002). *Facts and Fallacies of Software Engineering*, Reading, MA: Addison Wesley.

[3]. Scott E. Donaldson, Stanley G. Siegel, (2000). *Successful Software Development*, Reading, NJ: Prentice Hall PTR.

[4]. Pressman, R. S. (2000). *Software engineering: A practitioner's approach*. 5th ed. New York: McGraw-Hill.

[5]. Philippe Kruchten, (2003). *The Rational Unified Process: An Introduction*, Reading, MA: Addison-Wesley.

[6]. Per Kroll, Philippe Kruchten, (2003). *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*, Reading, MA: Addison-Wesley.

[7]. Rational Software Corporation. (2003) "Rational Unified Process 2003.06," Available at: http://www.rational.com/

[8]. Software Academy. (2006) "Unified Process Knowledge Base," Available at: http://www.unifiedProcess.info/

[9]. Scott W. Ambler, (2000). *The Unified Process Inception Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

فصل هفتم

فازِ تشریح (معماری)

مهم‌ترین موضوعات مورد بررسی در این فصل، عبارتند از:

- ماهیّت فازِ تشریح (معماری)
- آشنایی با اهداف فازِ تشریح (معماری)
- بررسی اهمّیّت و جایگاهِ معماری در فرایند

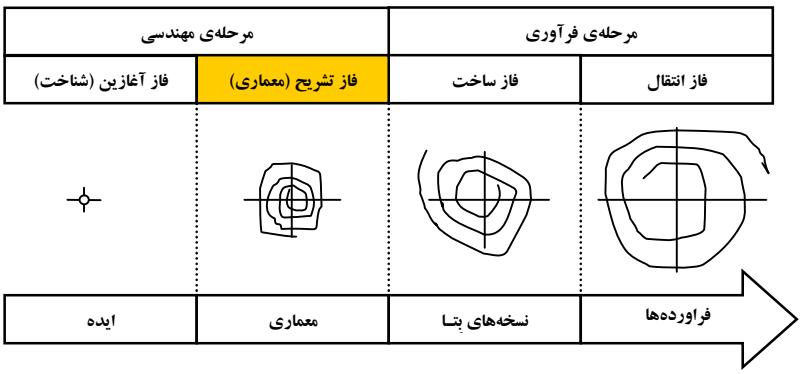
فازِ تشریح (معماری)

۷

این فصل به معرفی دوّمین فاز از چرخه‌ی تولید فراورده، یعنی فازِ تشریح یا معماری که آخرین فاز از مرحله‌ی مهندسی است، اختصاص دارد. بهترین منبع و مرجع برای توصیف کاملِ فاز تشریح، خودِ آر.یو.پی است. بنابراین در این فصل از کتاب، به معرفی برخی از مهم‌ترین نکات مرتبط با این فاز، اِکتفا خواهیم نمود. مطالب تکمیلی مرتبط با این فاز را در آر.یو.پی پیگیری نمایید.

شکل ۷-۱

فاز تشریح، دومین فاز از مرحله‌ی مهندسی در چرخه‌ی تولید



یکی از خصوصیات جالب و قابلِ تأملْ در رابطه با این فاز، تفاوت بسیار زیاد آن با دومین فاز از رویکرد آبشاری[1] (یعنی فازِ طراحی) می‌باشد. البته فراموش نکنیم که ماهیّت تمام فازهای آر.یو.پی به طور کامل، با ماهیّت و فضای موجود در فازهای سنتی رویکردِ آبشاری، متفاوت می‌باشد. اما تفاوت‌ها در این فاز بسیار چشمگیرتر و برجسته‌تر است.

[1] - Waterfall

به منظور معرفی فازِ تشریح (معماری)، اهداف و مقصودهای[1] تعریف شده برای این فاز را بررسی خواهیم نمود. بدین ترتیب دوباره این نکته را خاطر نشان می‌نماییم که در هر پروژه‌ای، با هر اَبعـاد و انـدازه‌ای، تنهـا مفاهیمِ ثابت از نظر آر.یو.پی، اهدافِ مورد انتظار در فازها می‌باشد. شما صرفاً باید فعالیت‌هایی را انجام دهیـد که بیشترین تأثیر را در دستیابی به این اهداف دارند. بدین ترتیب، از انجام فعالیـت‌هـای اضـافی و یـا تولیـد دستاوردهای غیر ضروری که تنها موجب افزایش هزینه و طولانی‌تر شدن زمان پروژه می‌شوند، پرهیز نمایید.

همواره به یاد داشته باشید که تصمیم‌گیری در رابطه با میزان رسمیّت[2] دسـتاوردها نیـز بـسیار ضـروری است. بدیهی است که بر این اساس ممکن است برخی از دستاوردها روی یک وایتْبُرد، یـک مـدل، و یـا در قالبِ یک سندِ رسمی تولید شوند.

فازِ تشریح (معماری) به نقطه‌ی تصمیم‌گیری در رابطه با تثبیتِ معماری ختم می‌گردد. در این فاز، غلبـه بر ریسک‌های فنی با تثبیت و مبنا قرار دادنِ معماری سیستم، امکان‌پذیر می‌گردد.

اینکه، آر.یو.پی فاز دوم از چرخه‌ی تولید را به تثبیتِ معماری اختصاص داده‌است، نشان می‌دهد که تثبیتِ معماری، نقش کلیدی و بسیار مهمّـی در موفقیـت پـروژه دارد. در ایـن فـاز، معمـاری سیـستم بـا توجـه بـه نیازمندی‌هایی که از نظر معماری، قابل ملاحظه[3] می‌باشند و نیز بر اساس ارزیابی ریسک‌ها، و نیز ملاحظـاتی مانندِ محدودیت‌های هزینه و زمان، شکل گرفته و پس از انجامِ تست و آزمون‌های لازم، تثبیت می‌گردد.

در طولِ این فاز، در پی تسکین ریسک‌های ذیل می‌باشیم:

- ریسک‌های مرتبط با نیازمندی‌ها (اینکه آیا در حالِ پیاده‌سازیِ سیستمِ درستی[4] هستیم یا نه؟)
- ریسک‌های مرتبط با معماری (اینکه آیا راهکارِ مناسبی را می‌سازیم یا نه؟)
- ریسک‌های مرتبط با هزینه و زمان (آیا واقعاً طبق برنامه هستیم؟)

---

[1] - Objectives
[2] - Formal
[3] - Architecturally Significant Requirements
[4] - Right Application

- ریسک‌های مرتبط با فرایند، ابزارها، و محیط (اینکه آیا فرایند و ابزارهـای مناسـبی بـرای پـروژه در اختیار داریم و یا نه؟)

تنها پس از اطمینان از تسکین و از بین رفتنِ ریسک‌های فوق، می‌توانیم قدم در فـاز بَعـدی، یعنـی فـازِ ساخت، بگذاریم. اهدافِ اصلی فازِ تشریح (معماری) که متناظر با ریسک‌های فوق و برای تسکین آنها تعریف شده است. عبارتند از:

۱. درک جزئیاتِ بیشتری از نیازمندی‌ها
۲. طراحی، پیاده‌سازی، اعتبار سنجی[1]، و مبنا قرار دادن (تثبیت) معماری
۳. تسکین ریسک‌های عمده و بهبود تخمین‌های هزینه و زمانِ پروژه
۴. پالایش قالب و الگوی فرایند تولید[2] و آماده‌سازی بستر، محیط، و ابزارهای مناسب

[1] - Validate
[2] - Development Case

فازِ تشریح (معماری) و تکرارها[1]

بسیاری از پروژه‌ها در فازِ تشریح دارای بیش از یک تکرار هستند. با توجه به میزان ریسک‌های موجود که عمدتاً از نوع ریسک‌های فنی می‌باشند، تعداد و طولِ زمانیِ تکرارها متفاوت می‌باشد.

برخی از مهم‌ترین شرایطی که داشتن چند تکرار را در فازِ تشریح ایجاب می‌نمایند، عبارتند از:

- پروژه دارای پیچیدگی زیادی باشد به گونه‌ای که درک آن با مشکل مواجه باشد.
- فناوری‌های جدیدی استفاده شود.
- پروژه دارای ذینفعان متعددی باشد و ارتباطات پیچیده‌ای بین آن‌ها وجود داشته باشد.
- تولیدِ سیستم به صورت توزیع‌شده[2] انجام شود.
- قراردادهای پیچیده و خاصی وجود داشته باشد.
- نیاز به تطابق با قوانین و استانداردهای بیرون از سازمان وجود داشته باشد.

اگر تجربه‌ی قبلی در فناوری مورد استفاده وجود داشته باشد یا اینکه پروژه‌ی فعلی چرخه‌ی تکامل از یک فراورده‌ی قبلی باشد که در آن معماری چندان تغییری نمی‌کند، عموماً با ریسک‌های کمتری در این فاز روبرو خواهیم بود و لذا برنامه‌ریزیِ یک تکرار در این فاز کافی خواهد بود.

قبل از بررسی اهدافِ کلیدی فازِ تشریح، لازم است به این نکته توجه داشته باشید که هیچ‌گونه ترتیبی میان این اهداف وجود ندارد و نیز این اهداف، معادلِ فعالیت‌های قابلِ انجام، نمی‌باشند. در واقع، اهدافِ مورد بررسی، حال‌وهوا و مقصودِ تمامیِ فعالیت‌های هر یک از تکرارهای[3] مختلف این فاز را بیان می‌نمایند؛ هر فعالیتی که انجام می‌شود، حتماً باید در راستای دستیابی به حداقل یکی از این اهداف باشد؛ در غیر اینصورت، فعالیت زائدی محسوب می‌گردد.

---

[1] - Iteration
[2] - Distributed Development
[3] - Iteration

هدف ۱. درک جزئیات بیشتری از نیازمندی‌ها

در انتهای فاز قبل، یعنی فازِ آغازین[1] (شناخت)، باید یک چشم‌انداز[2] خوب از پروژه و نیز توصیف جزئیات مربوط به حدود ۲۰ درصد از مواردِ کاربردِ[3] سیستم، بدست آمده باشد. البته، برای بقیه‌ی مواردِ کاربردِ شناخته شده در فاز اول، توصیف مختصری نیز ارائه شده است.

در انتهای فاز تشریح[4] (معماری)، باید بخشِ عمده‌ای از مواردِ کاربردِ سیستم، شناخته شده و توصیف شده باشند. برخی از مواردِ کاربرد بسیار ساده‌اند و یا اینکه شباهت زیادی به بقیه‌ی موارد کاربرد تشریح شده دارند، شاید تفاوت‌شان این باشد که روی داده‌های متفاوتی کار می‌کنند. توصیفِ این مواردِ کاربرد را می‌توان به فازِ بَعد (یعنی فازِ ساخت[5]) موکول نمود. البته، ممکن است هیچ‌گاه نیازی به توصیف رسمی این دسته از مواردِ کاربرد نباشد. توجّه داشته باشید که تشریحِ جزئیات چنین مواردِ کاربردی، هیچ ریسک عمده‌ای را موردِ خطاب قرار نمی‌دهد.

همچنین در صورت لزوم، باید یک پیش‌الگو[6] برای واسطِ کاربر[7] مربوط به هر یک از مواردِ کاربرد عمده‌ی سیستم تهیه شود. هر یک از مواردِ کاربرد را با کاربران نهایی مورد بررسی قرار دهید تا از صحّت آنها مطمئن شوید. استفاده از پیش‌الگوی واسط کاربر[8] در تست مواردِ کاربرد به همراه کاربران، بسیار مفید می‌باشد.

در حین توصیف مواردِ کاربرد و تشریحِ جزئیات مرتبط با آنها، ممکن است یکسری مواردِ کاربرد دیگر هم پیدا شده و اولویت‌بندی شوند. در ضمن باید همواره واژه‌نامه[9] را به روز رسانی نمود.

---

[1] - Inception
[2] - Vision
[3] - Use-Case
[4] - Elaboration
[5] - Construction Phase
[6] - Prototype
[7] - User Interface (UI)
[8] - UI Prototype
[9] - Glossary

در پروژه‌های کوچک، ممکن است یک نفر هم نقش تحلیل‌گر و هم نقش برنامه‌نویس را ایفا نماید. در این صورت، مستندسازی کاملِ مواردِ کاربرد، ضرورتی ندارد و همگام با پیاده‌سازی مواردِ کاربرد، تست و بررسی صحت آنها قابل انجام می‌باشد.

انتظار می‌رود که در پایان فازِ تشریح و برای دستیابی به هدف اولِ این فاز، در حدود ۸۰ درصد نیازمندی‌های سیستم شناخته شده و جزئیات مرتبط با آنها در قالب توصیف‌های مربوطه، مستند گردد. در حین پیاده‌سازی مواردِ کاربرد در طول فازِ ساخت[1]، هر یک از این مواردِ کاربرد در صورت لزوم، پالایش خواهند شد. ممکن است در فازِ ساخت هم، یکسری مواردِ کاربرد دیگر پیدا شود، ولی این موضوع عمدتاً به صورت یک استثناء می‌باشد.

هدف ۲. طراحی، پیاده‌سازی، اعتبار سنجی، و مبنا قرار دادن معماری

از آنجایی که هدفِ دوم فازِ تشریح، مرتبط با معماری، دستاوردها، و فعالیت‌های مرتبط با آن می‌باشد، نخست تعریف مختصری از معماری ارائه خواهیم داد. آر.یو.پی، معماری نرم‌افزار را به صورت مفهومی در برگیرنده‌ی تصمیم‌های کلیدی مرتبط با فراورده‌ی نرم‌افزاری تعریف می‌نماید. برخی از این تصمیم‌ها، عبارتند از:

- مهم‌ترین اجزاء سازنده‌ی[2] ساختار سیستم، واسط‌های[3] مربوط به این اجزاء و نیز تصمیم‌های مرتبط با خرید، ساخت، و یا استفاده‌ی مجدد[4] از این اجزاء ساختاری.
- توصیفی از چگونگی تعامل[5] میان اجزاء سازنده در زمانِ اجرا[6] (به منظور پیاده‌سازی مهم‌ترین سناریوهای شناسایی شده)
- پیاده‌سازی و تستِ یک پیش‌الگو[1] از معماری، به منظورِ اعتبارسنجی[2] کارکِردِ سیستم، اطمینان از حل‌شدن ریسک‌های فنی عمده[3]، دارا بودن شرایط کیفی مورد انتظار، کارایی[4]، مقیاس‌پذیری[5]، و هزینه[6].

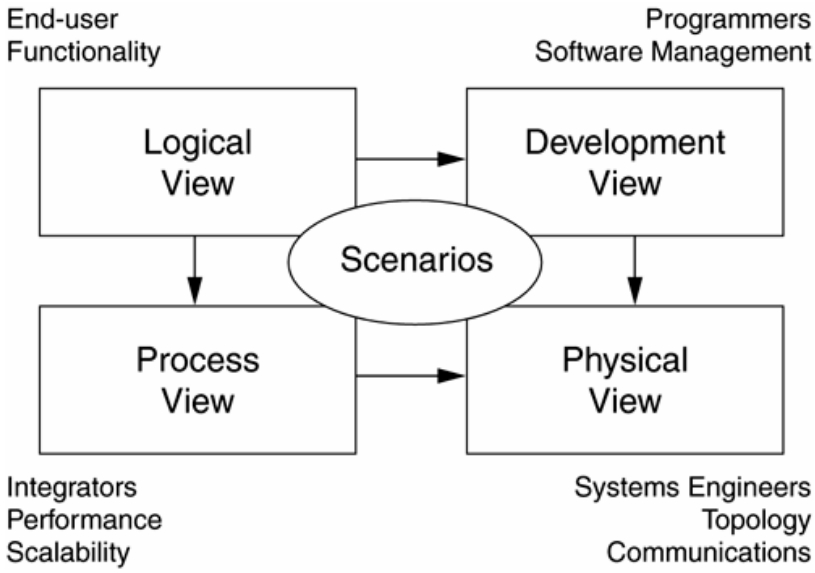
---

[1] - Construction
[2] - Building Blocks
[3] - Interface
[4] - Reuse
[5] - Interaction
[6] - Run-Time

در آر.یو.پی، مدلی تحت عنوانِ « معماری ۱+۴ » برای معماری ارائه شده است. این مدل که در شکل ۷-۲ نشان داده شده، جنبه‌های مختلفی از معماری را با توجه به انتظارات هر یک از ذینفعان پروژه، در قالب چند منظر[۷] بخش‌بندی می‌نماید. سندِ معماری نرم‌افزار که یکی از دستاوردهای مهم در آر.یو.پی و بیانگر برخی ملاحظات مهم از معماری می‌باشد، با توجه به همین مدل، سازماندهی شده است. در شکل ۷-۳، یک اَبَرمدل از معماری که بیانگر ارتباط میان مؤلفه‌ها و مفاهیم مختلف مرتبط با معماری می‌باشد، نشان داده شده است.

شکل ۷-۲

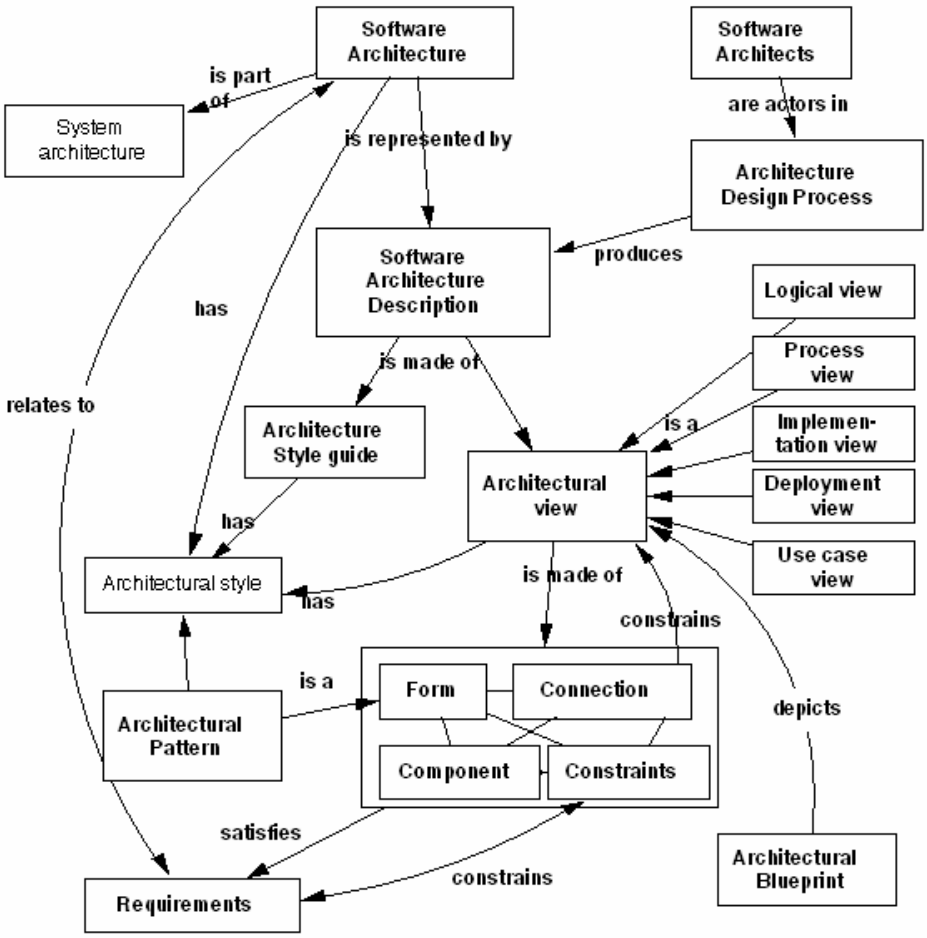
مدل ۱+۴ معماری: ارائه‌ی منظرهای مختلفی از معماری بر حسب انتظارات ذینفعان



[1] - Prototype
[2] - Validation
[3] - Major Technical Risks
[4] - Performance
[5] - Scalability
[6] - Cost
[7] - View

شکل ۷-۳

اَبَرمدلی از معماری



بنابراین، معماری بسیار فراتر از طرح و نقشه است و علاوه بر ملاحظات ساختاری، شامل مسائل مرتبط با رفتارِ زمانِ اجرای سیستم، کارایی، و حتی ملاحظات مرتبط با هزینه و زمان نیز می‌گردد. برای اینکه معماری بتواند به خوبی ملاحظات مذکور را در خود، نشان دهد، باید شکلِ نرم‌افزاری داشته باشد. در واقع بنا بر اصطلاح آر.یو.پی، معماری باید قابل اجرا[1] باشد.

[1] - Executable

با ایجاد یک معماری قابلِ اجرا که پیش‌الگویِ معماری‌گونه[1] نیز نامیده می‌شود، اِمکـان بررسـی و تأییـدِ پاسخ‌گو بودن به نیازهای شناسایی شده، فراهم می‌گردد. آیا پایه و زیربنای مناسـب و مـستحکمی بـرای بنـا کردن کلِ سیستم به وجود آمده است؟

مسلماً، بدون اطمینان از اِستحکام و پاسخ‌گو بودن معماری، هیچ‌گاه ایجاد یک ساختمانِ جدید مقرون به صرفه یا به عبارت دیگر، مهندسی نخواهد بود. می‌دانید که اگر معماری قابل اتکایی در اختیار نداشته باشید، یا ساختمان فرو خواهد ریخت، یا به موقع تحویل نشده، و یا هزینه‌ی ساخت آن از مقدار پیش‌بینی اولیـه فراتـر می‌رود. این فرو ریختن ممکن است در حین ساخت و یا بعد از آن اتفاق بیافتد. آثار و تبعاتِ ناشـی از داشـتن یک معماری نامناسب در ساخت یک ساختمان، واضح و روشن است و می‌توان آنرا لمس نمود. امّا، متأسـفانه چنین امری در رابطه با نرم‌افزار چندان محسوس نیست.

به همان اندازه که نگرانِ معماری یک ساختمان و یا یک پل هستیم، باید مراقب و نگرانِ معماری یـک فراورده‌ی نرم‌افزاری نیز باشیم. به طور قطع، چیزی که بتواند نگرانی‌های ما را نسبت به معماری یک سیستم نرم‌افزاری رفع نماید، نمی‌تواند صرفاً چند طرح و نقشه باشد؛ ما به چیزی نیاز داریم که صحّت، کارایی، مقرون به صرفه‌بودن، و ملاحظاتی از این دست را اثبات نماید. تجربه نشان داده است که تنها و مطمئن‌ترین راهکار موجود برای اثبات معماری یک سیستم، پیاده‌سازی پیش‌الگویِ معماری‌گونه از آن می‌باشد. حتی به واسطه‌ی اینکه مثلاً از فلان چارچوب استاندارد مانند J2EE یا Net. استفاده می‌نمایید، یا اینکه قبلاً موارد مـشابهی را تجربه نموده‌اید، نباید به هیچ وجه، اثباتِ کارایی معماری در پروژه‌ی جدید را فراموش نمایید.

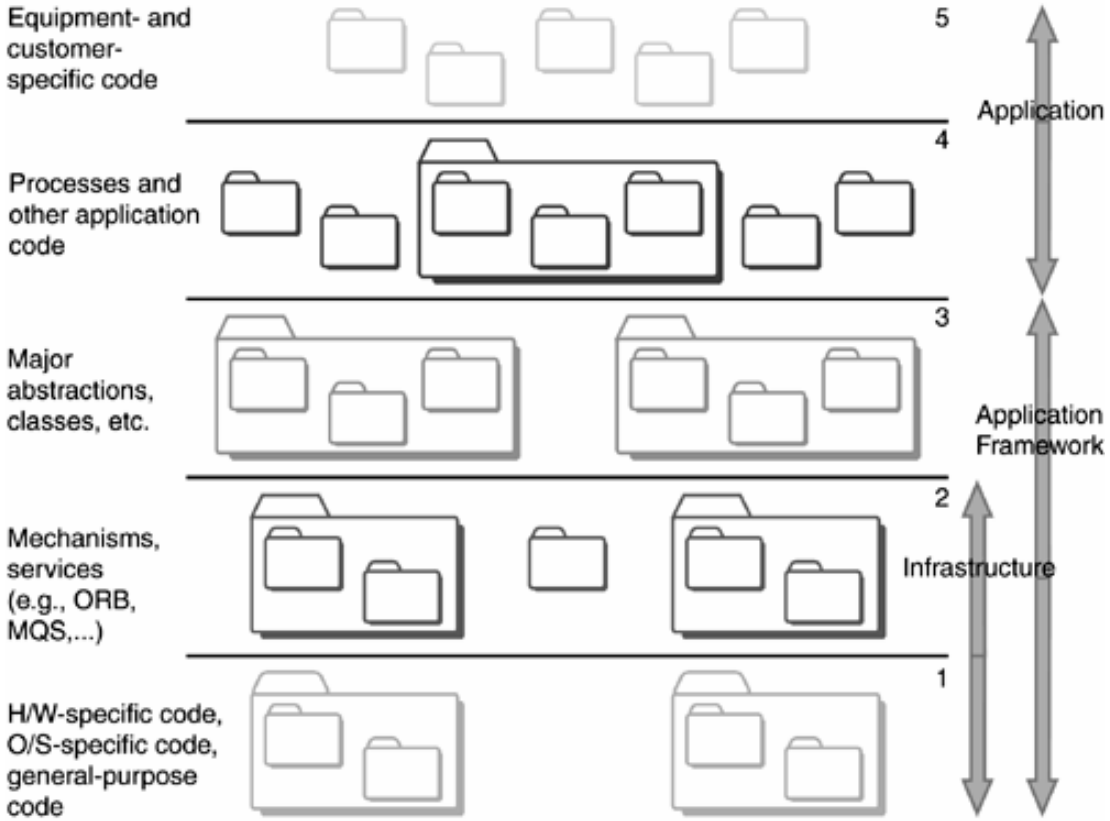
در یک معماری چندلایه‌ای[2] که در شکل ۷-۴ نشان داده شده است، عناصر لایه‌های پایینی یـا از قبـل وجود دارند و یا در همین فاز ایجاد می‌شوند؛ بخش‌هایی از لایه‌ی بالایی معماری، یعنی لایه‌ی کاربرد[3] نیز در همین فاز پیاده‌سازی می‌شود. این پیاده‌سازی محدود و بـا هـدفِ فـراهم نمـودنِ امکـان انجـام تـست‌هـای

[1] - Architectural Prototype
[2] - Layered Architecture
[3] - Application Layer

بارگذاری[1] و کارایی[2] انجام می‌شود. تست‌های مذکور به صورت اصطلاحاً انتها بـه انتهـا[3] انجـام مـی‌شـوند و بنابراین لازم است اسکلتی از عناصر و مؤلفه‌های لایه‌ی کاربرد نیز پیاده‌سازی شوند.

شکل ۷-۴

لایه‌های مختلف در یک معماری لایه‌بندی شده



[1] - Load
[2] - Performance
[3] - End-to-End

در پایان فازِ تشریح، معماری اصطلاحاً مبنا[1] قرار داده شده و تثبیت می‌گردد. مبنا قرار گرفتن معماری بدان معناست که شما می‌توانید از معماری‌تان به عنوان یک مرجعِ مستحکم[2] و قابلِ اعتماد برای بنا نمودنِ بقیه‌ی سیستم استفاده نمایید. از این نقطه به بعد، هر گونه تصحیح و تغییری در معماری، باید با احتیاط و تنها در صورتی‌که دارای دلیل منطقی باشد، انجام می‌شود. توجه داشته باشید که هر چه پروژه دارای پیچیدگی‌های فنی بیشتری باشد، تثبیت معماری مهم‌تر، حیاتی‌تر، و مستلزم هزینه‌ی بیشتری خواهد بود.

در واقع، با تثبیت معماری و رساندن آن به وضعیت مبنا قرار گرفتن، راه‌حلِ مسأله، اثبات می‌گردد. مادامی که به یک راه‌حلِ اثبات شده نرسیده‌ایم، به هیچ وجه واردِ فازِ بَعد، یعنی فازِ ساخت، نخواهیم شد.

شکل‌گیری معماری، قالبِ کلی و ماهیّتِ آن، با استفاده از برخی از مواردِ کاربرد[3] که به اصطلاح از نظر معماری قابل ملاحظه[4] می‌باشند، انجام می‌شود. در طی فاز شناخت (آغازین)، حدود بیست تا سی درصد از مواردِ کاربرد شناسایی شده است. این مواردِ کاربرد، نقش ارزنده‌ای در شکل‌گیری ماهیتِ سیستم و تعریفِ آن ایفا می‌نمایند. برخی از همین مواردِ کاربرد، در شکل‌گیری معماری نیز نقش دارند.

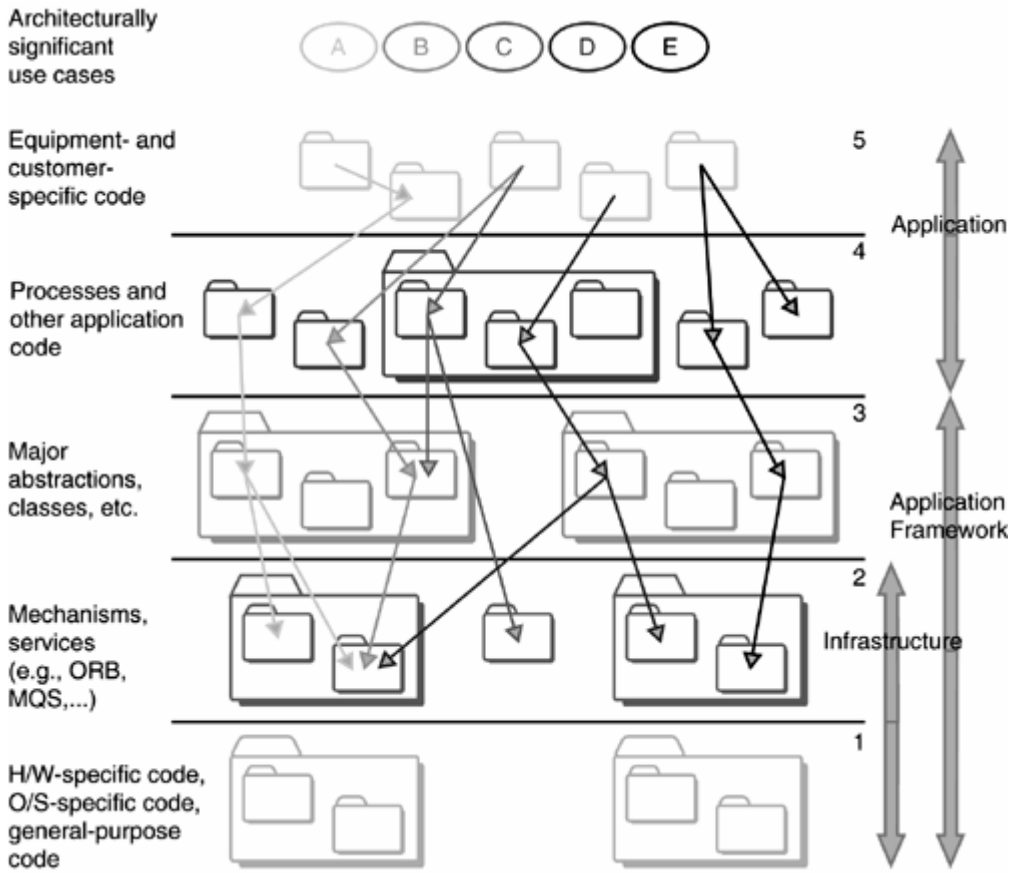
البته، توجه داشته باشید که علاوه بر مواردِ کاربردِ اشاره شده، باید برخی از عناصرِ خاص از نیازمندی‌ها که عمدتاً نیازمندی‌های غیر وظیفه‌مندی[5] می‌باشند، نیز شناسایی گردد. البته شناسایی مواردِ کاربرد مهم به لحاظ معماری و نیز نیازمندی‌های غیر وظیفه‌مندی، از مشکل‌ترین و در برخی موارد پیچیده‌ترین فعالیت‌های پروژه می‌باشد. همکاری میان نقش‌های مهندس فرایند[6]، معمار[7]، و مدیر پروژه[8]، تأثیر بسیار زیادی بر تثبیتِ موفق‌آمیز معماری دارد.

---

[1] - Baseline
[2] - Stable Reference
[3] - Use-Case
[4] - Significant
[5] - Non-functional Requirements
[6] - Process Engineer
[7] - Architect
[8] - Project Manager

شکل ۷-۵

مواردِ کاربردِ قابل ملاحظه از نظرِ معماری، شکل‌دهنده‌ی ماهیت معماری می‌باشند.



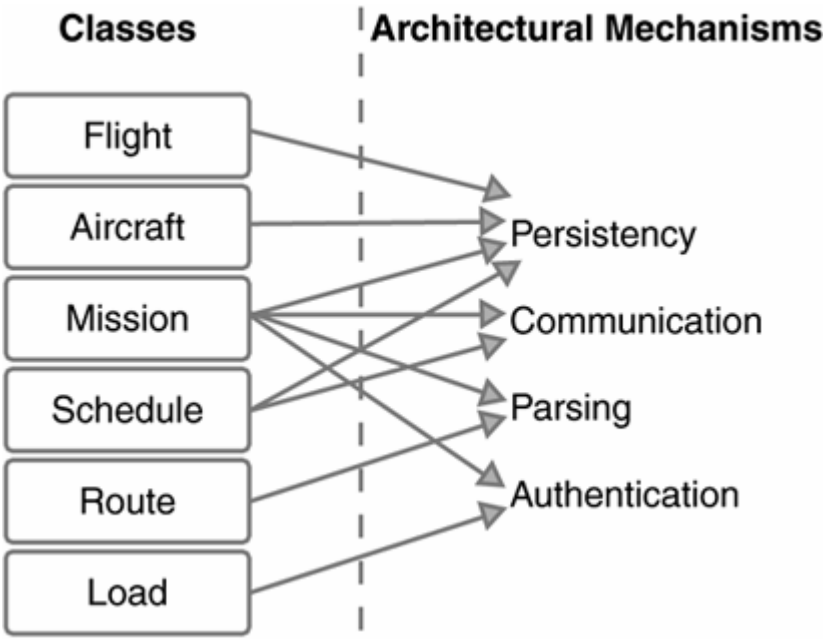
در نهایت، برای اطمینان از تثبیتِ معماری، مواردِ کاربرد مهم و قابل ملاحظه[1] را طراحی، پیاده‌سازی، و تست می‌نماییم. علاوه بر طراحی این مواردِ کاربرد، برای اطمینان از پوشش مناسبِ تمام ملاحظات معماری، ممکن است انتخاب و پیاده‌سازی جزئی چند موردِ کاربردِ غیر مهم نیز لازم باشد. طراحی بانک اطلاعاتی[2]، توصیف معماری در زمانِ اجرا (یعنی بررسی ملاحظاتی مانند همروندی[3]، پردازه‌ها[4]، تِرِدها[5]، و توزیع‌شدگی فیزیکی[6])، شناسایی مکانیزم‌های معماری[7]، پیاده‌سازی سناریوهای حیاتی، یکپارچه‌سازی[8] مؤلفه‌ها، و تستِ

[1] - Architecturally Significant Use-Cases
[2] - Database
[3] - Concurrency
[4] - Processes
[5] - Threads
[6] - Physical Distribution
[7] - Architectural Mechanisms
[8] - Integrate

سناریوهای حیاتی، از دیگر فعالیت‌های مهم در فاز تشریح می‌باشند. بدین ترتیب، بخش عمده‌ای از ریسک‌های قابل ملاحظه‌ی سیستم و پروژه، رفع خواهند شد.

شکل ۷-۶

مکانیزم‌های معماری راه‌حل‌هایی برای مسائل عمومی، فراهم می‌نمایند.



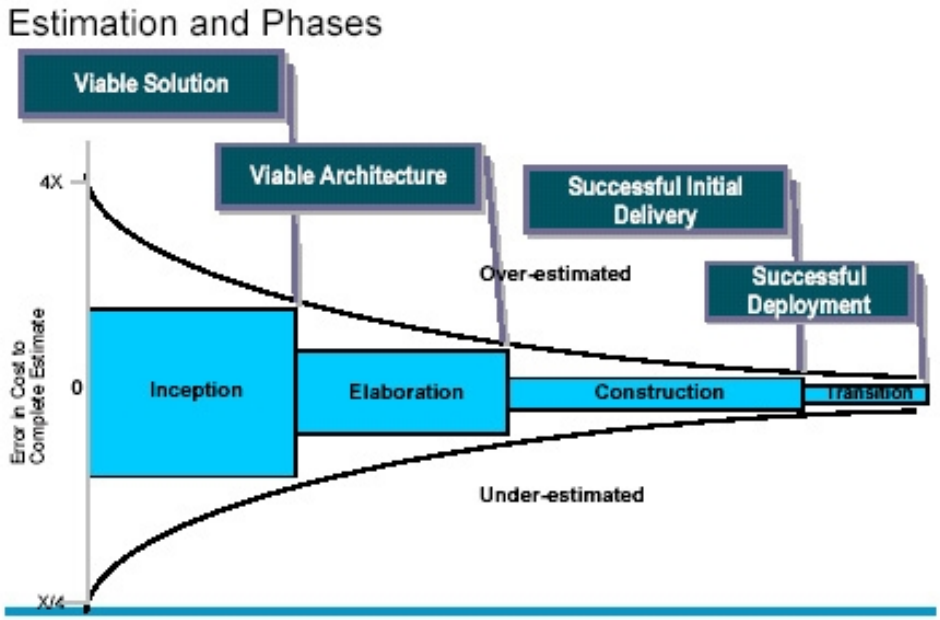
هدف ۳. تسکینِ ریسک‌های عمده و بهبودِ برنامه‌ی زمانی و تخمینِ هزینه‌ها

در طول فازِ تشریح (معماری)، عمدتاً با ریسک‌های فنی، ریسک‌های مرتبط با جلبِ توافق ذینفعان[1] نسبت به نیازمندی‌های[2] سیستم، و نیز ریسک‌های مرتبط با راه‌اندازی و آماده‌سازی بستر و محیط تولید، روبرو هستیم که باید مدیریت شوند. با نزدیک شدن به انتهای فازِ تشریح، اطلاعات دقیق‌تری در رابطه با پروژه بدست می‌آوریم. این اطلاعات می‌تواند ما را در دقیق‌تر نمودن تخمین‌های مربوط به زمان و هزینه که پیش از این در فاز آغازین و یا قبل از آن و در ابتدای پروژه ارائه شده بود، یاری می‌دهد.

[1] - Stakeholder
[2] - Requirements

شکل ۷-۷

روندِ دقیق‌تر شدن تخمین‌ها در طول فازهای چرخه‌ی تولید



با توجه به اینکه بخش عمده‌ای از نیازمندی‌های سیستم در این فاز شناسایی می‌شود، سند چشم‌اندازِ[1] پروژه نیز به‌روز رسانی می‌گردد. بدین ترتیب، این سند که تصویری از ماهیّت پروژه و ابعاد آن را ارائه می‌دهد، تا حد زیادی تثبیت می‌گردد.

با ایجاد یک چارچوبِ[2] ساختاری مستحکم، تحتِ عنوان معماری قابل اجرا[3]، قسمت عمده‌ای از مساله را حل نموده‌ایم. آنچه باقی مانده، بیشتر شبیه پر کردن چاله‌ها و بزرگ‌تر نمودن سیستم، می‌باشد. بدین ترتیب می‌توانیم با تقریب نسبتاً خوبی، میزان و حجمِ کارهای باقی‌مانده را تخمین بزنیم.

با انجام تکرارهای مختلف در این فاز، از میزان بهره‌وری و نحوه‌ی کار افرادِ تیم، ابزارها، و تکنولوژی، درک خوبی بدست خواهیم آورد. این اطلاعات، به بهتر کردن برنامه‌ی اجرایی پروژه و خصوصاً ملاحظات مرتبط با تخصیص منابع، کمک زیادی می‌نماید.

[1] - Vision
[2] - Framework
[3] - Executable

با تسکین بخش عمده‌ای از ریسک‌های فنی، امکان نزدیک‌تر کردن تخمین‌های بالا و پایینِ هزینه‌های زمانی و مالی فراهم می‌شود. در این فاز، نه تنها فرصت داریم کـه تخمـین‌هـای بهتـری ارائـه دهـیم، بلکـه می‌توانیم با کمکِ یک معماری مناسب، زمینه‌ی تعدیل در هزینه‌ها را نیز فراهم کنیم.

هدف ۴. بازبینی قالبِ فرایندِ تولید[1] و آماده‌سازی محیط

در طول فازِ آغازین (شناخت)، فرایندِ مناسبِ پروژه و نیز چگونگی بهره‌گیـری از آر.یـو.پـی را در سـندی تحتِ عنوان قالبِ فرآیندِ تولید، بیان نمودیم. در این سند که در حکمِ تعریف و الگوی اجرایی پروژه می‌باشـد، ابزارهای مورد نیاز، خروجی‌ها و دستاوردهای مطلوب، و ملاحظاتی از این دست، بیان شده است.

در فاز تشریح (معماری)، با انجام فعالیت‌های مختلف از همه‌ی دیسیپلین‌ها، درک مناسـبی از نقـش‌هـا[2]، دستاوردها[3]، فعالیت‌ها[4]، و نیز ابزارها[5] بدست می‌آید. بنابراین، اِمکان بهبود فرایند[6]، انجام تنظیم‌هـای بیـشتر و دقیق‌ترِ ابزارها و نیز آماده‌سازی کاملِ محیط تولید فراهم می‌شود.

بازبینی و بهبودِ فرایند تولید در همه‌ی فازها انجام می‌شود. اما فعالیت‌هایی کـه در ایـن فـاز (یعنـی فـاز تشریح) برای بهبود فرایند و استقرار کاملِ محیطِ تولید انجام می‌شود، تأثیر بسیار زیادی بر کارایی و بهره‌وری تیم و نیز ارتقای کیفیت فراورده، کم کردن زمان، و نیز صرفه‌جویی در هزینه‌ها دارد.

این بار نیز، همکاری میان مهندسِ فرایند[7] (خواه فردی از داخل سازمان، یا یک مشاور از بیرون سازمان) با مدیر پروژه و نیز معمارِ سیستم در دستیابی به این هدف، ضروری است. مهندس فرایند که مـسئولیت سـندِ قالبِ فرایند تولید را برعهده دارد، اطلاعات مربوط به ریسک‌ها، هزینه، و زمان را در تعامل بـا مـدیر پـروژه و

[1] - Development Case
[2] - Roles
[3] - Artifacts
[4] - Activities
[5] - Tools
[6] - Process Improvement
[7] - Process Engineer

اطلاعات فنی مهم را از معمارِ نرم‌افزار[1]، دریافت نموده و بر اساس آن‌ها، فرایندی بهتر و بهینه‌تر برای ادامه‌ی کار در فازهای ساخت و انتقال، ارائه می‌نماید.

بازبینی پروژه: نقطه‌ی تصمیم‌گیری درباره‌ی تثبیتِ معماری

در پایان فازِ تشریح، به یک نقطه‌ی تصمیم‌گیری کلیدی یا سازمانی می‌رسیم. در این نقطه‌ی تصمیم‌گیری که اِل.سی.اِی[2] نیز نامیده می‌شود، محدوده و اهداف تشریح شده‌ی سیستم، تثبیت معماری، و نیز غلبه بر ریسک‌های اصلی، مورد ارزیابی قرار می‌گیرد. در صورتی‌که یک پروژه در رسیدن به این نقطه ناکام باشد، یا باید یک تکرار دیگر به فاز تشریح اضافه شود، یا اینکه تجدید نظر اساسی در رابطه با خاتمه‌ی پروژه و یا ادامه‌ی آن، اتخاذ گردد.

معیارهای ارزیابی در نقطه‌ی تصمیم‌گیری در رابطه با تثبیتِ معماری، عبارتند از:

- آیا چشم‌انداز[3] پروژه و نیازمندی‌های[4] سیستم، به اندازه‌ی کافی مستحکم و دارای ثبات هستند؟
- آیا معماری دارای ثبات می‌باشد؟
- آیا رویکردها و روش‌های اصلی که باید در تست و ارزیابی مورد استفاده قرار گیرند، اثبات شده‌اند؟
- ایا تست و ارزیابی پیش‌الگویِ قابل اجرا[5] از معماری سیستم، نشان‌دهنده‌ی رفع و تسکینِ ریسک‌های عمده می‌باشد؟
- آیا برنامه‌های زمانبندی تکرارها برای فاز ساخت را می‌توان با تخمین‌های قابل قبولی ارائه کرد؟
- آیا تمامی ذینفعان با این موضوع موافقند که چشم‌اندازِ فعلی، بیان شده در سندِ چشم‌انداز، می‌تواند در چارچوب برنامه‌ی اجرایی و بر اساس معماری تثبیت‌شده، تحقق یابد؟
- آیا منابع اختصاص یافته، نسبت به آنچه برنامه‌ریزی شده بود، قابل قبول می‌باشد؟

---

[1] - Software Architect
[2] - LCA : Lifecycle Architecture
[3] - Vision
[4] - Requirements
[5] - Executable Prototype

در پروژه‌های بزرگ، این بازبینی ممکن است از یک روز تا حتی چند روز به طول بیانجامد. اما، در پروژه‌های کوچک، یک جلسه‌ی یک یا دو ساعته کافی خواهد بود.

چکیده‌ی فصل

در پایان فازِ تشریح[1] (معماری) که دومین فاز از چرخه‌ی تولید و آخرین فاز از مرحله‌ی مهندسی[2] در این چرخه، می‌باشد، عمده‌ی فعالیت‌های خود را به حل و فصلِ ریسک‌های فنی معطوف می‌نماییم. هسته‌ی اصلی سیستم، به صورت یک پیش‌الگوی قابلِ اجرا[3]، در این فاز شکل می‌گیرد؛ پایه‌ها و زیربنای سیستم، مستحکم شده و در طول فازهای بعد، سیستم همانند یک گلوله‌ی برف که از بالای کوه به سمت پایین سرازیر شده، به تدریج بزرگ و بزرگ‌تر می‌شود.

اهداف کلیدی مرتبط با این فاز، عبارتند از:

- حرکت از یک درکِ سطح‌بالا از نیازمندی‌های[4] مهمِ سیستم (حاصلِ فازِ آغازین) به توصیف و تشریح جزئیات نزدیک به هشتاد درصد از نیازمندی‌ها
- حرکت از یک معماری کاندید، مفهومی، و مُحتمِل به یک معماری قابل اجرا[5] (معماری که طراحی، پیاده‌سازی، و تست شده است) و مبنا قرار دادنِ آن برای ادامه‌ی کار
- تسکین ریسک‌های عمده‌ی معماری و ارائه‌ی تخمین‌های بهتری از زمان و هزینه
- بازبینی قالبِ فرایند تولید و آماده‌سازی محیط

[1] - Elaboration Phase
[2] - Engineering Stage
[3] - Executable Prototype
[4] - Requirements
[5] - Executable Architecture

پرسش‌هایی برای تحقیق و مطالعه‌ی بیشتر

۱. درباره‌ی تفاوت‌های فاز تشریح (معماری) در آر.یو.پی و فاز طراحی در رویکردِ منسوخِ آبشاری، تحقیق نمایید.

۲. در فاز معماری، عمدتاً چه نوع ریسک‌هایی مدیریت می‌شوند؟

۳. آیا ممکن است فاز معماری یک پروژه بسیار کوتاه باشد؟ در صورت مثبت بودن پاسخ، در چه مواردی؟

۴. با مطالعه‌ی سند معماری نرم‌افزار (Software Architecture Document)، چگونگی مستندسازی معماری ۱+۴ را بررسی نمایید.

۵. در رابطه با معماری خدمت‌گرا (Service-Oriented Architecture) و ملاحظات مرتبط با آن در آر.یو.پی، تحقیق نمایید.

۶. معماری چه نقشی در سازمان‌دهی تیم و انجام کار تیم ایفا می‌نماید؟

۷. در رابطه با چگونگی تستِ معماری در این فاز تحقیق نمایید.

۸. چگونه می‌توان مواردِ کاربردِ[1] مهم به لحاظِ معماری را شناسایی کرد؟

۹. در رابطه با مهارت‌های معمارِ نرم‌افزار تحقیق نمایید.

۱۰. تخمین بهترِ هزینه‌های زمانی و مالی، چگونه بر اساس معماری بدست می‌آید؟

[1] - Use-Case

منابع و مراجع

[1]. Ivar Jacobson, Grady Booch, James Rumbaugh, (1999). *The Unified Software Development Process*, Reading, MA: Addison-Wesley

[2]. Philippe Kruchten, (2003). *The Rational Unified Process: An Introduction*, Reading, MA: Addison-Wesley.

[3]. Per Kroll, Philippe Kruchten, (2003). *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*, Reading, MA: Addison-Wesley.

[4]. Rational Software Corporation. (2003) "Rational Unified Process 2003.06," Available at: http://www.rational.com/

[5]. Software Academy. (2006) "Unified Process Knowledge Base," Available at: http://www.unifiedProcess.info/

[6]. Scott W. Ambler, (2000). *The Unified Process Elaboration Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

فصل هشتم

فازِ ساخت

مهم‌ترین موضوعات مورد بررسی در این فصل، عبارتند از:

- ماهیّت فازِ ساخت
- آشنایی با اهداف فازِ ساخت
- بررسی برخی نکات مهم در رابطه با تکرارهای این فاز
- آشنایی با برخی از مفاهیم کلیدی مرتبط با فازِ ساخت

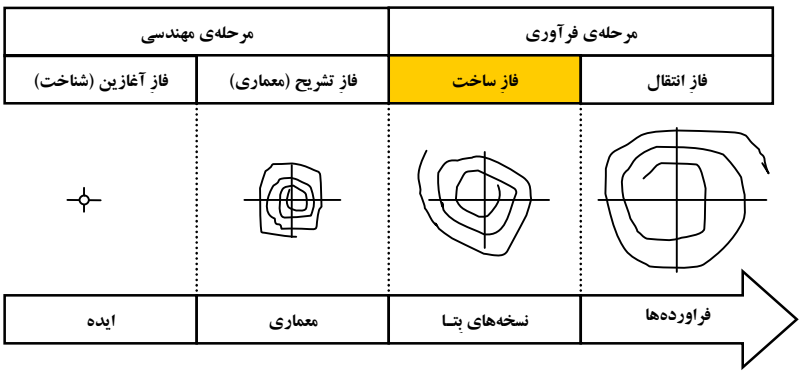
فازِ ساخت

۸

در این فصل، سومین فاز از چرخه‌ی تولید فراورده، یعنی فازِ ساخت را بررسی خواهیم نمود. فازِ ساخت، اولین فاز از مرحله‌ی فرآوری در فرآیند تولید می‌باشد. با پایان یافتن فازهای آغازین (شناخت) و تشریح (معماری)، مرحله‌ی مهندسی خاتمه یافته و اَکنون با با ورود به فازِ ساخت، وارد مرحله‌ی فرآوری خواهیم شد. این فاز، معمولاً طولانی‌ترین و در عین حال قابل انعطاف‌ترین فاز در فرایندِ تولید می‌باشد. مهم‌ترین ریسک‌های موجود در فازِ ساخت، ریسک‌های به اصطلاح لُجستیکی می‌باشند.

شکل ۸-۱

فازِ ساخت، اولین فاز از مرحله‌ی فرآوری در چرخه‌ی تولید



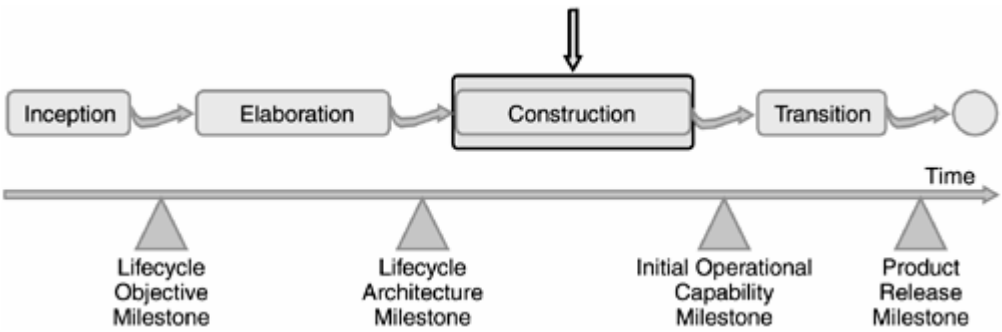
همانگونه که در شکل ۸-۲ نشان داده شده است، پایانِ این فاز، متناظر با یکی دیگر از گام‌های اصلی[1] یا نقاطِ تصمیم‌گیری سازمانی[2] در چرخه‌ی تولید می‌باشد. این گامِ اصلی تحتِ عنوان گامِ رسیدن به « قابلیتِ عملیاتی آغازین »[3] یا « بِتا[1] » ، جایی است که اصطلاحاً نسخه‌ی بِتای سیستم ارائه می‌گردد. نسخه‌ی بِتایِ

[1] - Major Milestone
[2] - Business Decision Point
[3] - Initial Operational Capability

یک فراورده‌ی نرم‌افزاری، عبارت است از یک سیستمِ نرم‌افزاری که در آن تمام قابلیت‌های توصیف شـده در نیازمندی‌ها و چشم‌انداز پروژه، پیاده‌سازی گشته و سیستم آماده‌ی استقرار در محیطِ مشتری و کاربرانِ نهایی می‌باشد. البتّه، ممکن است که هنوز اشکالات و نواقصی وجود داشته باشد که در طی تکرارهای فازِ بعد، یعنی فازِ اِنتقال، رفع خواهد شد.

شکل ۸-۲

فازِ ساخت، سومین فاز از چرخه‌ی تولید و نقطه‌ی تصمیم‌گیری ارائه‌ی اولین نسخه‌ی عملیاتی



در فصل قبل بیان شد که فاز تشریح (معماری) منجر به ارائه‌ی یک نسخه‌ی داخلـی[۲] از معمـاری قابـلِ اجرا[۳] و تثبیت شده[۴] می‌گردد. بر اساس این نسخه‌ی اجرایی از معماری سیستم، قادر خواهیم بود که بسیاری از ریسک‌های فنی عمده‌ی پروژه، مانند ریسک‌های مرتبط با درگیری و تداخل منابع[۵]، کارایی[۶]، و امنیت داده‌ها[۷] را بررسی نموده و راهکار مناسبی برای رفع و یا تسکین آن‌ها ارائه دهـیم. در طـولِ فـازِ سـاخت، بـه منظـور تکمیل قابلیت‌ها و کارکِردهای سیستم، تمرکز بیشتری بر طراحی تفـصیلی[۸]، پیـاده‌سـازی، و تـست خـواهیم داشت.

در فازِ ساخت، قسمت عمده‌ی از حجم کاری پروژه انجام می‌شود. بنـابراین، بهتـر اسـت حتمـاً بـا یـک معماری تثبیت شده پا به این فاز بگذاریم؛ در غیر این صورت، هزینه‌های زیادی را باید متحمّل شویم. توجـه

[1] - Beta
[2] - Internal Release
[3] - Executable
[4] - Baseline
[5] - Resource Contention Risks
[6] - Performance
[7] - Data Security
[8] - Detailed Design

داشته باشید که ممکن است براساس واقعیت‌هایی که یک معماری پیاده‌سازی شده و قابلِ اجرا نشان می‌دهد، تصمیم بر عدمِ ورود به فازِ ساخت و بستن پروژه گرفته شود. عدمِ تثبیت مناسبِ معماری سبب بروز دوباره‌کاری‌های زیادی در فازِ ساخت می‌شود.

به هر حال، درصورت تسکین یافتن ریسک‌های عمده‌ی فنی، وارد فازِ ساخت خواهیم شد. در طی فازهای قبل، بسیاری از مواردِکاربرد[۱] صِرفاً تشریح شده و پیاده‌سازی در مورد آنها انجام نشده‌است (در واقع لزومی به پیاده‌سازی آن‌ها نبوده). عمده‌ی موارد کاربردی که پیاده‌سازی شده‌اند، به منظور نشان دادن رفع یا تسکین ریسک‌های عمده بوده است. با توجه به تثبیتِ معماری در فازِ قبل، زیرسیستم‌ها[۲] به خوبی تعریف شده و واسط‌های[۳] مورد نیاز پیاده‌سازی شده است. امّا تا به اینجا، تنها زیرمجموعه‌ی کوچکی از کدهای سیستم پیاده‌سازی شده است. برای مثال، هنوز کدهای لازم برای اداره کردن منطقِ کارِ[۴] سیستم، شق‌های[۵] مختلفِ جریان‌های وقایع[۶]، و اداره‌کردن خطاهای[۷] ممکن، نوشته نشده‌است. با پیاده‌سازی سرویس‌ها و کارکردهای بیشتری از سیستم، نیازمندی‌ها شکل دقیق‌تر و منطقی‌تری به خود می‌گیرند. بنابراین در فازِ ساخت، فعالیت‌های عمده‌ای مانند دقیق‌تر کردن نیازمندی‌ها[۸]، طراحی تفصیلی، پیاده‌سازی، و تست انجام می‌شود.

در شکل ۸-۳، نحوه‌ی توزیعِ کار در فازهای مختلفِ آر.یو.پی، نشان داده شده است. همانگونه که در این شکل دیده می‌شود، بیشتر حجمِ کار در فازِ ساخت، مربوط به پیاده‌سازی، تست، و نیز طراحی تفصیلی و دقیق‌تر کردن نیازمندی‌ها می‌باشد. در اواخر این فاز که سیستم تقریباً در حال شکل‌گیری و کامل‌شدن می‌باشد، فعالیت‌های مرتبط با استقرار[۹] نیز حجم کاری زیادی خواهند داشت. فعالیت‌های مرتبط با دیسیپلینِ[۱۰]

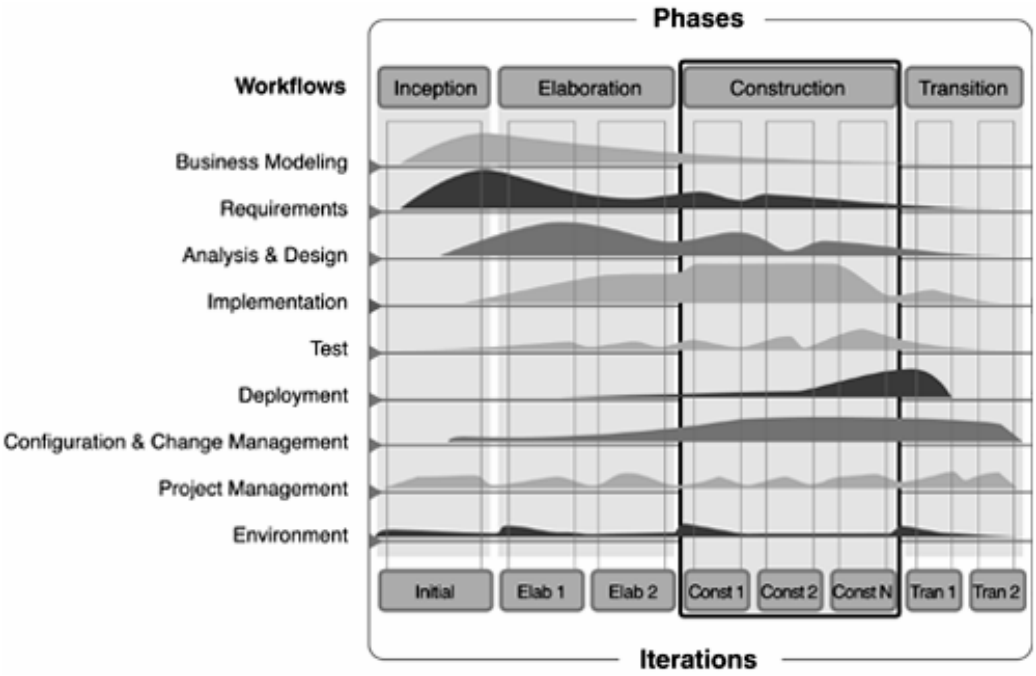
---

[1] - Use-Case
[2] - Subsystem
[3] - Interface
[4] - Business Logic
[5] - Alternatives
[6] - Flows of Events
[7] - Error Handling
[8] - Requirement Tuning
[9] - Deployment
[10] - Discipline

مدیریت پیکربندی و تغییرات[1]، بیشترین حجم کاری را نسبت به فازهای قبل داشته و تقریباً در تمام طول این فاز، بطور ثابت اجرا می‌شوند.

شکل ۸-۳

توزیع حجمی فعالیت‌ها در فازهای آر.یو.پی

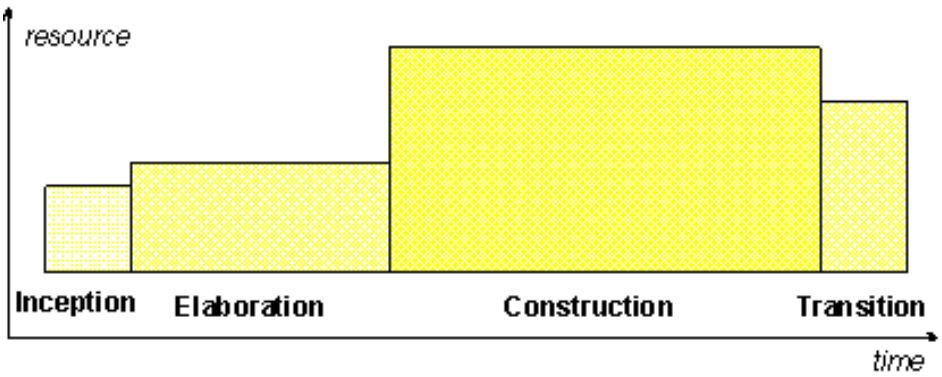


در حقیقت، معمولاً فازِ ساخت زمان زیادی از پروژه را به خود اختصاص می‌دهد. به طور متوسط در حدود ۶۵ درصد از کلِ حجم فعالیت‌ها و حدود ۵۰ درصد از زمان کلی پروژه به این فاز اختصاص دارد. توجه داشته باشید که این اعداد و ارقام در پروژه‌های مختلف متفاوت خواهد بود. ولی قدر مسلم اینکه به طور متوسط حجم زیادی از هزینه‌های پروژه در این فاز صرف می‌شود چه به لحاظ زمانی و چه به لحاظ مالی. این موضوع در شکل ۸-۴ و جدول ۸-۱ نشان داده شده است.

[1] - Configuration and Change Management

شکل ۸-۴

نسبت متوسط حجمِ فعالیت‌ها در فازهای مختلف



جدول ۸-۱

مقایسه‌ی متوسط زمان و حجمِ کار در فازهای مختلف

	آغازین (شناخت)	تشریح (معماری)	ساخت	اِنتقال
تلاش (حجمِ کار)	٪ ۵ ~	٪ ۲۰ ~	٪ ۶۵ ~	٪ ۱۰ ~
زمان (زمانبندی)	٪ ۱۰ ~	۳۰ ~	٪ ۵۰ ~	٪ ۱۰ ~

با وجودی‌که عمده‌ترین ریسک‌های فنی در فاز تشریح (معماری) رفع شده یا تسکین یافته‌اند، در طول فازِ ساخت نیز بطور مستمر با ریسک‌های مختلف روبرو خواهیم شد. بطور کلی، ریسک‌های فنی جدید باید تأثیر کمی بر کلیّت معماری[1] داشته باشند. در غیر این صورت، اگر ریسک‌های فنی جدید به نوعی معماری را با چالش‌های عمده مواجه نمایند، به احتمال زیاد، در فازِ قبل یعنی فازِ تشریح (معماری)، کار را به خوبی انجام نداده‌ایم.

در طول فازِ ساخت باید بر تولید کدهای با کیفیت مطلوب[2] و مقرون به‌صرفه[1] تمرکز داشته باشیم. می‌توانیم با استفاده از مکانیزم‌ها (سازوکارهای) معماری[2] تولیدِ کد را تسریع نماییم. خصوصاً در پروژه‌های

[1] - Overall Architecture
[2] - High-Quality

بزرگ، اطمینان از تمامیّت[۳] و یکپارچگی معماری، توسعه به صورت موازی[۴]، مدیریت پیکربندی و تغییرات[۵]، و خودکارسازیِ تست[۶]، جزء ضروریات پروژه برای دستیابی به موفقیت محسوب می‌شوند. معماری نرم‌افزار نقش بسیار مهم و تعیین کننده‌ای در دستیابی به اهداف مدنظر در فازِ ساخت ایفا می‌نماید. بنابراین، هرچه معماری دقیق‌تر و تثبیت‌شده‌تر باشد، احتمال موفقیت بیشتر می‌شود.

در این فاز، تمرکز زیادی بر برقراری توازن میان کیفیت، محدوده‌ی پروژه، زمان، و محتوا (سرویس‌های قابل ارائه) خواهیم داشت. بکارگیری ابزارهای یکپارچه‌ی مهندسی نرم‌افزار، نقش بسیار مهمّی در بهبود کارایی تیم ایفا می‌نماید. تصمیم بر منبع‌یابی بیرونی[۷] بر اساس معماری و توسعه‌ی به صورت موازی، از دیگر اقدامات ممکن در بهینه‌سازی تولید می‌باشد.

اهدافِ فازِ ساخت

اولین مسأله‌ای که قبل از بررسی اهداف فازِ ساخت باید بدان توجه داشته باشیم این است که در این فاز، مهم‌ترین دغدغه‌ی افرادِ تیم، در رابطه با تولیدِ مقرون به صرفه‌ی[۸] یک فراورده‌ی کامل[۹] می‌باشد. همان‌گونه که پیش از این نیز اشاره گردید، یک فراورده‌ی کامل عبارتست از یک نسخه‌ی عملیاتی[۱۰] از سیستمِ در حال تولید که می‌توان آن را در محیطِ کاربر مستقر نمود.

این دغدغه و نگرانی، بیانگر دو هدف اصلی فازِ ساخت می‌باشد. این اهداف عبارتند از:

۱. کمینه‌کردن هزینه‌های تولید و بهره‌گیری مناسب از اِمکانِ توسعه به صورت موازی[۱۱].

---

[1] - Cost-Effective
[2] - Architectural Mechanisms
[3] - Integrity
[4] - Parallel Development
[5] - Configuration and Change Management
[6] - Automated Testing
[7] - Outsourcing
[8] - Const Effective
[9] - Complete Product
[10] - Operational Version
[11] - Parallel Development

در این فاز سعی می‌شود که با هدف بهینه‌کردن منابع و پرهیز از دوباره‌کاری‌های[۱] غیر ضروری، هزینه‌های تولید، اَعَم از هزینه‌های زمانی و مالی به کمترین مقدار ممکن تقلیل یابد. در ضمن با کمک معماری مبتنی بر مؤلفه، اِمکان توسعه‌ی موازی مؤلفه‌های مستقل از هم به وجود می‌آید و این خود تأثیر بسزایی بر کاهش هزینه‌های تولید و کم کردن زمان تولید خواهد داشت.

۲. توسعه‌ی تدریجی[۲] و تکرارشونده‌ی[۳] یک فراورده‌ی کامل که آماده‌ی انتقال[۴] به محیطِ کاربر[۵] باشد.

اولین نسخه‌ی عملیاتی سیستم (تحت عنوان نسخه‌ی بِتا[۶]) به وسیله‌ی توصیفِ مواردِکاربرد[۷]، تشریح سایر نیازمندی‌های باقی‌مانده، تفصیل بیشتر جزئیات طراحی‌ها، تکمیل پیاده‌سازی، و تست نرم‌افزار ایجاد می‌شود. در ضمن، آماده بودنِ محل‌های راه‌اندازی و استقرار و نیز آمادگی کاربران برای کار با سیستم مورد بررسی قرار می‌گیرد.

فازِ ساخت و تکرارها[۸]

در این بخش، بحث تکرارها را در فازِ ساخت مورد بررسی قرار می‌دهیم. قبل از هر چیز باید خاطر نشان نماییم که یکی از مهم‌ترین ویژگی‌های پیاده‌سازی و برنامه‌نویسی (که حجم عمده‌ای از فعالیت‌های این فاز را تشکیل می‌دهد)، داشتن تکرارهای زیاد می‌باشد. در برخی از فرایندها مانند اِکس.پی[۹]، این موضوع مبنای رویکرد تکرار شونده در کلیه‌ی مراحل تولید مورد توجه می‌باشد.

---

[1] - Rework
[2] - Incremental
[3] - Iterative
[4] - Transition
[5] - User Community or User Environment
[6] - Beta Release
[7] - Use Cases
[8] - Iteration
[9] - XP

تعدادِ تکرارها در این فاز نیز همانند سایر فازها، به عوامل متعددی بستگی داشته و در پروژه‌های مختلف، متفاوت می‌باشد. بطور کلی، در بسیاری از موارد، تعداد تکرارهای این فاز بیشتر از فازهای دیگر می‌باشد. بطور معمول، دو یا سه تکرار در این فاز برنامه‌ریزی می‌شود.

فاز ساخت

سؤالی که مطرح می‌شود اینست که در هر یک از تکرارهای این فاز چه اتفاقاتی خواهد افتاد؟ مبنای برنامه‌ریزی این تکرارها بر چه اساسی است؟ در این فاز، تکرارها بر اساس مواردِ کاربردی که باید پیاده‌سازی شوند، برنامه‌ریزی می‌شوند.

در واقع آنچه مبنای تکرار و تکامل تدریجی واقع می‌شود، مواردِ کاربرد و سرویس‌های سیستم می‌باشد. مواردِ کاربرد و سرویس‌هایی که برای مشتری ضروری‌تر بوده و دارای اولویت بیشتری باشند و نیز مواردِ کاربردی که در رابطه با ریسک‌های فنیِ مهم‌تری هستند، زودتر پیاده‌سازی می‌شوند. در تکرار اول و نیز تاحدودی در تکرار دوم، برای شناسایی و غلبه‌ی سریع‌تر بر بیشترین ریسک‌های ممکن و داشتن یک پیاده‌سازی قبل از تشریح کامل‌تر مواردِ کاربرد، اقدام به پیاده‌سازی جزئی مواردِ کاربرد (یعنی پیاده‌سازی چند سناریوی محدود از یک موردِ کاربرد) خواهیم کرد. بعد از تعیین اینکه چه مواردِ کاربردی به طور کامل یا به صورت جزئی پیاده‌سازی شوند، باید مؤلفه‌های لازم برای ارائه‌ی سرویس‌های مربوط به این مواردِ کاربرد، شناسایی شوند. در مرحله‌ی بعد، طراحی تفصیلی این مؤلفه‌ها انجام شده و سپس در همان تکرار، پیاده‌سازی و تست می‌شوند. این شناسایی تأثیر بسزایی در درک بهتر هزینه و زمان لازم برای پیاده‌سازی مواردِ کاربرد خواهد داشت. ممکن است بر حسب منابع موجود لازم باشد که محدوده‌ی پروژه حداقل در یک تکرار خاص، دچار تغییر و تحوّل شود.

به عنوان یک نمونه‌ی موردی، اجازه دهید فرض کنیم که در فازِ ساخت از یک پروژه‌ی فرضی، ۱۵ موردِ کاربرد و سه تکرارِ برنامه‌ریزی شده داریم. با توجه به اهداف فاز ساخت، برنامه‌ریزی این فاز در پروژه‌ی مذکور چگونه خواهد بود؟ در جدول ۸-۲، نمونه‌ای از یک برنامه‌ی ممکن برای این پروژه نشان داده شده است. در این جدول، ورودی مطلوب به فازِ ساخت (که نتیجه‌ی انتهای فازِ معماری می‌باشد) و نیز نتیجه‌ی مطلوبِ حاصل از سه تکرار در انتهای فازِ ساخت، نشان داده شده است.

جدول ۸-۲

پیشرفت در ابتدا و در طول تکرارهای مختلف یک پروژه‌ی فرضی در فاز ساخت

فاز ساخت

نیازمندی‌ها	مؤلفه‌ها	تست‌ها
انتهای فازِ تشریح (معماری) و ابتدای فازِ ساخت		
– شناسایی ۱۵ موردِ کاربرد – تشریح جزئیات ۸ مورد از مواردِ کاربرد ، توصیف برخی جزئیات ۴ موردِ کاربرد و ارائه‌ی خلاصه‌ای از ۳ موردِ کاربرد دیگر	– شناسایی ۱۸ مؤلفه‌ی اصلی – پیاده‌سازی ۵۰ درصدی ۴ مورد از مؤلفه‌ها و نیز پیاده‌سازی تمام واسطها (interfaces) – ۱۰ مورد از مؤلفه‌ها دارای یک پیاده‌سازی منطقی (با حدود ۱۰ تا ۲۰ درصد از کدهای نهایی) و دارای واسطهای مورد نیاز – مؤلفه‌های موجود در لایه‌های پایینِ معماری، تقریباً به طور کامل پیاده‌سازی شده‌اند. – تمام کدهای پیاده‌سازی شده، تست شده‌اند.	– تست‌های اولیه‌ی کارایی (Performance) و بارگذاری (Load) روی معماری انجام شده است. این تست‌ها عمدتاً در رابطه با مواردِ کاربرد با اهمیت از نظر معماری (Architecturally Significant UCs) انجام شده است. – صحت و درستی وظیفه‌مندی‌های (Functionality) مربوط به ۴ مورد از مواردِ کاربردِ با اهمیت از نظر معماری، به خوبی تست شده است.
انتهای اولین تکرار در فاز ساخت		
– جزئیات مرتبط با ۱۲ مورد از مواردِ کاربرد تشریح شده و ۳ مورد دیگر توصیف بیشتری (بدون تشریح کامل جزئیات) ارائه می‌گردد.	– کماکان ۱۸ مؤلفه شناسایی شده (یکی از مؤلفه‌ها به دلیل حذف یک موردِ کاربرد، دیگر لازم نخواهد بود) – ۱۰ مورد از مؤلفه‌ها تقریباً بطور کامل پیاده‌سازی شده‌اند. – ۴ مورد از مؤلفه‌ها تا حدود ۵۰ درصد پیاده‌سازی شده‌اند. این مؤلفه‌ها دارای واسطهای کامل می‌باشند. – ۴ مؤلفه‌ی دیگر دارای واسط بوده و پیاده‌سازی منطقی (تقریباً ۱۰ تا ۲۰ درصر کدنویسی) دارند. – مؤلفه‌های لایه‌های پایین معماری تقریباً به طور کامل پیادسازی شده است. – تست واحدِ کدهای پیاده‌سازی شده	– برای اطمینان از مستحکم ماندن معماری و تطابق آن با شرایط و نیازمندی‌های تعیین شده، تست کارایی (Performance) و بارگذاری (Load) روی سیستم ادامه می‌یابد. – با تکمیل پیاده‌سازی مواردِ کاربرد، تست وظیفه‌مندی (Functional) آن‌ها انجام می‌شود.
انتهای دومین تکرار در فاز ساخت		
– به علت محدودیت زمانی، یکی از ۳ موردِکاربرد باقی مانده (که هنوز توصیف نشده‌اند) حذف می‌شود. – جزئیات بقیه‌ی ۱۴ موردِکاربرد تشریح می‌گردد.	– همانند تکرار قبل، ۱۸ مؤلفه شناسایی شده است (و یکی از این مؤلفه‌ها به علت حذف یک موردِکاربرد حذف خواهد شد) – پیاده‌سازی ۱۰ مؤلفه بطور کامل – پیاده‌سازی حدود ۵۰ درصدی ۸ مؤلفه‌ی دیگر که شامل تمام واسطهای لازم می‌باشد. – تست واحد (unit test) کدهایی که پیاده‌سازی شده‌اند	– برای اطمینان از مستحکم ماندن معماری و تطابق آن با شرایط و نیازمندی‌های تعیین شده، تست کارایی (Performance) و بارگذاری (Load) روی سیستم ادامه می‌یابد. – با تکمیل پیاده‌سازی مواردِ کاربرد، تست وظیفه‌مندی (Functional) آن‌ها انجام می‌شود.

ادامه‌ی جدول ۸-۲

پیشرفت در ابتدا و در طول تکرارهای مختلف یک پروژه‌ی فرضی در فاز ساخت

نیازمندی‌ها	مؤلفه‌ها	تست‌ها
انتهای سومین و آخرین تکرار در فاز ساخت		
– توصیف جزئیات ۱۴ موردِکاربرد باقی‌مانده	– شناسایی نهایی ۱۸ مورد مؤلفه – سیستم بطور کامل وظیفه‌مندی‌های پیش‌بینی‌شده و مورد توافق را انجام می‌دهد و بنابراین یک نسخه‌ی بِتا خواهیم داشت. – تمام ۱۸ مؤلفه‌ی معماری سیستم، به طور کامل پیاده‌سازی شده‌اند. البته رفع برخی نواقص احتمالی و انجام تنظیمات دقیق‌تر به فاز انتقال موکول می‌شود.	– برای اطمینان از مستحکم ماندن معماری و تطابق آن با شرایط و نیازمندی‌های تعیین شده، تست کارایی (Performance) و بارگذاری (Load) روی سیستم ادامه می‌یابد. – با تکمیل پیاده‌سازی موارد کاربرد، تست وظیفه‌مندی (Functional) آن‌ها انجام می‌شود.

پیش از بررسی اهدافِ کلیدی این فاز، لازم است به این نکته توجه داشته باشید که هیچ‌گونه ترتیبی میان این اهداف وجود نداشته و نیز این اهداف، معادلِ فعالیت‌های قابلِ انجام در طول فاز نمی‌باشند. در واقع، اهدافِ مورد بررسی، حال‌وهوا و مقصودِ تمامیِ فعالیت‌های هر یک از تکرارهای[1] مختلف در این فاز را بیان می‌نمایند؛ هر فعالیتی که انجام می‌شود، حتماً باید در راستای دستیابی به حداقل یکی از این اهداف باشد؛ در غیرِ این‌صورت، فعالیت زائدی محسوب می‌گردد.

هدف ۱. کمینه‌کردن هزینه‌های تولید و دستیابی به امکان توسعه‌ی موازی

در تمام فازهای چرخه‌ی تولید، در فکر تولید مقرون به‌صرفه‌ی[2] فراورده‌ی نرم‌افزاری هستیم. اگر تنها در فاز ساخت به این موضوع بیاندیشیم، مسلماً شکست خواهیم خورد و به این هدف دست پیدا نخواهیم کرد. اگر فاز تشریح (معماری) به درستی انجام شده باشد، نتیجه‌ی آن ایجاد یک معماری قابل اِجرا[3]، مستحکم[4]، و تثبیت شده[5] خواهد بود. بر اساس این معماری، قادر خواهیم بود یک راهکارِ مقرون به صرفه ارائه نماییم. این اِمکان از آنجایی حاصل می‌گردد که با داشتن چنین معماری، امکان استفاده‌ی مجدد[6] از مکانیزم‌های معماری[7] به

[1] - Iteration
[2] - Cost-effective
[3] - Executable Architecture
[4] - Robust
[5] - Baseline
[6] - Reuse
[7] - Architectural Mechanism

وجود آمده و نیز قادر خواهیم بود با توجه به معماری، فعالیت‌های موازی را تعریف نماییم. در ضمن، با توجه به اینکه بسیاری از ریسک‌های عمده‌ی فنی به وسیله‌ی معماری رفع شده‌اند، بـه احتمـال زیـاد بـا مـشکلات و چالش‌های کمتری روبرو خواهیم شد.

در ادامه، برخی ملاحظات مرتبط با فازِ ساخت را که تیم‌ها و پروژه‌های بـزرگ بـرای موفقیـت در تولیـد نرم‌افزار باید آنها را مدنظر قرار دهند، ذکر خواهیم نمود.

سازماندهی حول معماری

یکی از مهم‌ترین مزایای یک معماری تثبیت شده و مستحکم اینست که این معماری مسئولیت‌های[۱] سیستم را در میان یک‌سری زیرسیستمِ[۲] به‌خوبی‌تعریف‌شده[۳]، تقسیم می‌نماید. در این صورت، یک معمار یا یـک تـیم معماری[۴] خواهیم داشت که نگران معماری و چگونگی حفظِ یکپارچگی و مجتمع‌سازی آن بوده و بقیه‌ی افراد بر زیرسیستم یا زیرسیستم‌هایی که به آنها اختصاص یافته است، تمرکز خواهند داشت.

توجه داشته باشید که با وجود شکسته‌شدن مسئولیت‌های سیستم در قالب زیرسیستم‌ها، کلیه‌ی افـراد در تیم‌های مختلف (اَعَم از تحلیل‌گران، طراحان، برنامه‌نویسان، کارشناسان تست، مدیران پروژه و ...)، باید درک مناسبی از سرتاسرِ سیستم[۵] داشته باشند، اما تمرکز اصلی‌شـان بـر زیرمجموعـه‌ی مشخـصی از سیـستم کـه متشکّل از یک یا چند زیرسیستم است، خواهد بود.

هنگامی که سازماندهی فعالیت‌ها حُولِ معماری صورت پذیرد، فعالیت‌های هیچ یک از افراد تـیم بـا هـم تداخل نخواهد داشت. سازماندهی حُولِ معماری به برقراری ارتباطات مناسب‌تر نیز کمـک مـی‌کنـد. معمـولاً ارتباطاتِ رو در رو[۶] مؤثرترین شکل ارتباطات می‌باشد، اما با بزرگ‌تر شدن پروژه و یا توزیع جغرافیایی آن، بـه ناچار اِمکان برقراری ارتباطات رو در رو کمتر می‌شود.

---

[1] - Responsibility
[2] - Subsystem
[3] - Well-defined
[4] - Architecture Team
[5] - Overall System
[6] - Face-to-face Communication

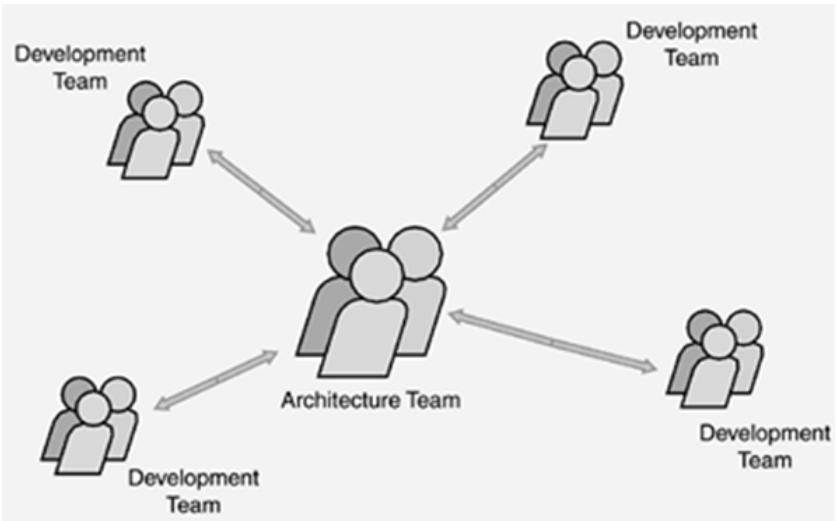
توجه داشته باشید که تعدادِ راه‌های ممکن برای برقرای ارتباطات میان اعضای یک تیم، با افزایش تعدادِ اعضاء به صورت یک تصاعد هندسی زیاد می‌شود. برای یک تیم با اندازه‌ی N عضو، تعداد مسیرهای ارتباطی برابر خواهد بود با N*(N-1)/2. این بدان معناست که در یک تیم دو نفره، یک مسیر ارتباطی، در یک تیم سه نفره، سه مسیر ارتباطی، و در یک تیم شش نفره ۱۵ مسیر ارتباطی وجود دارد.

فاز ساخت

افزایش تعداد مسیرهای ارتباطی، کارایی تیم را به شدت کاهش می‌دهد و بنابراین لازم است به جای اینکه تک تک افراد با هم در ارتباط باشند، روش ارتباطی مناسب‌تری پیدا شود. یکی از موفق‌ترین راهکارهای موجود اینست که یک تیم خاص را مسئول معماری نموده و چندین تیم کوچک هر کدام مسئولیت یک یا چند زیرسیستم را بر عهده گیرند. ارتباطات میان این تیم‌ها از طریق تیم معماری برقرار می‌شود و همین تیم مسئول تطابق معماری هر یک از زیر سیستم‌ها با معماری جامع می‌باشد. این موضوع، در شکل ۸–۵ نشان داده شده است.

شکل ۸–۵

سازماندهی تیم‌ها حُولِ معماری



با این تدبیر، ارتباطات بسیار ساده‌تر شده و حتی در پروژه‌های بسیار بزرگ نیز می‌توان میان تیم‌های مختلف ارتباطات مؤثری برقرار نمود. توجه داشته باشید که بر اساس همین دلایل، در یک پروژه‌ی بزرگ، وجود یک تیم مسئولِ یکپارچه‌سازی[۱] نیز ضروری خواهد بود.

مدیریت پیکربندی[۲]

سیستم مدیریت پیکربندی[۳] عموماً در طولِ فاز آغازین[۴] (شناخت) نصب و راه‌اندازی شده و در فاز تشریح[۵] (معماری) همزمان با تثبیتِ معماری سیستم، بازبینی و تصحیح می‌شود. در این بخش چرایی[۶] و مزایای داشتن یک سیستم مدیریت پیکربندی را بررسی می‌نماییم.

با پیشرفت پروژه، به علّتِ افزایش میزان تغییرات اِعمالی در بسیاری از دستاوردها[۷]، پیگیری نسخه‌های مختلف از این دستاوردها، به شکل روزافزونی مشکل‌تر و پیچیده‌تر خواهد شد. خصوصاً با داشتن رویکرد تکرارشونده، مستمراً نسخه‌های جدیدی از نرم‌افزار به وجود می‌آید و بنابراین لازم است که ابزار مناسبی برای موفقیت در هر یک از موارد و شرایط زیر، در اختیار داشته باشیم:

- توسعه‌ی تکرارشونده به معنای ایجادِ نسخه‌های متعدد[۸] از نرم‌افزار می‌باشد. این نسخه‌ها حتی ممکن است به صورت روزانه ایجاد شوند. در این حالت باید مشخص نماییم که چه نسخه‌ای از مؤلفه‌های مختلف در هر ساخت‌وسازِ میانی[۹] استفاده خواهد شد. در برخی از موارد، نسخه‌ی نهایی از یک مؤلفه‌ی خاص و در مواردی نیز نسخه‌های قبلی آن که درستی عملکردشان اثبات شده یا به علت کامل نشدن نسخه‌ی نهایی، در ایجادِ نسخه‌ی جدید استفاده می‌شود.

---

[1] - Integration Team
[2] - Configuration Management
[3] - Configuration Management System
[4] - Inception
[5] - Elaboration
[6] - Why
[7] - Artifact
[8] - Frequent Builds
[9] - Build

- در توسعه‌ی تکرارشونده، معمولاً راهکارهای مختلفی را برای دستیابی به نتایج مطلوب آزمایش می‌نماییم. در این صورت لازم است محیط مناسبی برای مدیریت راهکارهای مختلف در طول مسیر وجود داشته باشد.
- با بزرگ‌تر شدن حجم و ازدیادِ فعالیت‌های پروژه، برای جلوگیری از تداخل و ایجاد اثرات نامطلوب، ضروری خواهد بود که تغییرات ایجاد شده توسط یک تیم از دیدِ سایر تیم‌ها پنهان بماند. علاوه بر این اعضای تیم‌ها نیز علاقمند به کنترل محیط کارشان می‌باشند.

فاز ساخت

- در پروژه‌های بزرگ، باید افرادی را که اجازه‌ی اِعمال یک‌سری تغییرات خاص در پروژه را دارند، کنترل نماییم.
- هنگامی که با یک خطا و اشتباه برخورد می‌نماییم، علاقمند هستیم که به عقب برگشته و خطا را ریشه‌یابی نماییم. در ضمن بسیار ضروری است که بتوانیم به سرعت به نسخه‌های قبلی که تثبیت شده‌اند، دسترسی پیدا کرده و در صورت لزوم آنها را جایگزینِ سیستم مشکل‌دارِ فعلی نماییم.

راه‌حل مناسب، استفاده از یک سیستم مدیریت پیکربندی می‌باشد. بعد از اینکه چنین سیستمی راه‌اندازی گردید، اعضای تیم نگران هزاران فایل و دستاورد تهیه شده، نسخه‌های مختلف و نیز وابستگی و ارتباط میان آنها، نخواهند بود و بدین ترتیب قادر خواهند بود وقت بیشتری را به فعالیت‌های اصلی پروژه اختصاص دهند.

برنامه‌ریزی یکپارچه‌سازی[1]

در رویکردِ تکرارشونده، یکپارچه‌سازی و تستِ یکپارچه‌سازی، بسیار پیچیده‌تر می‌باشد. در هر تکرار، باید یک برنامه‌ی یکپارچه‌سازی[2] داشته باشیم که بیانگر قابلیت‌هایی قابلِ تست در هر نسخه‌ی میانی[3] به علاوه‌ی مؤلفه‌هایی که برای ارائه‌ی این قابلیت‌ها باید با هم تلفیق شوند، می‌باشد. قابلیت‌های مورد نظر ممکن است

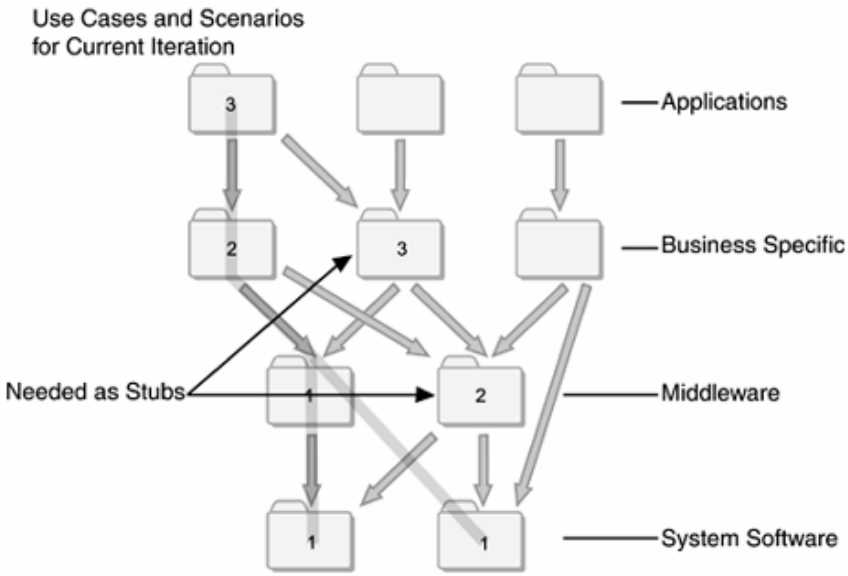
[1] - Integration Planning
[2] - Integration Build Plan
[3] - Build

یک یا چند موردِکاربرد، قسمتی از مواردِکاربرد و یا هر وظیفه‌مندی[1] قابل تست دیگری باشد. تست‌ها نیز ممکن است دربرگیرنده‌ی تست‌های وظیفه‌مندی[2]، بارگذاری[3]، فشار[4]، و دیگر تست‌های متداول باشد.

در بسیاری از موارد، یک نسخه‌ی میانی خود از تلفیق چند نسخه‌ی میانی کوچک‌تر بدست می‌آید. شکل ۸-۶ نشان‌دهنده‌ی ایجادِ سه نسخه‌ی میانی می‌باشد. همان‌گونه که در این شکل نشان داده شده است، ایجادِ نسخه‌های میانی، معمولاً در لایه‌های معماری به صورت از پایین به بالا انجام می‌شود. در این شکل، ابتدا ایجادِ نسخه‌ی میانی ۱ انجام می‌شود و سپس این نسخه، تحت تست قرار می‌گیرد. در ادامه، نسخه‌ی میانی ۲ نیز به نسخه‌ی میانی ۱ که تست شده است اضافه شده و تست مجدداً انجام می‌شود. در نهایت ایجادِ نسخه‌ی میانی ۳ و تستِ آن انجام خواهد شد.

شکل ۸-۶

نمونه‌ای از روندِ ایجادِ نسخه‌های میانی (Builds)



[1] - Functionality
[2] - Functional
[3] - Load
[4] - Stress

اجبار در بکارگیری و پایبندی به معماری

برای اینکه به طور کامل از مزایای معماری بهره‌مند شویم، لازم است که به طـور فعّـال، از پایبنـدی بـه آن اطمینان حاصل نماییم. در یک تیم کوچک، این کار با داشتن یک سری جلساتِ بحث در رابطـه بـا طراحـی معماری، امکان‌پذیر است. اما در تیم‌های بزرگ‌تر، دقت بیشتری لازم می‌باشد.

اطمینان از بکارگیری مکانیزم‌های معماری که در قالبِ معماری سیستم، تعریف شده‌اند، پرهیز از اِبداعِ این مکانیزم‌ها توسط افراد تیم، و نیز اِطمینان از عدمِ تغییر دلخواهانه‌ی[1] واسط‌ها، از دیگر نکات مهم در رابطه با معماری می‌باشد. در این رابطه ممکن است که لازم باشد همه‌ی افراد تیم، آموزش‌های لازم را درباره‌ی معماری و مکانیزم‌های موجود در آن ببینند.

فاز ساخت

اطمینان از پیشرفت مستمر[2]

برای اطمینان از پیشرفت مستمر پروژه، باید ضمن قرار دادن اهداف میانی و کوتاه مدت در طول پروژه، به طور مستمر، دستیابی به آن اهداف را نیز اثبات نماییم. برای اطمینان از موفقیت باید به نکات ذیل توجه داشته باشید:

- تنها یک تیم و با یک ماموریتِ[3] واحد ایجاد نمایید. باید از ایجاد تیم‌های به اصطلاح تخصصی جـدا مانند تیم تحلیل‌گران و یا تیم طراحان پرهیز شود. برای مثال داشتن یک تیم جداگانه‌ی تحلیل کـه پس از تکمیل نیازمندی‌های سیستم، دستاوردهای مربوط را در اختیار تیمـی متـشکل از طراحـان و برنامه‌نویسان برای طراحی و پیاده‌سازی قرار می‌دهند و آنها نیز دستاوردهای خود را در اختیـار تـیمِ تست قرار می‌دهند، اصلاً توصیه نمی‌گردد. در عوض، مناسب‌ترین و موفق‌ترین تیم‌هـا، ترکیبـی از تخصص‌های مختلف[4] را در خود دارند؛ تیم‌هایی که در آنهـا هـر یـک از اعـضای تـیم، خـود را در دستیابی به اهداف و پیشرفت کارِ تیم مسئول می‌داند.

---

[1] - Arbitrarily
[2] - Continual Progress
[3] - Mission
[4] - Cross-functional Teams

- تعیین اهداف شفاف و قابل دستیابی برای افراد. هر یک از اعضای تیم باید تصویر روشنی از آنچه که در یک تکرار یا بخشی از آن، انجام خواهد شود و آنچه که حاصل می‌گردد، داشته باشند. همه‌ی اعضای تیم باید درباره‌ی قابل دستیابی بودن نتایج با هم توافق داشته باشند.
- تست و ارائه‌ی مستمر کدها و نسخه‌های اجرایی از نرم‌افزار. مهم‌ترین معیار برای سنجش میزان پیشرفت، نسخه‌ی اجرایی نرم‌افزار می‌باشد.
- اجبار در رابطه با یکپارچه‌سازی مستمر. در صورت امکان، نسخه‌های میانی را به صورت روزانه ایجاد نمایید. البته در پروژه‌های بزرگ و نیز در پروژه‌هایی که ابزارهای مناسب مدیریت پیکربندی وجود ندارد، امکان ایجاد نسخه‌های میانی به صورت روزانه، وجود نخواهد داشت. نتیجه‌ی مستقیمِ ایجاد نسخه‌هایی میانی متعدّد، انجام یکپارچه‌سازی و تست مستمرِ یکپارچگی در طول پروژه می‌باشد.

هدف ۲. توسعه‌ی تکرارشونده‌ی یک فراورده‌ی کامل به گونه‌ای که آماده‌ی انتقال به محیط کاربر باشد.

توصیف مواردِکاربرد[1] و سایر نیازمندی‌های باقی‌مانده

اغلب در حین پیاده‌سازی و تستِ یک موردِکاربرد، لازم است که حداقل برخی از نیازمندی‌های تفصیل شده، مورد بازبینی مجدد قرار گیرند و حتی در مواردِ بسیاری، ممکن است که لازم باشد درباره‌ی تمام مواردِکاربرد، تجدیدِ نظر نموده و به یک راهکار بهتر بیاندیشیم.

در فازِ تشریح[2] (معماری)، مواردِکاربردِ غیر ضروری و نیز آن‌هایی که به لحاظ معماری اهمیتی نداشتند، مورد توجه قرار نگرفته و به فازِ ساخت[3] موکول شده‌اند. برخی از مواردِکاربرد نیز در فازِ تشریح (معماری) به طور مختصر توصیف شدند و اکنون در فازِ ساخت به طور کامل تشریح خواهند شد.

بسیاری از نیازمندی‌های غیروظیفه‌مندی[1]، مانند نیازمندی‌های مرتبط با کارایی، برای دستیابی به یک معماری مستحکم و تثبیت شده که مبنای توسعه قرار گیرد، ضروری است. بنابراین بیشترِ این دسته از

[1] - Use-Case
[2] - Elaboration Phase
[3] - Construction Phase

نیازمندی‌ها در فازِ تشریح (معماری)، تجزیه و تحلیل و در مواردی پیاده‌سازی شده‌اند. اما ممکن است در فـازِ ساخت نیاز باشد برای شناخت بیشتر، برخی جزئیات تکمیلی به آن‌ها اضافه شود.

تکمیل طراحی‌ها

در فازِ تشریح (معماری)، زیر سیستم‌ها و واسطهای مربوط به آن‌ها، مؤلفه‌های کلیدی و واسطهایـشان و نیـز مکانیزم‌های معماری تعریف شد. اگر از معماری لایه‌بنـدی شـده اسـتفاده شـود، در فـاز تـشریح (معمـاری)، لایه‌های پایین (لایه‌های زیرساختی) و نیز مواردِکاربردِ مهم از نظر معماری، پیاده‌سازی می‌شوند. **فاز ساخت**

در هر یک از تکرارهای فازِ ساخت، به تکمیلِ طراحی مجموعـه‌ای از مؤلفـه‌هـا و زیرسیـستم‌هـا و یـک مجموعه از مواردِکاربرد خواهیم پرداخت. با پیاده‌سازی مؤلفه‌ها، ممکن است ایجاد یک‌سری مؤلفه‌های جدیـد نیز ضرورت یابد. در تکرارهای اولِ فازِ ساخت، بر مواردی که ریسک‌های بیشتری دارند، مانند مسائل مرتبط با واسطها[۲]، کارایی[۳]، و قابلیتِ استفاده[۴] تاکید خواهیم داشت. برای دستیابی به این هدف، باید سناریوهای کلیدی را از برخی مواردِکاربردِ انتخاب شده، طراحی، پیاده‌سازی، و تست نماییم. در تکرارهای آخر، تأکید بیشتری بـر تکمیل کارها خواهیم داشت؛ به گونـه‌ای کـه همـه‌ی مواردِکـاربرد انتخـاب شـده، بـه طـور کامـل طراحـی، پیاده‌سازی، و تست شود. در فازِ تشریح (معماری)، اولین نسخه از پیاده‌سازی بانکِ اطلاعاتی[۵] ایجاد می‌گردد. در فازِ ساخت، بانک اطلاعاتی بطور کامل ایجاد می‌شود.

سایر اقدامات

پیاده‌سازی و تستِ واحدِ[۶] کدها، انجام یکپارچه‌سازی و تست سیستم، انجام استقرارهای[۷] اولیه و دریافت بازخوردهای[۸] حاصل، آمادگی برای استقرار نسخه‌ی بِتـا، و آمادگی برای استقرار نهایی، از دیگر اقدامات و

---

[1] - Non-functional Requirements
[2] - Interfaces
[3] - Performance
[4] - Usability
[5] - Database
[6] - Unit Test
[7] - Deployment
[8] - Feedback

نکات مرتبط با فازِ ساخت می‌باشد. جزئیات هر یک از این اقدامات در آر.یو.پی به طور مفصّل تشریح شده است.

پایان فازِ ساخت: گامِ اصلی یا نقطه‌ی تصمیم‌گیری ارائه‌ی اولین قابلیت‌های عملیاتی[1]

فاز ساخت با یکی از مهم‌ترین گام‌های اصلی در پروژه خاتمه می‌یابد. این گام اصلی، به اصطلاح، گامِ ارائه‌ی اولین قابلیت‌های عملیاتی یا گامِ بِتـا[2] نامیده می‌شود. در این گام، زمان تصمیم‌گیری درباره‌ی این موضوع می‌باشد که آیا فراورده‌ی بدست آمده آماده‌ی استقرار و تست در محیط کاربران می‌باشد یا نه. برای اتخاذ این تصمیم و خاتمه‌ی فازِ ساخت، باید پرسش‌های زیر را پاسخ داد:

- آیا محصول بدست آمده به اندازه‌ی کافی پایدار[3] و کامل[4] هست که بتوان آن‌را در محیط کاربر مستقر نمود؟
- آیا همه‌ی ذینفعان آماده‌ی انتقالِ سیستم به محیط کاربر هستند؟
- آیا برآمدِ هزینه‌ی منابع هنوز نسبت به برآورد برنامه‌ریزی شده، کماکان قابل قبول می‌باشد؟

در صورتی که نتایج و شرایط مورد انتظار در گام بِتـا بدست نیامده باشد، رفتن به فازِ اِنتقال به تعویق افتاده، یک تکرار دیگر نیز در انتهای فازِ ساخت برنامه‌ریزی خواهد شد. توجه داشته باشید که معیارها و پرسش‌های اشاره‌شده در این نقطه‌ی تصمیم‌گیری، همانند سایر نقاط تصمیم‌گیری در انتهای فازهای مختلف، به صورت جلسات بازبینی و با حضور ذینفعان کلیدی بررسی می‌شود.

---

[1] - Initial Operational Capability Milestone
[2] - Beta Milestone
[3] - Stable
[4] - Mature

چکیده‌ی فصل

در طول فاز ساخت[1] که سومین فاز از چرخه‌ی تولید در فرایند مبتنی بر آر.یو.پی و اولین فاز از مرحله‌ی تولید[2] می‌باشد، نسبت به فازهای قبل، پیشرفت قابل توجهی در پروژه بدست می‌آید. عمدتاً در یک فازِ ساختِ موفق، فعالیت‌های زیر انجام می‌شود:

- با بهره‌گیری از مزایای داشتن یک معماری تثبیت شده (از جمله به کمک مکانیزم‌های معماری)، قادر خواهیم بود نسبت به تولید و توسعه‌ی مقرون به صرفه‌ی فراورده اقدام نماییم. در ضمن با سازماندهی و هماهنگی سازمان در قالب تیم‌هایی حول معماری، می‌توان از قابلیت توسعه به صورت موازی نیز بهره‌مند شد.
- توانایی زیادتر کردن تعداد تیم‌های تولید با داشتن یک معماری پایدار و تثبیت شده و سازماندهی حول معماری.
- ایجاد و ارزیابی چندین نسخه‌ی میانی داخلی (نسخه‌های آلفا) برای حصول اطمینان از اینکه سیستم نیازهای کاربران را پوشش می‌دهد.
- در این فاز، از یک معماری قابل اجرا[3] به سمت ارائه‌ی اولین نسخه‌ی عملیاتی از سیستم، حرکت می‌نماییم. در پایانِ فازِ ساخت، نسخه‌ای از محصول تحت عنوان نسخه‌ی بِتا را در اختیار خواهیم داشت که شامل مؤلفه‌های نصب و راه‌اندازی، مستندات پشتیبانی، و نیز موادِ آموزشی می‌باشد.

---

[1] - Construction
[2] - Production Stage
[3] - Executable Architecture

پرسش‌هایی برای تحقیق و مطالعه‌ی بیشتر

۱. درباره‌ی تفاوت‌های فازِ ساخت در آر.یو.پی و فازِ پیاده‌سازی در رویکرد آبشاری تحقیق نمایید.

۲. در فازِ ساخت، عمدتاً چه نوع ریسک‌هایی مدیریت می‌شوند؟

۳. آیا ممکن است فازِ ساخت در یک پروژه بسیار کوتاه باشد؟ در صورت مثبت بودن پاسخ، در چه مواردی؟

۴. مدلِ پیاده‌سازی از مهم‌ترین دستاوردهای فازِ ساخت است. درباره‌ی محتویات این مدل تحقیق نمایید.

۵. در رابطه با انواع و قابلیت‌های ابزارهای مدیریت پیکربندی تحقیق نمایید.

۶. در رابطه با ملاحظات مرتبط با توسعه به صورتِ موازی[1] تحقیق نمایید.

۷. منبع‌یابی بیرونی[2] چیست؟ آر.یو.پی چگونه می‌تواند در منبع‌یابی بیرونی استفاده شود؟

[1] - Parallel Development
[2] - Outsourcing

منابع و مراجع

[1]. Ivar Jacobson, Grady Booch, James Rumbaugh, (1999). *The Unified Software Development Process*, Reading, MA: Addison-Wesley

[2]. Philippe Kruchten, (2003). *The Rational Unified Process: An Introduction*, Reading, MA: Addison-Wesley.

[3]. Per Kroll, Philippe Kruchten, (2003). *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*, Reading, MA: Addison-Wesley.

[4]. Rational Software Corporation. (2003) "Rational Unified Process 2003.06," Available at: http://www.rational.com/

[5]. Software Academy. (2006) "Unified Process Knowledge Base," Available at: http://www.unifiedProcess.info/

[6]. Scott W. Ambler, (2000). *The Unified Process Construction Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

فصل نهم

فازِ انتقال

مهم‌ترین موضوعات مورد بررسی در این فصل، عبارتند از:

- بررسی اهداف فازِ انتقال
- مفهوم تکرارهای متعدد در فازِ انتقال
- فازِ انتقال و چرخه‌ی تولید

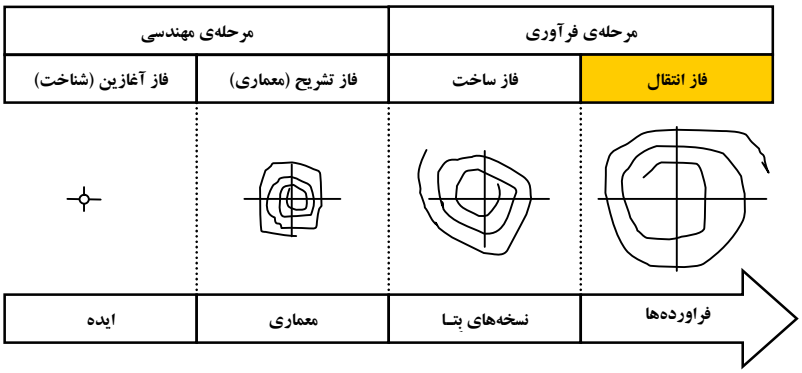
فازِ انتقال

۹

در این فصل، چهارمین و در واقع آخرین فاز از چرخه‌ی تولیدِ نرم‌افزار را که آخرین فاز از مرحله‌ی فراوری آن نیز می‌باشد، بررسی می‌نماییم. در انتهایِ این فاز، فراورده‌ی نرم‌افزاری به طور کامل به محیط مشتری و کاربران اِنتقال یافته و تمام کاربران نهاییِ سیستم قادر خواهند بود همه‌ی خدمات مورد نیازشان را بدون نیاز به حضور مستمر تولیدکنندگان، از سیستم دریافت نمایند؛ پروژه به طور کامل بسته شده و سیستم به مرحله‌ی نگهداری[۱] و تکامل[۲] وارد می‌گردد.

شکل ۹-۱

فاز اِنتقال، آخرین فاز از مرحله‌ی فرآوری در چرخه‌ی تولید

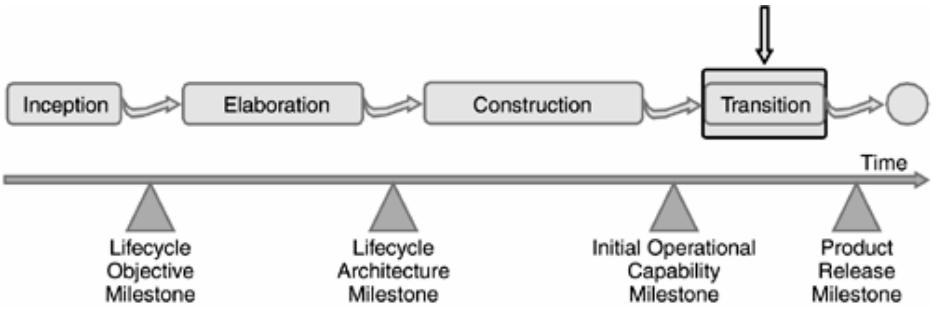


پایانِ این فاز، معادلِ آخرین نقطه‌ی تصمیم‌گیری سازمانی در چرخه‌ی تولید، یعنی ترخیصِ فراورده[۳] می‌باشد.

[1] - Maintenance
[2] - Evolution
[3] - Product Release Milestone

شکل ۹-۲

فازِ انتقال و چهارمین (آخرین) نقطه‌ی تصمیم‌گیری سازمانی در چرخه‌ی تولید



فازِ قبل، یعنی فازِ ساخت[1] با ارائه‌ی نسخه‌ای از نـرم‌افـزار دربرگیرنـده‌ی تمـام قابلیـت‌هـا، کارکِردهـا، و وظیفه‌مندی‌های[2] مورد انتظارِ ذینفعان، یعنی نسخه‌ی بِتـا[3]، پایان می‌یابد. این نـسخه، شـامل ابزارهـای لازم برای نصب، مستندات تکمیلی و پشتیبان[4]، و مواد و محتوای آموزشی[5] می‌باشد. امّا می‌دانیم که این نـسخه‌ی بِتـا، محصول نهایی نیست و هنوز لازم است که تنظیم‌ها و پیکربندی‌های دقیق‌تری روی وظیفـه‌منـدی‌هـا، کارایی، و به طور کلّی کیفیتِ آن انجام شود.

تمرکز اصلی فعالیت‌ها در فاز انتقال، بر این موضوع است که اطمینان یابیم، سیستمِ نرم‌افزاری تولید شده، به طورِ کامل نیازهای کاربرانش را برآورده می‌نماید. به طور معمول، در این فاز یک یا دو تکرار برنامـه‌ریـزی می‌شود که عمدتاً شاملِ تست و آزمون فراورده به منظور آمادسازی آن برای تحویل و نیز اِعمال تنظیمـات و اصلاحات جزئی بر اساس بازخوردِ دریافت شده از کاربران می‌باشد.

در این نقطه از چرخه‌ی تولید، نباید هیچ‌گونه مسأله یا ریسک عمده‌ای داشـته باشـیم و بنـابراین بیـشتر بازخوردهای کاربران باید متوجه مسائلی مانند پیکربندی، ملاحظات مرتبط با نصب، تنظیم دقیـق[6]، و مـسائل مرتبط با سهولت استفاده[7]، مانند تغییر برخی واسطهای کاربر باشد.

[1] - Construction Phase
[2] - Functionality
[3] - Beta Release
[4] - Supporting Documentation
[5] - Training Material
[6] - Fine-tuning
[7] - Usability

پروژه‌هایی که با پیچیدگی زیادی مواجه هستند، اغلب ممکن است نیازمند به چندین تکرار در فازِ انتقال باشند. در چنین حالتی، در هر یک از تکرارها، با اِعمال برخی تغییرات و تصحیح‌های مورد نیاز، نسخه‌ای قابل استقرار[۱] از سیستم ارائه می‌شود. در اغلبِ موارد، در این فاز مجبور خواهیم شد که برخی از خصیصه‌های[۲] سیستم را که پیش از این در فازهای قبل به منظور پایبندی به برخی ضرب‌العجل‌های[۳] تعیین شده در زمانبندی به تعویق افتاده بودند، کامل نماییم.

باید خاطرنشان کنیم که در رویکردِ آر.یو.پی، فازِ انتقال به طور اساسی با فاز انتهایی در سایر رویکردهای سنتی (مانند رویکرد آبشاری) متفاوت است، زیرا هنگامی که در آر.یو.پی، وارد فازِ نهایی می‌شویم، با نسخه‌ای تست‌شده، مجتمع[۴]، و پایدار[۵] از سیستم سر و کار داریم. در صورتی‌که در رویکرد سنتی آبشاری[۶]، در آخرین فاز که مرتبط با یکپارچه‌سازی[۷] می‌باشد، عمدتاً با شکستن و به هم ریختن‌های وسیعِ[۸] سیستم مواجه می‌شویم به گونه‌ای که حتی در برخی موارد ممکن است قادر به کامپایل[۹] تمام سیستم نباشیم، یا اینکه واسط‌های میان زیرسیستم‌ها با هم سازگاری نداشته باشند، یا اینکه سیستم پی در پی از کار می‌افتد؛ در نتیجه در فازِ آخرِ رویکرد آبشاری عموماً شاهد انجام دوباره‌کاری‌های زیادی می‌باشیم که تأخیرِ پروژه، نتیجه‌ی منطقی آن خواهد بود. با به وجود آمدن تأخیر، مدیریت باید وقت زیادی را صرف مذاکره‌ی مجدّد با ذینفعان و تنظیم مجدّد انتظارات و خواسته‌های ذینفعانِ کلیدی نماید. تجربه نشان داده است که در چنین مواردی افزودن نیروهای بیشتر برای تکمیل سریع‌تر کار نه تنها موجبات کم شدن تأخیر را فراهم نمی‌نماید، بلکه عملاً نتیجه‌ی آن تأخیر بیشتر و آشفتگی بیش از پیشِ پروژه می‌باشد.

---

[1] - Deployable
[2] - Features
[3] - Deadlines
[4] - Integrated
[5] - Stable
[6] - Traditional Waterfall Approach
[7] - Integration
[8] - Major Breakage
[9] - Compile

اهدافِ فازِ انتقال

مهمترین اهدافِ فازِ انتقال عبارتند از:

۱. انجام تست‌های بِتـا به منظور تأیید اینکه سیستم پاسخ‌گوی انتظارات کاربران می‌باشد.
۲. آموزش کاربران و نگهدارندگان[۱] سیستم به منظور دستیابی به قابلیت خوداتکایی[۲] آنان. این دسـته از فعالیت‌ها برای اطمینان از اینکه سازمان یا سازمان‌های پذیرنده‌ی نرم‌افـزار، شـرایط و قابلیـت‌هـای بکارگیری اصولیِ سیستم را داشته باشند، انجام می‌شود.
۳. آماده‌سازی محل استقرار و تبدیل[۳] بانک‌های اطلاعاتیِ عملیـاتی[۴]. بـرای اینکـه سیـستم جدیـد بـا موفقیت راه‌اندازی شود، ممکن است که لازم باشد سخت‌افزارهای جدیدی خریـداری شـود، فـضای جدیدی برای سخت‌افزارهای جدید افزوده شده و یا داده‌های موجود در بانک‌های اطلاعاتی فعلی به قالب مناسب برای سیستم جدید تبدیل شود.
۴. در حالتی که محصول به صورت یک بسته‌ی تجاری[۵] می‌باشد، آماده‌شدن برای بسته‌بندی، فرآوری، عرضه برای بازاریابی[۶]، توزیع، فروش، و نیز آموزش کارکنان مربوطه، ضروری می‌باشد.
۵. دستیابی به توافق تمام ذینفعان[۷] نسبت به اینکه نسخه‌های تثبیت‌شده در استقرار، کامـل بـوده و بـا معیارها و شرایط ارزیابی مورد بحث در چشم‌انداز[۸] پروژه، تطابق دارد.
۶. با ثبت درس‌های آموخته شده و تجربیات کسب شده در طول این چرخه، مقدمات بهبودهای آینده را فراهم نماییم. خصوصاً تجارب بدست آمده از بهبود فرایند و بکارگیری ابزارها اهمیت بسیاری دارد.

---

[1] - Maintainers
[2] - Self-reliability
[3] - Convert
[4] - Operational Databases
[5] - Commercial Package
[6] - Marketing Rollout
[7] - Stakeholders
[8] - Vision

تکرارها[1] در فازِ انتقال[2]

فازِ انتقال می‌تواند بر حسب نوعِ محصول، بسیار سر راست[3] و آسان و یا به شدّت پیچیده باشد (شکل ۹-۳ را ملاحظه نمایید). فعالیت‌های مرتبط با انتقالِ نسخه‌ی جدیدی از یک محصول دِسک‌تاپ[4]، احتمالاً خیلی ساده بوده و شاید تنها یک تکرار برای رفعِ چند نقص کوچک نیاز خواهیم داشت. در مقابل، جایگزینی یک سیستم کنترل ترافیک هوایی[5] ممکن است به شدت پیچیده بوده و مستلزم برنامه‌ریزیِ چندین تکرار باشد که در طی آن‌ها، سیستمِ جدید به تدریج با مجموعه سیستم‌های مختلف تلفیق‌شده و خصیصه‌های لازم به آن اضافه می‌شود. در ضمن، انتقال سیستم جدید و جایگزینی آن با سیستم قبلی، مستلزم فعالیت‌های بسیار پیچیده‌ای در سطح سازمان می‌باشد. این انتقال ممکن است شامل راه‌اندازی همزمان سیستم‌های قدیم و جدید، انتقال داده‌ها، آموزش کاربران، و تعدیل فرایندهای سازمانی باشد.

بر حسب اهداف یک پروژه، فعالیت‌های مختلفی در تکرارهای فازِ انتقال انجام می‌شود. فازِ انتقال در بیشتر پروژه‌ها منطقاً ساده است. بیشتر فعالیت‌ها که عمدتاً پیاده‌سازی و تست می‌باشد، به تصحیح خطاها و مشکلات موجود ارتباط دارد. گه‌گاه ممکن است لازم باشد که تعداد محدودی ویژگی جدید[6] به سیستم اضافه شود که در این صورت تکرار برنامه‌ریزی شده در این فاز، همانند تکرارهای فاز ساخت بوده و باید کمی روی نیازمندی‌ها[7]، تحلیل[8]، و طراحی[9] نیز کار شود.

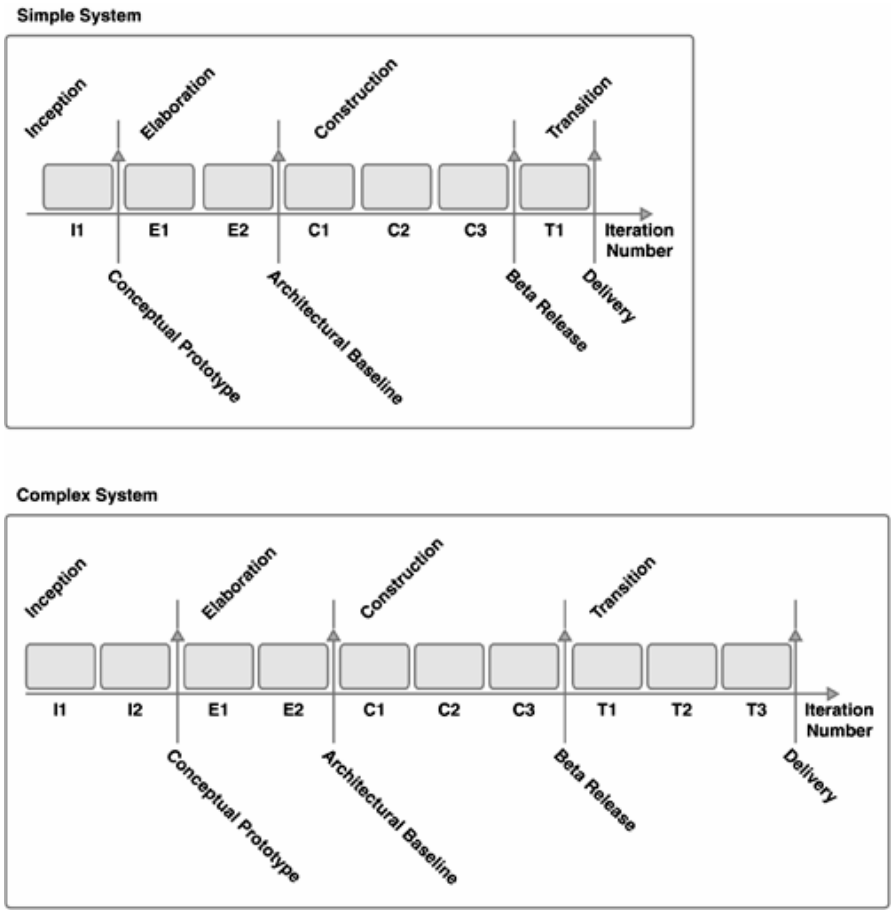
در رابطه با تولید محصولاتِ تجاری و نیز برخی اوقات که سیستم‌هایی برای کاربردِ گسترده‌ی داخلی ایجاد می‌شوند، باید مسائلی مانند بسته‌بندی[10]، فرآوری[11]، بازاریابی[1]، فروش، و پشتیبانی نیز مورد توجه قرار گیرد. این موضوع را در رابطه با هدف چهارمِ این فاز، بررسی خواهیم کرد.

---

[1] - Iterations
[2] - Transition
[3] - Straightforward
[4] - Desktop
[5] - Air-traffic control system
[6] - New Features
[7] - Requirements
[8] - Analysis
[9] - Design
[10] - Packaging
[11] - Production

شکل ۹-۳

تعداد تکرارها در فاز انتقال.



در حالتی که پروژه‌ی شما تولیدِ یک سیستم کاربردی می‌باشد که نیازمند به تعدادی سخت‌افزار جدید است، مانند یک سیستم پردازش مالی بزرگ، یا مستلزم تبدیل داده‌های سیستم قدیم می‌باشد، باید فعالیت‌هایی را که در هدف سوم فاز انتقال، یعنی آماده‌سازی محل استقرار و تبدیل بانک‌های اطلاعاتی عملیاتی توصیف شده، انجام دهید.

پروژه‌های بسیار پیچیده ممکن است نیازمند داشتن یک رویکرد تحویلِ تدریجی[۲] باشند که در این صورت در هر استقرار یک نسخه‌ی به تدریج کامل‌تر و با کیفیت بالاتر تولید می‌شود. این رویکرد، خصوصاً در مواردی که تنها راه تنظیمِ دقیقِ سیستم، گرفتنِ بازخورد از استفاده‌ی واقعی سیستم[۳] باشد، ضروری است. این

[1] - Marketing
[2] - Incremental Delivery
[3] - Actual System Usage

روش برای سیستم‌های بزرگ مدیریت اطلاعات[۱] یا سیستم‌های فرمان و کنترل[۲] با استقرارهای توزیع‌شده[۳] و سخت‌افزارهای پیچیده، مستلزم اینست که چندین سیستم با هم مجتمع[۴] شده و تنظیمِ دقیق[۵] شوند. در چنین پروژه‌هایی، تکرارهای فازِ انتقال، بسیار شبیه تکرارِ آخر یا دو تکرارِ انتهای فازِ ساخت می‌باشد (رجوع شود بـه فصل قبل)، که به آن پیچیدگیِ داشتن چندین استقرار نیز افزوده می‌شود. توصـیف پیچیـدگی ایـن دسـته از پروژه‌ها از حوصله‌ی مباحث این کتاب خارج است.

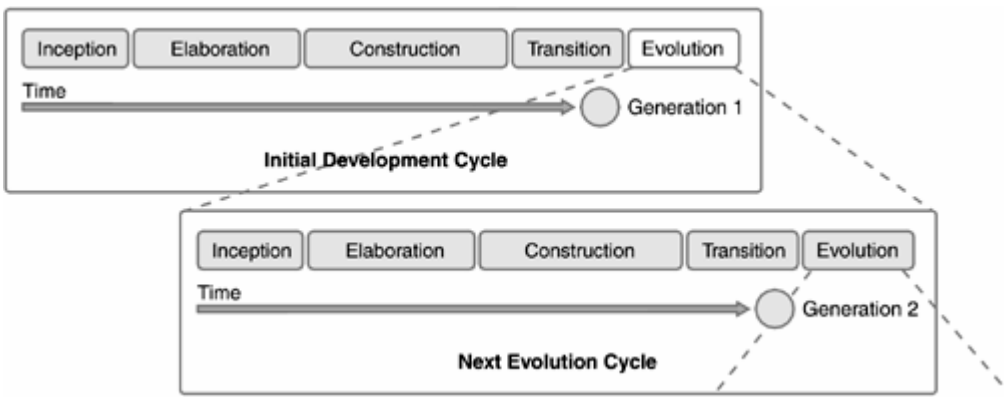
[1] - Large-scale Management Information Systems
[2] - Command and Control Systems
[3] - Distributed Deployment
[4] - Integrate
[5] - Fine-tune

فازِ اِنتقال و چرخه‌های توسعه[1]

یک گذر کامل از چهار فازِ فرایند (آغازین، تشریح، ساخت، و انتقال) را یک چرخه‌ی توسعه یا چرخه‌ی تولید می‌نامند. در پایان فازِ انتقال، یک چرخه‌ی توسعه‌ی کامل خواهیم داشت (شکل ۹-۴ را ملاحظه نمایید). در هر چرخه‌ی توسعه، یک نسل[2] از نرم‌افزار تولید می‌شود. به جز در حالتی که اصطلاحاً محصول می‌میرد[3] و به طور کل کنار گذاشته می‌شود، یک محصول در نسل بعدی‌اش با تکرار همان توالی از فازهای شناخت، معماری، ساخت و انتقال، اما با تاکیدی متفاوت روی فازهای مختلف که با توجه به اهداف جدید پروژه می‌باشد، رشد و تکامل[4] می‌یابد. این چرخه‌های بعدی[5] را چرخه‌های تکامل[6] می‌نامند.

شکل ۹-۴

چرخه‌ی توسعه و چرخه‌های تکامل



در پایانِ فازِ انتقال، باید به اهداف پیش‌بینی شده، دست یافته باشیم و پروژه در موقعیتی باشد که بتوان آن‌را خاتمه داد. در برخی از پروژه‌ها، پایان چرخه‌ی فعلی ممکن است همزمان باشد با شروع یک چرخه‌ی دیگر که منجر به نسل بعدی[7] همان محصول خواهد شد. در دیگر پروژه‌ها، پایان فازِ انتقال ممکن است

[1] - Development Cycles
[2] - Generation
[3] - Die
[4] - Evolve
[5] - Subsequent
[6] - Evolution Cycles
[7] - Next Generation

مصادف باشد با تحویل کامل دستاوردها به شخص ثالثی[1] که مسئولیت عملیاتی کردن، نگهداری، و توسعه‌ی آتی سیستم تحویل شده را بر عهده خواهد داشت.

توجه داشته باشید که در بسیاری از موارد، فازِ آغازین از یک چرخه‌ی تکامل با فازِ انتقال از چرخه‌ی توسعه، همپوشانی دارد. همزمان که در انتهای چرخه‌ی توسعه‌ی جاری، محصول در حال اِنتقال به محیط کاربران می‌باشد، می‌توان فازِ آغازینِ چرخه‌ی توسعه‌ی جدیدی را شروع کرد و احتمالاً برخی از نیازمندی‌هایی را که به سبب محدودیت‌ها و شرایط فعلی در چرخه‌ی فعلی امکان تحقّق آن‌ها وجود نداشته، در چرخه‌ی توسعه‌ی بعدی و در واقع در نسل بعدی فراورده، پیاده‌سازی خواهیم کرد.

البته، این امکان نیز وجود دارد که چرخه‌ی توسعه‌ی بعدی حتی زودتر از فاز انتقال و در طی فازهای قبل از آن شروع شود. اما توجه داشته باشید که در این صورت، پیچیدگی بسیار زیادی به وجود خواهد آمد و تنها سازمان‌هایی که دارای فرایندهای پیشرفته‌ی مدیریت پیکربندی[2] می‌باشند، قادر به چنین کاری هستند.

با وجودی که عمدتاً چرخه‌ی تکاملِ یک فراورده، مستلزم تعریف پروژه‌ای مبتنی بر آر.یو.پی و در برگیرنده‌ی هر چهار فاز آغازین، تشریح، ساخت، و اِنتقال می‌باشد، در بسیاری از پروژه‌ها، چرخه‌ی تکامل وارد یک سبک و اسلوب[3] دیگر می‌شود که در آن کارهای عملیاتی[4]، پشتیبانی[5]، و فعالیت‌های عادی و روزمره‌ی نگهداری[6] انجام می‌شود. در چنین حالتی ممکن است نیازی به یک سری فازهای جدید و از پیش تعریف شده نداشته باشیم؛ ساختِ یک نسخه‌ی جدید از نرم‌افزار که در آن خطاها و اشتباهات برطرف شده، همانند یک تکرار ساده در فاز انتقال است.

---

[1] - Third Party
[2] - Advanced Configuration Management Processes
[3] - Mode
[4] - Operations
[5] - Support
[6] - Routine Maintenance

برنامه‌ریزی برای نسخه‌ی بعدی[1]

در دنیای نرم‌افزار، تجربه نشان داده است که بیشتر نرم‌افزارها با تغییر خواسته‌های کاربران، تغییر فناوری، تغییر شرایط کسب و کار، و عواملی از این دست، به سرعت دچار تحوّل شده و مسیر تکامل خود را در پیش می‌گیرند. تقریباً بیشتر پروژه‌های نرم‌افزاری بلافاصله وارد چرخه‌ی بعدی توسعه، یعنی چرخه‌ی تکامل می‌شوند. این چرخه‌ی جدید، همه‌ی فازهای یک چرخه‌ی تولید یعنی فازهای آغازین (شناخت)، تشریح (معماری)، ساخت، و اِنتقال را در بر خواهد داشت.

برخی از دستاوردهای[2] کلیدی قابل تحویل[3] فازِ اِنتقال

دستاوردهای این فاز شباهت زیادی با دستاوردهای کلیدی فازِ ساخت دارند. با این تفاوت که در این فاز شکلِ کامل‌تر به خود می‌گیرند. دستاوردهای کلیدی فازِ اِنتقال عبارتند از:

- نرم‌افزار قابلِ اجرا به همراه ماجول‌های نصبِ آن
- اسناد حقوقی مانند قراردادها، لیسانس[4]، گارانتی‌ها
- کلیه‌ی مدل‌ها و دستاوردهای جانبی محصول
- راهنمای کاملِ کاربران، مدیران سیستم، اپراتورها، و نیز مواد و محتویاتِ آموزشی
- مراجع پشتیبانی از مشتری از جمله وب‌سایت پشتیبانی محصول که شامل اطلاعات بیشتری درباره‌ی محصول، امکان دادن گزارش خطا، و پیداکردن اطلاعاتی درباره‌ی بهبودها و به‌روز رسانی‌ها می‌باشد.

---

[1] - Next Release or Generation
[2] - Artifact
[3] - Key Deliverables
[4] - License

در ادامه، شش هدفِ کلیدی فازِ انتقال را بررسی خواهیم نمود. اما پیش از آن، اجازه دهید که نکات ذیل را یادآوری نماییم:

- اهدافِ ذکر شده برای فازها در آر.یو.پی، ماهیّت فعالیت ندارند، بلکه کلیه‌ی فعالیت‌هایی که در هر فاز انجام می‌شود باید حداقل در راستای دستیابی به یکی از این اهداف باشد.
- هیچ‌گونه ترتیب خاصی میان این اهداف وجود ندارد.
- همه‌ی اهداف دارای اهمیت می‌باشند. با وجودیکه اهدافِ فازها دارای اهمیت و حساسیت یکسانی نمی‌باشند، ولی توجه داشته باشید که در صورت عدم تحقّق حتی یکی از آنها، فاز به پایان نمی‌رسد.

هدف ۱. انجام تستِ بِتا[۱] به منظور اطمینان از برآورده شدن انتظارات کاربران[۲]

همانگونه که پیش از این نیز اشاره شد، اولین نسخه‌ی عملیاتیِ[۳] سیستم، یعنی نسخه‌ی بِتا، در انتهای فازِ قبل، یعنی فازِ ساخت، به دست آمده و مستقر[۴] گردید. در طی فازِ انتقال، باید تستِ بِتا را روی این نسخه‌ی عملیاتی انجام شود؛ این تست، بازخوردهای[۵] بسیاری را از کاربران تست‌کننده‌ی سیستم، در اختیار شما قرار می‌دهد. بنابراین باید راهکار و استراتژی مناسبی برای دریافت، سازماندهی، تحلیل، و در نهایت تصمیم‌گیری مناسب نسبت به این درخواست‌های تغییر[۶] داشته باشید. بدیهی است هرگونه ترتیب اثری نسبت به این درخواست‌های تغییر را که عمدتاً به صورت رفع خطاها و انجام پاره‌ای بهبودها و پیکربندی‌هاست، باید پیش از تحویلِ نهایی فراورده، انجام دهیم.

بخش عمده‌ای از درخواست‌های تغییر تنها برای انجام برخی بهبودهای جزئی، نظیر رفع خطاها و نقص‌های کوچک، بهبود مستندات یا موادِ آموزشی[۷]، یا تنظیمِ[۸] دقیق‌تر کارایی[۹] سیستم می‌باشند. برخی اوقات با درخواست‌هایی برای اضافه‌کردن یک ویژگی و قابلیت جدید روبرو می‌شویم. در این حالت در صورتی‌که

[1] - Beta Test
[2] - User Expectations
[3] - First Operational Version
[4] - Deployed
[5] - Feedback
[6] - Change Requests
[7] - Training Materials
[8] - Tuning
[9] - Performance

اضافه کردن ویژگی درخواست شده در چارچوب زمان و هزینه‌ی پیش‌بینی شده، قابل انجام باشد، باید مجدداً به سراغ نیازمندی‌ها، تحلیل و طراحی، پیاده‌سازی، و تست برویم.

البته به خاطر داشته باشید که نیاز به اضافه‌کردنِ یک ویژگی جدید در این فاز، عموماً بـه معنـای وجـود نقص و اشتباهاتی در فازهای گذشته است، اما ممکن است در پروژه‌های بزرگ، با این مـورد برخـورد داشـته باشیم. در این حالت، بهترین کاری که می‌توان انجام داد اینست که تا حد اِمکان ویژگی‌های جدیـدی را کـه نیاز و درخواست‌شان در این فاز مطرح می‌شود، به تعویق انداختـه و آن‌هـا را بـه چرخـه‌ی تکامـلِ[1] فـراورده، موکول نماییم. با این وجود، در مواردی هم ممکن است که بدون اضافه کردن ویژگی جدید، سیستم به خوبی نصب و راه‌اندازی نشود.

در طی فازِ انتقال باید زمان مناسبی را به بهبود مـستندات[2]، راهنمـایی‌هـای آن.لایـن[3] (بـرخط)، مـواد و محتوایِ آموزشی، رهنمودهای کـاربران[4]، رهنمودهـای عملیـاتی[5]، و دیگـر مـستندات تکمیلـی و پـشتیبانی، اختصاص دهید. تست و آزمونِ دقیق و مناسبِ این عناصر در محیط مقصد، بسیار با اهمیت می‌باشد.

تمرکز تست در فاز انتقال، عمدتاً متوجه بهبود کیفیت می‌باشد. علاوه بـر ایـن، در بـسیاری از پـروژه‌هـا، تست‌های پذیرش رسمی[6] نیز در این فاز انجام می‌شود.

در صورتیکه نقایص و خطاهای جدی و حیاتی در تست‌های بِتا کشف گردید، باید پس از رفع خطا، یـک نسخه‌ی جدیدِ بدونِ خطا، جایگزین نسخه‌ی مبنا قرار گرفته[7]، شـود. ایـن نـسخه را معمـولاً نـسخه‌ی پَـچ[8] (وصله) می‌نامند.

همانطور که در ابتدای این فصل ذکر شد، ممکن است در فازِ انتقال، بیش از یک تکرار[1] داشته باشیم؛ در این صورت نسخه‌های نتیجه‌ی این تکرارها به صورت، بِتای ۱ ، بِتای ۲ ، و مانند آن، نام‌گذاری می‌شوند.

---

[1] - Evolution Cycle
[2] - Documentations
[3] - Online Help
[4] - User’s Guides
[5] - Operational Guides
[6] - Formal Acceptance Test
[7] - Baselined Release
[8] - Patch

داشتن معیارهایی[۲] برای تشخیص زمان خاتمه‌ی فازِ انتقال بسیار ضروری است. بر خلاف تصور، خاتمه‌ی این فاز همیشه به سادگی انجام نمی‌شود. برای اینکه بتوانیم تشخیص دهیم که چه موقع ایـن فـاز را خاتمـه دهیم، باید معیارهایی مانند سنجش نقایص و تست‌ها در اختیار داشته باشیم. تحلیل این معیارها پاسـخ‌هـایی برای سوالات زیر فراهم می‌نماید:

- کیفیت در چه زمانی به حد مطلوب و کافی[۳] می‌رسد؟
- یافتن چه تعداد نقص بیشتری را می‌توانیم انتظار داشته باشیم؟
- چه موقع همه‌ی وظیفه‌مندی‌ها[۴] را تست کرده‌ایم؟

از آنجایی که داشتن معیارهایی برای پاسخ‌گویی به سؤالات اشاره شده و به تَبع آن، خاتمه‌ی فازِ انتقال و بستن پروژه، ضروری است، لازم است دو معیار کلیدی موفق، یعنی معیارِ نقایص[۵] و معیارِ تـست[۶] (آزمـون) را بیشتر بررسی نماییم.

برای داشتن معیار نقایص، باید موارد زیر را ثبت و ردگیری نماییم:

- چه تعداد نقایط جدید در هر روز کشف می‌شوند؟
- در هر روز چه تعداد از این نقایص، رفع می‌شوند؟

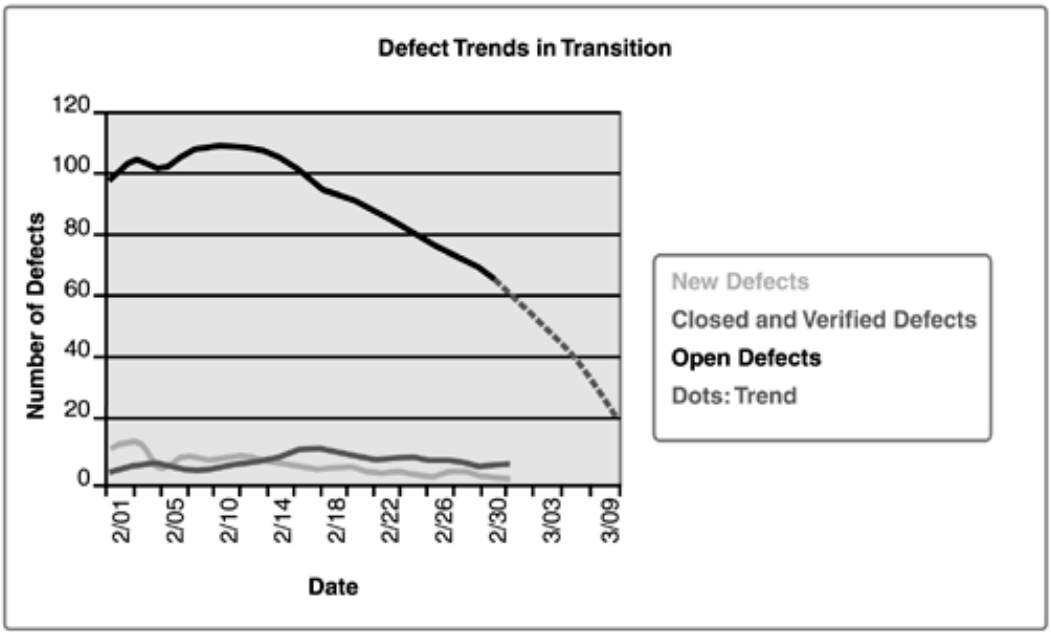
مهم‌تر از اعداد و ارقام بدست آمده باید به روندِ[۷] تغییر این اعداد و ارقام توجه داشته باشید. در شـکل ۹-۱۰ ، نمونه‌ای از یک نمودار پیگیریِ روندِ تغییر کشف و رفع نقایص نشان داده شده است. از روی این نمـودار می‌توان به طور تقریبی، تخمینی از تعداد خطاها در روزهای آینده ارائه نمود. اگر این تعـداد از یـک آسـتانه‌ی مشخص، که البته بر حسب نوع پروژه متفاوت می‌باشد، کمتر بود، می‌توان برای خاتمه‌ی فـازِ انتقـالْ دلایـل منطقی ارائه نمود.

---

[1] - Iteration
[2] - Metrics
[3] - Good Enough Quality
[4] - Functionality
[5] - Defect Metrics
[6] - Test Metrics
[7] - Trend

شکل ۹-۱۰

تحلیلِ روند کشف و رفعِ نقایص



معیارِ مهم دیگری که می‌تواند در زمینه‌ی تصمیم‌گیری در رابطه با کافی‌بودنِ آزمون‌ها و خاتمه‌ی فازِ انتقال مفید باشد، معیارِ تست نام دارد. این معیار عمدتاً به تکمیل تست‌های پیش‌بینی شده مرتبط می‌باشد. برای مثال، اگر تنها ۶۰ درصد تست‌های برنامه‌ریزی شده، کاملاً اجرا شدند، می‌توان انتظار داشت که بتوان نقایص بیشتری را با تکمیل همه‌ی تست‌ها، کشف نمود. شما می‌توانید با مشخص کردن تعداد نقایصی که معمولاً با انجامِ هر موردِ تست[1] کشف می‌شود و ضرب کردن آن در تعداد مواردِ تست، تخمینی از تعداد نقایص احتمالی در تست‌های باقی مانده، بدست بیاورید.

مطمئناً قصد ندارید که سیستمی را با تعداد نقایص زیاد از حد تحویل دهید؛ تحلیل روندِ تکمیل و موفقیت مواردِ تست، می‌تواند معیار خوب دیگری برای تصمیم‌گیری در رابطه با توقف آزمون‌ها و خاتمه‌ی فازِ انتقال باشد.

[1] - Test Case

هدف ۲. آموزش کاربران و نگهدارندگان[1] سیستم به منظور نیل به قابلیت خوداتکایی[2]

در طی فاز انتقال باید اطمینان یابیم که همه‌ی کاربران، نیروهای عملیاتی، و نیز تیم نگهداری، آموزش‌های لازم و مناسبی را برای بکارگیری سیستم گذرانده باشند. این آموزش‌ها فرصتی نیز برای آزمودنِ مواد و محتویات آموزشی، مستندات کاربران، و دست‌نامه‌های عملیاتی فراهم می‌نماید.

هنگامی که لازم است تعداد زیادی از کاربران را آموزش دهید، لازم است مستندات آموزشی مناسب و با کیفیتی فراهم کنید. بدین منظور باید از قبل و مثلاً در طی فازِ ساخت و یا حتی در فازِ تشریح، به ایجاد چنین مستندات و موادِ آموزشی اقدام کرده باشید. در چنین حالتی باید در همان فازهای قبل، آموزشیاران و مربیان آموزشی خود را نیز آماده کرده باشید.

هدف ۳. آماده‌سازی محل استقرار[3] و تبدیل[4] بانک‌های اطلاعاتیِ عملیاتی[5]

هنگامی‌که باید فراورده‌ی نرم‌افزاری خود را جایگزین یک سیستم موجود نمایید، انتقال نرم[6] و تدریجی سیستمِ جدید، کار بسیار پیچیده‌ای است. داده‌های فعلی سیستم موجود را باید انتقال داد و حتی ممکن است لازم باشد برای اطمینان از کارایی دقیق و قابل اطمینانِ سیستم جدید، این سیستم تا مدتی به طور موازی با سیستم قدیمی راه‌اندازی شود. این موضوع موجب انجام کارهای بیشتری مانند واردسازی داده به هر دو سیستم و بررسی کارایی و انطباق عملکرد سیستم جدید با سیستم قبلی می‌باشد. در برخی از موارد ممکن است که شما به فضا و تسهیلاتِ بیشتری برای جا دادنِ ماشین‌ها و ابزارهای جدید، نیاز داشته باشید.

توجه داشته باشید که ممکن است در انتقال داده‌های موجود به سیستمِ جدید، نیاز به استراتژی و ابزارهای خاصی داشته باشید. حتی ممکن است لازم باشد که خودتان ابزارهای لازم را در فازهای قبل تهیه نمایید.

---

[1] - Maintainers
[2] - Self-Reliability
[3] - Deployment Site
[4] - Convert
[5] - Operational Databases
[6] - Smooth Transition

هدف ۴. آماده‌سازی برای روانه‌کردن[۱] فراورده: بسته‌بندی[۲]، فرآوری[۳]، و معرفی به بازار[۴]

با وجودی‌که این هدف عمدتاً در شرکت‌هایی که تولیداتشان را وارد بازار می‌نمایند[۵]، مصداق دارد، برخی اقدامات مرتبط با این هدف می‌تواند در پروژه‌های که باید فراورده‌ی نرم‌افزاری برای تعداد زیادی از کاربران ارسال گردد نیز مفید باشد.

توجه داشته باشید که در پروژه‌های تجاری که باید در آنها یک بسته‌ی نرم‌افزاری تولید شود، اقدامات مرتبط با دستیابی به این هدف، باید خیلی زودتر و در فازهای ساخت و حتی تشریح، شروع شده باشد.

برای شناسایی و تفکیک همه‌ی اجزای بکار رفته در فراورده‌ی نهایی از مستندی تحتِ عنوانِ صورت وضعیتِ فراورده[۶] استفاده می‌شود. بسته‌ی نهایی یک فراورده‌ی نرم‌افزاری، شامل سیستم نرم‌افزاری قرار گرفته روی یک رسانه‌ی ذخیره‌سازی، برخی از مستندات و دستْنامه‌ها[۷]، فرم‌ها و اسناد مربوط به موافقت‌نامه‌ی لیسانس[۸]، و نیز خودِ مکانیزمِ بسته‌بندی[۹] می‌باشد.

اطمینان از قرارگیری همه‌ی اجزاء در محل مناسب‌شان بسیار ضروری است. سیستم نرم‌افزاری و نیز نرم‌افزارِ نصب و راه‌اندازی باید بررسی مجدد شوند. مستندات و دستْنامه‌ها نیز باید از لحاظ مناسب بودن برای چاپ و یا سهولت استفاده بررسی شوند. پس از اطمینان از صحت همه‌ی اجزاء ، فراورده‌ی نرم‌افزاری در اختیار بخش تولید انبوه قرار گرفته و به تعداد زیاد تکثیر می‌شود.

---

[1] - Launch
[2] - Packaging
[3] - Production
[4] - Marketing Rollout
[5] Independent Software Vendors (ISV – چنین شرکت‌هایی را می‌نامند.)
[6] - Bill of Materials (BOM)
[7] - Manuals
[8] - Licensing Agreements Forms
[9] - Packaging

اگر یک محصول تجاری را تولید کرده باشید، بایستی فراهم‌آوری چند دستاوردِ دیگر را نیز در نظر داشته باشید. این دستاوردها، عبارتند از:

- یک توصیف یک تا دو صفحه‌ای را در چند حالت مختلف، به صورت کوتاه و مختصر، متوسط، و نیز توصیفی بلند و مفصل از فراورده، موقعیت آن، و ویژگی‌ها و مزایای کلیدی آن، فراهم نمایید. این توصیف که عموماً سی.اِم.پی[۱] نامیده می‌شود، سنگ بنا و شالوده‌ای برای موفقیت معرفی فراورده می‌باشد.
- برخی اطلاعات فرعی و تکمیلی در رابطه با فراورده[۲] از جمله: مقالات فنی، اطلاعاتی روی وب‌سایت، دِموهایی[۳] از فراورده، ارائه‌های چند رسانه‌ای درباره‌ی فراورده، وایت‌پِی‌پِر[۴] و نیز داده‌های آماری مرتبط
- اطلاعات پشتیبانی فروش[۵]. ارائه‌های فروش[۶]، ارائه‌های فنی، اطلاعات آموزشی برای آشنایی با زمینه‌ی کاربرد فراورده[۷]، اطلاعاتی آماری درباره‌ی بکارگیری فراورده[۸]، مقالات توصیف موقعیتِ فراورده[۹]، راهنمایی‌هایی در رابطه با چگونگی دستیابی به اهدافِ فروش، مراجع[۱۰]، داستان‌هایی از موفقیت فراورده[۱۱]، و مانند آن.
- خبرنامه‌های داخلی، کاتالوگ‌ها و بروشورها، خلاصه‌های تحلیلی

---

[1] - CMP : Core Message Platform
[2] - این دسته از اطلاعات را اطلاعات ثانوی یا به اصطلاح Customer-Consumable Collateral. می‌نامند
[3] - Demo
[4] - Whitepapers
[5] - Sales Support Materials
[6] - Sales Presentations
[7] - Field Training Material
[8] - Fact Sheets
[9] - Positioning Papers
[10] - References
[11] - Success Stories

هدف ۵. بدست آوردن توافق[۱] همه‌ی ذینفعان[۲] نسبت به اینکه استقرار[۳] کامل شده است.

آزمونِ پذیرشِ فراورده[۴]، آخرین عملِ تست، پیش از استقرار نهایی نرم‌افزار می‌باشد. هدف این آزمون، تأیید آمادگی نرم‌افزار برای عملیاتی شدن و انجام وظیفه‌مندی‌ها[۵] می‌باشد.

سه استراتژی مرسوم برای انجام آزمونِ پذیرش عبارتند از:

- پذیرشِ رسمی[۶]
- پذیرشِ غیر رسمی[۷]
- تستِ بِتا[۸]

آزمون پذیرشِ رسمی، فرایندی است که مدیریت آن به خوبی و به طور دقیق انجام می‌شود. این آزمون، اغلب شکل خاصی از تستِ سیستم[۹] می‌باشد. آزمون‌ها (تست‌ها) به دقت برنامه‌ریزی و طراحی می‌شوند. مواردِ آزمون[۱۰] انتخاب شده باید زیر مجموعه‌ای از مواردِ آزمونِ مربوط به تستِ سیستم باشند. در برخی از سازمان‌ها، این آزمون به صورت کاملاً خودکار[۱۱] انجام می‌شود. این نوع آزمون اغلب به وسیله‌ی سازمانِ فراهم‌کننده[۱۲] (پیمانکار) و تحتِ کنترلِ پذیرنده[۱۳] (کارفرما)، یا به وسیله‌ی خودِ پذیرنده، و یا به وسیله‌ی یک سازمانِ سوم[۱۴] که طرف قرارداد با پذیرنده است، انجام می‌شود.

---

[1] - Concurrence
[2] - Stakeholder
[3] - Deployment
[4] - Product Acceptance Test
[5] - Functionalities
[6] - Formal
[7] - Informal
[8] - Beta Test
[9] - System Test
[10] - Test Case
[11] - Fully Automated
[12] - Supplier
[13] - Acquirer
[14] - Third Party

در آزمونِ پذیرشِ غیر رسمی[1]، روش‌ها و فرایندهای پیچیده‌ای وجود ندارد. بدون در نظر گرفتنِ مواردِ آزمونِ خاصی، قابلیت‌ها و وظیفه‌مندی‌های سیستم کشف شده و تست می‌شوند. هر تست‌کننده خود تعیین کننده‌ی آزمون‌ها می‌باشد. کنترل در این نوع آزمون، نسبت به آزمونِ رسمی، به مراتب کمتر می‌باشد. این نوع آزمون عمدتاً توسط سازمان کاربرِ نهایی[2] انجام می‌شود.

توجه داشته باشید که از هر استراتژی که استفاده نمایید، باید در رابطه با مواردِ تست و نیز چگونگیِ ارزیابی آنها پیش از پیاده‌سازی و اجرای آزمون پذیرش، توافق حاصل نمایید.

آزمونِ پذیرشِ فراورده، در اغلبِ موارد به اجرای نرم‌افزار و اطمینان از آمادگی آن محدود نشده و تمام دستاوردهای فراورده را که به مشتری تحویل شده‌اند، در بر می‌گیرد؛ ارزیابی دستاوردها[3] با توجه به نوع و ماهیت‌شان متفاوت می‌باشد. آر.یو.پی برای بسیاری از دستاوردهای کلیدی مانند سندِ معماری نرم‌افزار[4] و مواردِ کاربرد[5]، چک‌لیست‌ها[6]، و رهنمودهایی[7] ارائه کرده است که می‌تواند در ارزیابی برخی از دستاوردها مفید باشد.

هدف ۶. بهره‌گیری از درس‌ها و تجربه‌های کسب شده برای بهبود کارایی[8] پروژه‌های آینده

به طور کلی توصیه می‌شود که در پایان هر پروژه، زمانی را به تحلیل و مستندسازی آنچه گذشت، دستاوردها، نقاط ضعف و قوت، راه‌کارها، ریسک‌ها، و خلاصه تجربیات بدست آمده، اختصاص دهیم. این کار زمینه‌ی مناسبی برای بهبود فرایند و افزایش کارایی پروژه‌های آتی فرهم آورده و نیز به مدیریت دانش در سازمان کمک می‌نماید.

---

[1] - Informal Acceptance Test
[2] - End-User
[3] - Artifacts
[4] - Software Architecture Document
[5] - Use-Case
[6] - Checklists
[7] - Guidelines
[8] - Performance

در ضمن، می‌توانید بررسی نمایید که آیا کار انجام شده‌ای را می‌توان برای استفاده در پروژه‌های آتی بکار گرفت. در این صورت باید با حذف داده‌های حساس، قسمت‌های قابل استفاده‌ی مجدد را به طور جداگانه در محل مناسبی قرار دهید.

بازبینی پروژه: نقطه‌ی تصمیم‌گیری سازمانی یا گام اصلیِ[1] ترخیص محصول[2]

فازِ انتقال با رسیدن به چهارمین گام اصلی پروژه، یعنی گام ترخیص محصول، پایان می‌یابد. تصمیم‌گیری کلیدی که در این گام انجام می‌شود به این موضوع اختصاص دارد که آیا همه‌ی ذینفعان به اهداف تدوین شده در چشم‌انداز پروژه دست یافته‌اند و نیز آیا یک شروع یک چرخه‌ی توسعه‌ی دیگر ضرورت دارد یا خیر. ممکن است چندین چرخه‌ی توسعه در فاز شناخت و یا در فازهای بعد برنامه‌ریزی شده باشد. در برخی موارد، این گام اصلی، با پایان فاز شناخت از چرخه‌ی بعدی توسعه‌ی محصول، همزمان می‌شود.

مهم‌ترین سؤالاتی که در این گام باید پرسیده شوند، عبارتند از:

- آیا خواسته‌های مورد توافق با کاربران، تحقق یافته است؟ آیا کاربران ارضاء شده‌اند؟
- آیا هزینه‌های مالی و زمانی و نیز منابع اختصاص یافته مطابق برنامه‌ریزی‌ها و پیش‌بینی‌های قبلی بوده است؟ اگر خیر، چه کارها و اقداماتی باید در پروژه‌های آینده انجام شود؟

وقتی که در گام ترخیص محصول قرار می‌گیریم، محصول در مرحله‌ی فرآوری قرار می‌گیرد و فرآیندهای نگهداری، عملیاتی‌کردن، و پشتیبانی آغاز می‌گردد. این مرحله ممکن است خود شامل چرخه‌ی توسعه‌ی جدیدی برای بهبود محصول فعلی باشد.

[1] - Major Milestone
[2] - Product Release Milestone

چکیده‌ی فصل

در طی فاز انتقال که چهارمین و آخرین فاز از چرخه‌ی توسعه‌ی آر.یو.پی[۱] و نیز آخرین فاز از مرحله‌ی فرآوری[۲] یا تولید می‌باشد، باید اطمینان یابیم که محصول نرم‌افزاری تولید شده، برآورده‌کننده‌ی نیازهای کاربران بوده و می‌تواند با موفقیت در محیط مقصد[۳] مستقر شود. در این فاز، انجام فعالیت‌های مختلف برای دستیابی به اهداف زیر می‌باشد:

- انجام تست‌های بِتا[۴] با مجموعه‌ی کوچکی از کاربران واقعی سیستم و انجام تنظیمات دقیق‌تر[۵] درصورت لزوم
- آموزش کاربران و نگهدارندگان[۶] سیستم به گونه‌ای که برای استفاده از سیستم، به اطلاعات موجود در فراورده متکی باشند (ویژگی خود اتکایی[۷]).
- آماده‌سازی محل برای استقرار، تبدیل بانک‌های اطلاعاتی موجود و به طور کلی، بسترسازی برای عملیاتی شدنِ موفقِ سیستم
- راه‌اندازی سیستم با توجه به بسته‌بندی و فرآوری؛ عرضه برای بازاریابی[۸]، پخش و توزیع[۹] و فروش و نیز آموزش نیروهای مربوطه. البته این موضوع بیشتر در رابطه با محصولات تجاری مصداق دارد.
- دستیابی به توافق[۱۰] همه‌ی ذینفعان[۱۱] نسبت به اینکه مبناهای استقرار[۱۲] (محصول تکمیل شده در طی فازِ انتقال) کامل بوده و با معیارهای ارزیابی چشم‌انداز پروژه، تطابق دارد.
- تحلیل و ثبت آموخته‌ها، نکات مثبت و منفی، تجارب، و مشکلات به منظور بهبود کارایی پروژه‌های آینده

---

[1] - RUP Lifecycle
[2] - Production Stage
[3] - Target Environment
[4] - Beta Tests
[5] - Fine Tune
[6] - Maintainers
[7] - Self-reliant
[8] - Rollout to Marketing
[9] - Distribution
[10] - Concurrence
[11] - Stakeholders
[12] - Deployment Baselines

پرسش‌هایی برای تحقیق و مطالعه‌ی بیشتر

۱. درباره‌ی تفاوت‌های فازِ انتقال در آر.یو.پی و فازِ تست در رویکرد آبشاری تحقیق نمایید.

۲. در فازِ انتقال، عمدتاً چه نوع ریسک‌هایی مدیریت می‌شوند؟

۳. آیا ممکن است فازِ انتقال یک پروژه بسیار کوتاه باشد؟ در صورت مثبت بودن پاسخ، در چه مواردی؟

۴. در باره‌ی ویژگی‌های یک فراورده‌ی نرم‌افزاریِ کامل، تحقیق نمایید.

منابع و مراجع

[1]. Ivar Jacobson, Grady Booch, James Rumbaugh, (1999). *The Unified Software Development Process*, Reading, MA: Addison-Wesley

[2]. Philippe Kruchten, (2003). *The Rational Unified Process: An Introduction*, Reading, MA: Addison-Wesley.

[3]. Per Kroll, Philippe Kruchten, (2003). *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*, Reading, MA: Addison-Wesley.

[4]. Rational Software Corporation. (2003) "Rational Unified Process 2003.06," Available at: http://www.rational.com/

[5]. Pankaj Jalote, (2002). *Software Project Management in Practice*, Reading, MA: Addison-Wesley.

[6]. Software Academy. (2006) "Unified Process Knowledge Base," Available at: http://www.unifiedProcess.info/

[7]. Scott W. Ambler and Larry L. Constantine, (2001). *The Unified Process Transition Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

بخش چهارم

فصل دهم: ساختارِ محتوایی آر.یو.پی

فصل یازدهم: دیسیپلینِ مدیریتِ پروژه

فصل دوازدهم: دیسیپلینِ مدل‌سازی سازمان

فصل سیزدهم: دیسیپلینِ نیازمندی‌ها

فصل چهاردهم: دیسیپلینِ تحلیل و طراحی

فصل پانزدهم: دیسیپلینِ پیاده‌سازی

فصل شانزدهم: دیسیپلینِ تست

فصل هفدهم: دیسیپلینِ اِستقرار

فصل هجدهم: دیسیپلینِ محیط

فصل نوزدهم: دیسیپلینِ مدیریتِ پیکربندی و تغییرات

فصل دهم

ساختارِ محتوایی آر.یو.پی

مهم‌ترین موضوعات مورد بررسی در این فصل، عبارتند از:

- معرفی ساختارِ محتوایی یا استاتیکِ فرایند
- آشنایی با عناصر محتوایی در فرایند آر.یو.پی:
  - نقش‌ها،
  - فعالیت‌ها،
  - دستاوردها

ساختارِ محتوایی آر.یو.پی

۱۰

در این فصل ساختار محتوایی یا استاتیکِ آر.یو.پی را بررسی خواهیم نمود. همان‌گونه که در فصل چهارم نیز اشاره شد، آر.یو.پی دارای دو بُعد می‌باشد. یک بُعدِ پویا (دینامیک) که در آن ملاحظات زمانی، مانند فازها و تکرارها مطرح است و یک بُعدِ ایستا (استاتیک) که در آن مفاهیمی مانند نقش‌ها[1]، فعالیت‌ها[2]، دستاوردها[3]، جریان‌های کار[4]، دیسیپلین‌ها[5]، و سایر عناصر بکار رفته در توصیفِ محتوای فرایند، مطرح می‌باشد.

در این فصل، مفاهیمِ ساختاری شکل‌دهنده‌ی بُعدِ استاتیکِ فرایند را بررسی خواهیم نمود. فصل‌های یازده تا نوزده، به معرفی دیسیپلین‌ها که مهم‌ترین اجزاء ساختاری فرایند آر.یو.پی می‌باشند، اختصاص دارد. مسلماً، درکِ بهتر و کامل‌ترِ مطالب این فصل، مستلزم مرور مطالب از روی نسخه‌ی نرم‌افزاری آر.یو.پی می‌باشد.

[1] - Roles
[2] - Activities
[3] - Artifacts
[4] - Workflows
[5] - Disciplines

مُدلِ ساختاری آر.یو.پی

یک فرایندِ تولید، با تعریف اینکه **چه‌کسی**[1]، **چه‌کاری**[2] را، **چـه‌موقـع**[3]، و **چگونـه**[4] بـرای تولیـدِ موفـقِ یـک فراورده‌ی دارای کیفیت مطلوب، باید انجام دهد، زمینه‌ی مناسبی را برای تحقّق کار تیمی فـراهم مـی‌نمایـد. ساختار آر.یو.پی به عنوان چارچوبی برای فرایندهای تولیدِ نرم افزار، از پنج مؤلفه‌ی اصلی تشکیل شده اسـت. این مؤلفه‌ها عبارتند از:

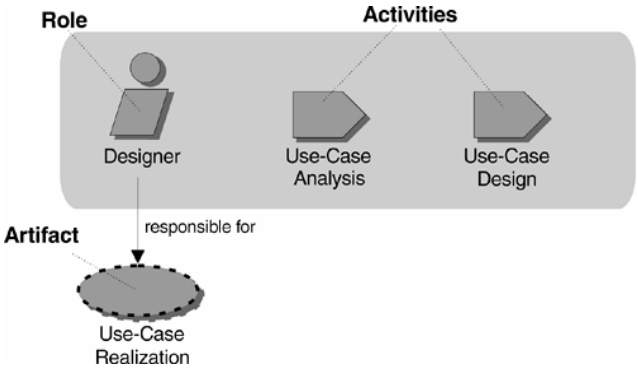
- نقش‌ها[5]: معادل مؤلفه‌ی «چه کسی» در توصیف یک فرآیند،
- فعالیت‌ها[6]: معادل مؤلفه‌ی «چگونه»،
- دستاوردها[7]: معادل «چه چیزی»،
- جریان‌های کار[8]: معادل «چه زمانی»،
- دیسیپلین‌ها[9]: ظرفی برای چهار نوع عنصرِ قبل.

شکل ۱۰-۱ ، بیانگر ارتباط میان سه عنصر اول در ساختار آر.یو.پی می‌باشد. شکل ۱۰-۲ ، ارتباط میـان مؤلفه‌های استاتیک و دینامیکِ آر.یو.پی را نشان می‌دهد و شکل ۱۰-۳ ، نمونه‌ای از یـک جریـان کـار را در آر.یو.پی به تصویر کشیده است.

[1] - Who
[2] - What
[3] - When
[4] - How
[5] - Role
[6] - Activity
[7] - Artifact
[8] - Workflow
[9] - Discipline

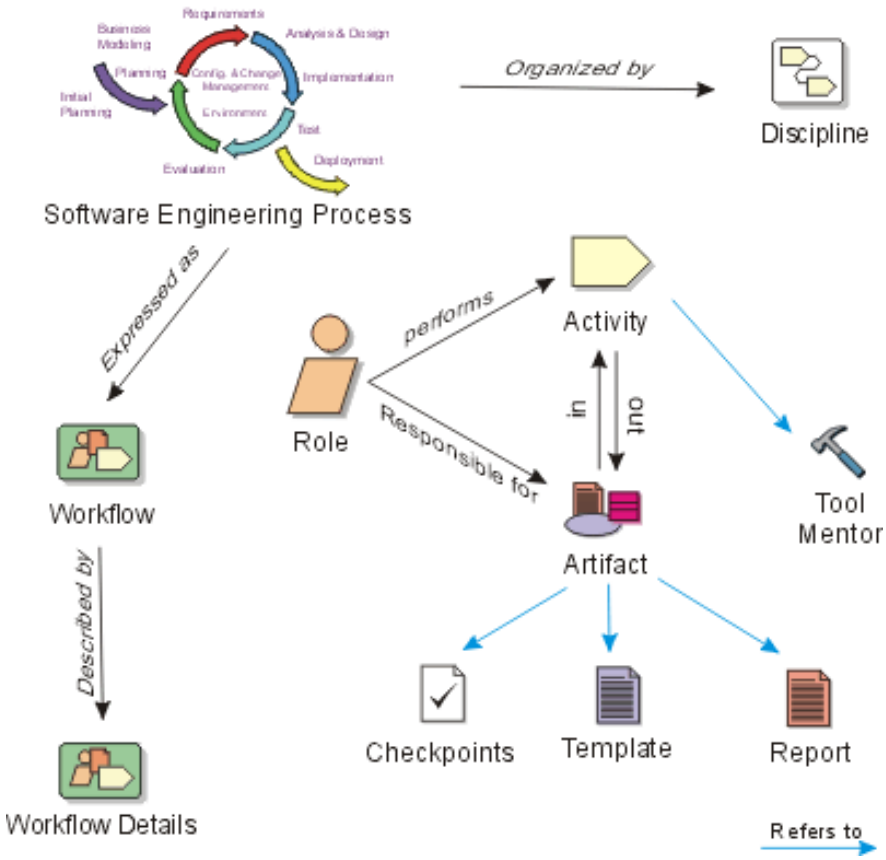
شکل ۱۰-۱

ارتباط میان نقش‌ها، فعالیت‌ها، و دستاوردها در فرایند آر.یو.پی



شکل ۱۰-۲

ارتباط میان مفاهیم استاتیک و دینامیک در آر.یو.پی



نقش‌ها[1]

نقش، تعریف‌کننده‌ی رفتار[2] و مسئولیت‌های[3] یک شخص یا گروهی از افراد است که با هم در یک تیم فعالیت می‌نمایند. رفتار در قالب فعالیت‌هایی[4] که یک نقش انجام می‌دهد، بیان می‌شود و هر نقشی با مجموعه‌ای از فعالیت‌های وابسته به هم در ارتباط است. وابستگی فعالیت‌ها در اینجا بدین معناست که مجموعه‌ی خاصی از فعالیت‌ها توسط یک شخصِ دارای مهارت‌های خاص، به شکل مناسب‌تر و بهتری قابل انجام است. مسئولیت‌های هر نقش معمولاً در ارتباط با دستاوردهای[5] مشخصی که آن نقش ایجاد، ویرایش، و کنترل می‌نماید، توصیف می‌شود.

این نکته که در آر.یو.پی ارتباط میان یک نقش و یک دستاورد به صورت رابطه‌ی مسئولیت، تعریف می‌شود، در عین حال که ساده به نظر می‌رسد، از اهمیت بسیاری برخوردار است. در واقع، تک‌تکِ دستاوردهای فرآیند، دارای مسئول مشخصی می‌باشند و این موضوع این امکان را به وجود آورده است که هر نقشی مسئولیت کیفیت آنچه را که تولید می‌نماید، برعهده داشته باشد.

توجه داشته باشید که مفهومِ نقش در آر.یو.پی با عناوین شغلی متفاوت می‌باشد و در ضمن لزومی ندارد که به تعداد نقش‌ها، افراد متفاوتی داشته باشیم! شما می‌توانید ساعتی در نقش مدیر پروژه فعالیت داشته باشید، سپس نقش یک معمار را بر عهده بگیرید و نقش‌های برنامه‌نویس و کارشناس تست را نیز در اواخر روز ایفا نمایید! در ضمن شما و دو نفر از همکاران‌تان ممکن است بطور همزمان نقش تحلیل‌گر را بر عهده داشته باشید.

در آر.یو.پی نقش‌های زیادی معرفی شده است (در حدود ۴۰ نقش) اما در یک پروژه‌ی خاص، لزومی ندارد که تمامی این نقش‌ها به کار گرفته شوند. معمولاً پس از شناسایی و تعیین دستاوردهای مطلوب و

---

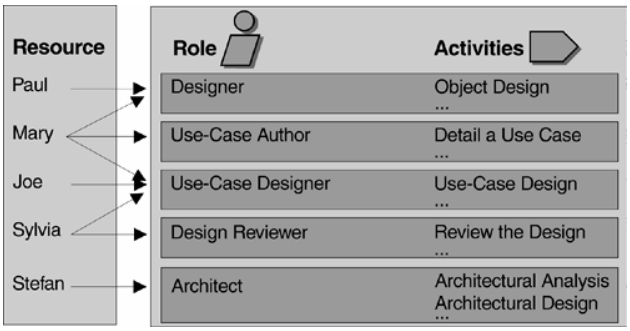
[1] - Role
[2] - Behaviour
[3] - Responsibility
[4] - Activity
[5] - Artifact

ضروری، مسئولین تولید، به‌روز رسانی، و مدیریت این دستاوردها که همان نقش‌های موجود در فرایند می‌باشند، تعیین می‌گردند.

در شکل ۱۰-۳ ، نمونه‌ای از نگاشت نقش‌ها به افراد در یک پروژه‌ی نرم‌افزاری نشان داده شده است.

شکل ۱۰-۳

نمونه‌ای از نگاشت میان نقش‌ها به افراد یک تیم



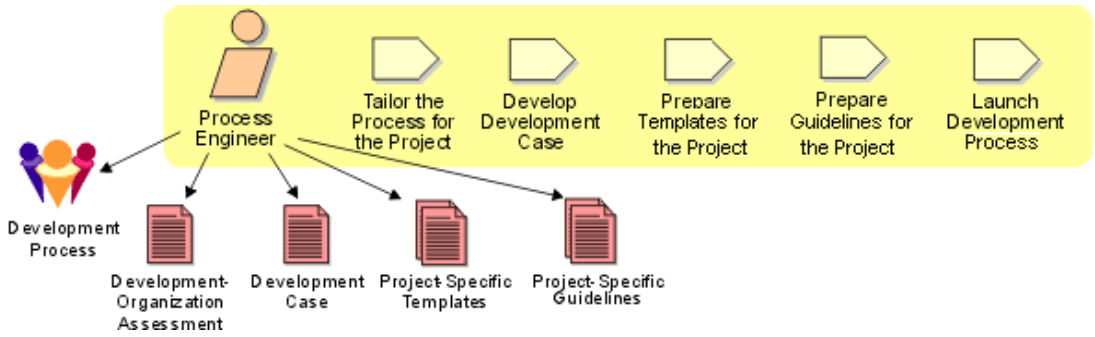
آر.یو.پی برای هر یک از نقش‌ها، اطلاعات ارزنده‌ای فراهم نموده است، از جمله:

- توصیف مختصری از مهارت‌ها[1] و شرایطِ کاری که ایفا کننده‌ی نقش باید در نظر داشته باشد،
- نوع تخصیص اشخاص به آن نقش،
- فعالیت‌هایی را که هر نقش انجام می‌دهد،
- دستاوردهایی را که یک نقش مسئولیت آنها را برعهده دارد.

[1] - Skills

شکل ۱۰-۴

مثالی از ارتباط یک نقش با فعالیت‌ها و دستاوردها



به علاوه، در آر.یو.پی نقش‌ها در پنج دسته‌ی مختلف به شرح زیر طبقه‌بندی شده‌اند:

- نقش‌های تحلیلی[۱] : مانند تحلیل‌گر فرایندهای سازمانی[۲] و تحلیل‌گر سیستم[۳]
- نقش‌های مرتبط با توسعه‌دهندگان[۴] : مانند معمار[۵] و طراح بانک اطلاعاتی[۶]، و برنامه‌نویس
- نقش‌های مرتبط با تست[۷] : مانند طراح تست[۸] و مدیر تست[۹]
- نقش‌های مدیریتی[۱۰] : مانند مدیر پروژه[۱۱] و مهندس فرایند[۱۲]
- نقش‌های مرتبط با تولید محصول و پشتیبانی[۱۳] : مانند کارشناس ابزارها[۱۴] و مدیر پیکربندی[۱۵]

[1] - Analyst Roles
[2] - Business-Process Analyst
[3] - System Analyst
[4] - Developer Roles
[5] - Architect
[6] - Database Designer
[7] - Tester Roles
[8] - Test Designer
[9] - Test Manager
[10] - Manager Roles
[11] - Project Manager
[12] - Process Engineer
[13] - Production and Support Roles
[14] - Tool Specialist
[15] - Configuration Manager

شکل ۱۰-۵

نمایی از دسته‌بندی عناوین و محتویات ساختاری فرایند در آر.یو.پی

شکل ۱۰-۶

شمای کلی دسته‌بندی نقش‌ها در محیطِ آر.یو.پی

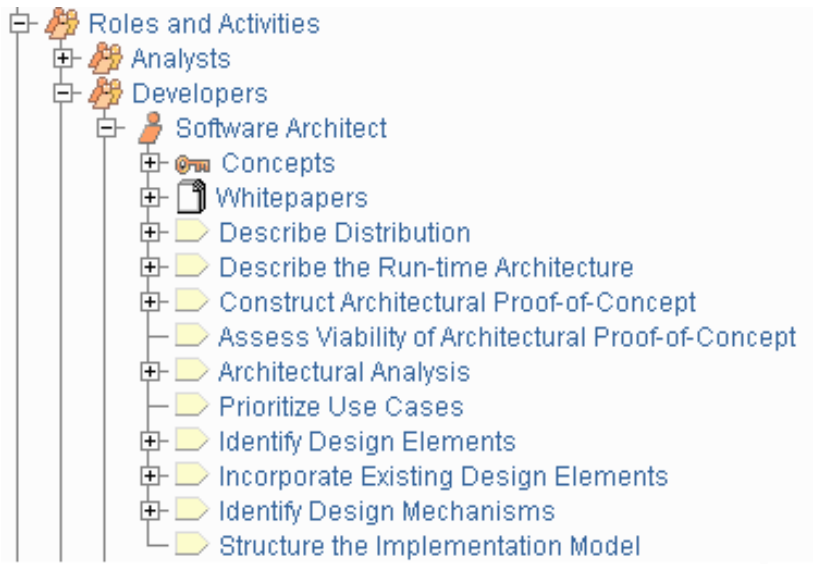
فعالیت‌ها[1]

نقش‌های مختلف در طول فرایند، یکسری فعالیت را انجام می‌دهند. فعالیت، عبارتست از یک واحدِ کار[2] که از یک شخص ایفا کننده‌ی یکی از نقش‌های فرایند، انجامِ آن انتظار می‌رود و در اثر انجامِ آن، نتیجه‌ای معنی‌دار[3] برای پروژه حاصل می‌گردد. هر فعالیت، هدف مشخصی دارد که عمدتاً به صورتِ تولید یا به‌روز رسانی دستاوردها[4] مانند به‌روز رسانی یک مدل، یک کلاس، یا یک طرح می‌باشد. در آر.یو.پی، هر کدام از فعالیت‌ها به یک نقش خاص اختصاص یافته است.

در شکل ۱۰-۷، مجموعه فعالیت‌های متناظر با یک نقش در محیط آر.یو.پی نشان داده شده است.

شکل ۱۰-۷

نمونه‌ای از فعالیت‌های متناظر با یک نقش



مدت زمان لازم برای انجام یک فعالیت ممکن است از چند ساعت (یا حتی کمتر از آن) تا چند روز باشد. به طور معمول، یک نفر که نقش متناظر با فعالیتی را برعهده دارد، با انجام آن فعالیت، روی یک یا تعداد

[1] - Activities
[2] - Unit of Work
[3] - Meaningful Result
[4] - Artifacts

محدودی دستاورد[1] تاثیر می‌گذارد. مدیریت پروژه می‌تواند از فعالیت‌های تعریف شده در آر.یو.پی، بـه منظـورِ برنامه‌ریزی و تخصیص منابع استفاده نماید. بدن منظور، این فعالیت‌ها نباید نه خیلی کوچک باشد که نادیـده گرفته شود و نه خیلی بزرگ در نظر گرفته شود که مدیریت آنها مشکل باشد.

یک فعالیتِ مشخص ممکن است چندین بار روی یک دستاوردِ خاص تکرار شود. ایـن امـر خـصوصاً در تکرارهای مختلف در طول فرایند و با دستیابی به شناخت بیشتر از سیستم اتفاق می‌اُفتد. توجّه داشته باشید که فعالیت‌ها در صورت تکرار، همواره توسط یک نقش مشخص انجام می‌شوند، ولی امکان دارد افراد مختلفی در طول زمان و در تکرارهای مختلف، نقش مورد نظر را ایفا نمایند.

از منظر مفاهیم شیءگرایی، نقش، یک شیء فعال[2] می‌باشد و فعالیت‌هایی که یک نقش انجام می‌دهـد، معادل مجموعه عملکردها[3] و متدهای آن شیء می‌باشند.

در اینجا چند نمونه از فعالیت‌های تعریف شده در آر.یو.پی آمده است:

- برنامه‌ریزی یک تکرار[4]، که به وسیله‌ی نقشِ مدیر پروژه انجام می‌شود،
- پیداکردن موارد کاربرد و اَکتورها[5]، که به وسیله‌ی نقشِ تحلیل‌گر سیستم انجام می‌شود،
- اجرای تست کارایی[6]، که بوسیله‌ی نقشِ تست‌کننده‌ی کارایی[7] انجام می‌شود.

اغلب فعالیت‌ها در آر.یو.پی با پیشوند «Activity» مشخص می‌شوند. به عنوان مثال، فعالیت پیـداکردن موارد کاربرد و اَکتورها به صورت «Activity: Find use case and actors» آمده است.

هر یک از فعالیت‌ها به مجموعه‌ای از گام‌ها[1] شکسته می‌شوند. با این وصف، کوچکترین کاری که در یک پروژه قابل انجام است، گامی است از یک فعالیت. بطور کلی سه نوع گام برای فعالیت‌های مختلف وجود دارد:

---

[1] - Artifact
[2] - Active Object
[3] - Operations
[4] - Plan an iteration
[5] - Find Use Cases and Actors
[6] - Execute a Performance Test
[7] - Performance Tester

- گام‌های اندیشیدن[۲] : در این دسته از گام‌ها، شخصی که نقش را ایفا می‌نماید، ماهیت کار را درک کرده و دستاوردها و مستندات مورد نیاز برای انجام فعالیت را جمع‌آوری و بررسی می‌نماید. در نهایت، نتیجه را به شکل مناسبی تنظیم می‌کند.
- گام‌های انجام دادن[۳] : در این دسته از گام‌ها، شخصی که نقش را ایفا می‌نماید یک یا چند دستاورد را ایجاد یا به‌روز رسانی می‌کند.
- گام‌های بازبینی[۴] : در این دسته از گام‌ها، نتایج حاصل از انجام فعالیت، با معیارها و شرایط کمّی و کیفی موجود مقایسه می‌شود.

برای مثال فعالیتِ « پیداکردن مواردِ کاربرد[۵] و اَکتورها[۶] » دارای گام‌های زیر می‌باشد:

۱. پیداکردن اَکتورها
۲. پیداکردن مواردِ کاربرد
۳. توصیف نحوه‌ی تعامل اَکتورها و مواردِ کاربرد
۴. بسته‌بندی مواردِ کاربرد و اَکتورها
۵. بازنمایی مدل مواردِ کاربرد در دیاگرام‌های مواردِ کاربرد
۶. ایجاد یک مستند بررسی[۷] از مدل مواردِ کاربرد
۷. ارزیابی نتایج

در نمونه فعالیت ذکر شده، بخش پیدا کردن (گام‌های ۱ تا ۳) به تفکر و اندیشه نیاز دارد؛ بخش انجام دادن (گام‌های ۴ تا ۶) شامل گرفتن نتایج و ارائه‌ی آنها در قالب مدل مواردِ کاربرد می‌باشد؛ بخش بازبینی (گام ۷) مستلزم ارزیابی نتایج برای اطمینان از کامل بودن و دربرداشتن سایر معیارهای کیفی است.

---

[1] - Steps
[2] - Thinking Steps
[3] - Performing Steps
[4] - Reviewing Steps
[5] - Use-Cases
[6] - Actors
[7] - Survey

توجه داشته باشید که با اجرای یک فعالیت، لزوماً همه‌ی گام‌های آن انجام نمی‌شود. به عبارت دیگر، ممکن است نحوه‌ی انجام یک فعالیتِ خاص در یک تکرار با انجام همان فعالیت در یک تکرارِ دیگر متفاوت باشد.

دستاوردها[1]

فعالیت‌های مختلف دارای دستاوردهایی به عنوان ورودی و خروجی می‌باشند. دستاورد، عبارتست از قطعه‌ای از اطلاعات[2] که در طی یک فرایند، تولید شده، تصحیح می‌شود، یا مورد استفاده قرار می‌گیرد. دستاوردها، محصولات قابل لمسِ[3] پروژه می‌باشند و عبارتند از چیزهایی که پروژه در طی مدت زمان رسیدن به فراورده‌ی نهایی تولید کرده یا استفاده می‌نماید. نقش‌های مختلف برای انجام فعالیت‌های تعریف شده در فرایند، از دستاوردها به عنوان ورودی استفاده می‌نمایند و حاصل انجام این فعالیت‌ها، یک یا چند دستاورد می‌باشد.

از آنجایی که ماهیت فعالیت‌ها و عملکردهای[4] اشیاء در دیدگاه شیءگرا شبیه به هم می‌باشد، می‌توان دستاوردها را پارامترهای ورودی و خروجی یک فعالیت قلمداد کرد. البته، در این صورت نقش‌ها را مشابه اشیاء فعال تصوّر کرده‌ایم.

دستاوردها، شکل‌های مختلفی به خود می‌گیرند، از جمله:

- یک مدل[5]، مانند مدل مواردِ کاربرد[6] یا مدل طراحی[7]
- عنصری از یک مدل[8]، مانند یک کلاس، یک موردِکاربرد، یا یک زیرسیستم[9]
- یک سند[1]، مانند سند چشم‌انداز[2] یا سندِ معماری نرم‌افزار[3]

---

[1] - Artifacts
[2] - A piece of information
[3] - Tangible
[4] - Operations
[5] - Model
[6] - Use Case Model
[7] - Design Model
[8] - A Model Element
[9] - Subsystem

- کد نرم‌افزار[۴]
- فایل اجرایی[۵]

توجه داشته باشید که واژه‌ی دستاورد[۶]، عمدتاً مختصِ آر.یو.پی است. سایر فرایندها از واژه‌هایی مانندِ Work-Product ، Work Units ، Deliverables و مشابه آن استفاده می‌نمایند.

یک دستاورد ممکن است شامل دستاوردهای دیگری باشد. برای مثال، برنامه‌ی تولیدِ نرم‌افزار[۷] شامل چندین برنامه[۸] از جمله برنامه‌ی تقسیم وظایف[۹]، برنامه‌ی فاز[۱۰]، برنامه‌ی سنجش[۱۱]، و برنامه‌های مربوط به تکرارها[۱۲] می‌باشد.

یکی از مسائل مهمی که درباره‌ی دستاوردها مطرح می‌باشد، بحث کنترل نسخه‌های مختلف[۱۳] و مدیریت پیکربندی[۱۴] است.

بر خلاف تصور اولیه، دستاوردها نوعاً به صورت سند نمی‌باشند. بسیار از فرایندها تأکید مفرطی بر تولید اسناد و خصوصاً اسنادِ کاغذی دارند. در مقابل، آر.یو.پی روش سیستماتیک تولید اسنادِ کاغذی را چندان تشویق نمی‌کند. بهترین و کاراترین روش مدیریت دستاوردهای یک پروژه، نگهداری این دستاوردها در داخل ابزارهای مناسب می‌باشد. در موارد ضروری، می‌توانیم به کمک همین ابزارها با گرفتن تصاویر لحظه‌ای[۱۵]، اسناد مناسبی را بلافاصله از روی این دستاوردها ایجاد نماییم.

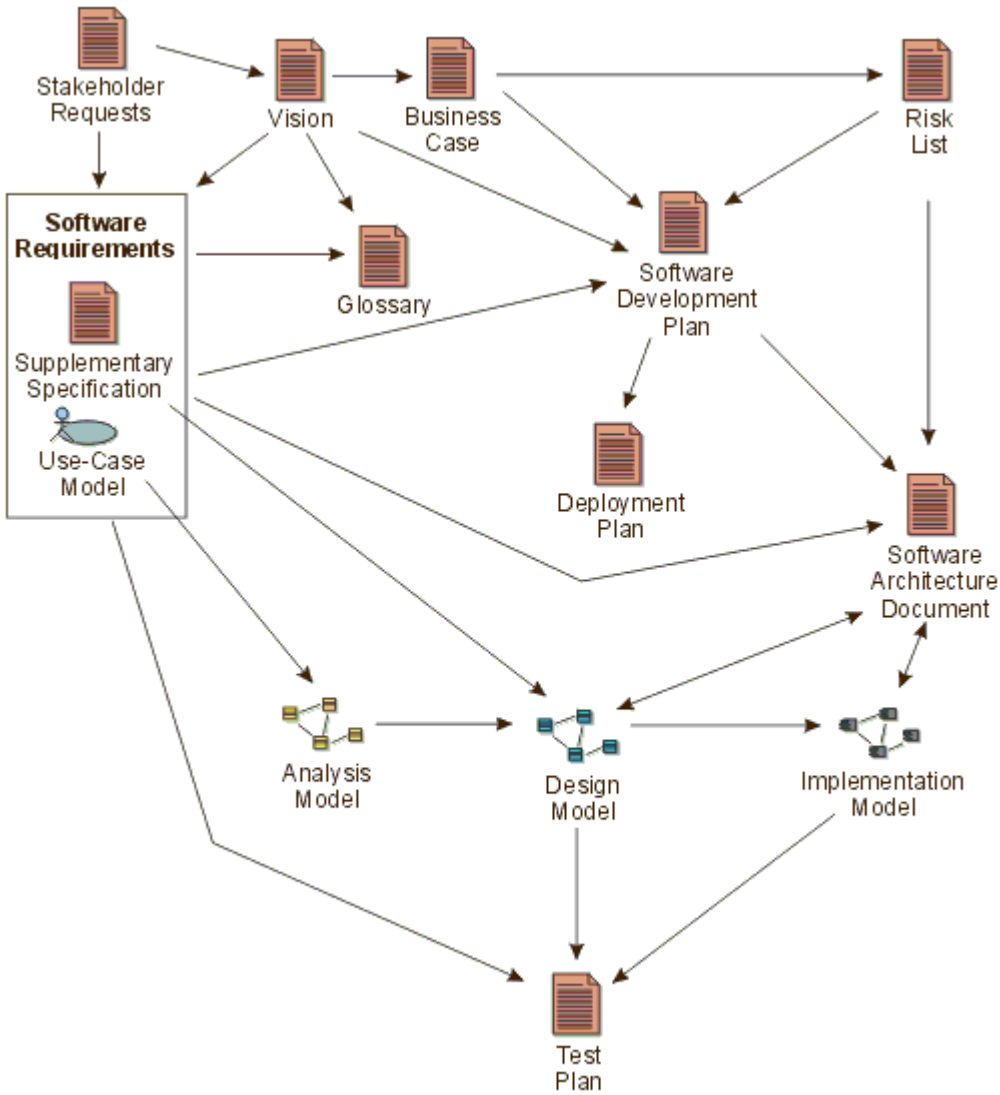
---

[1] - Document
[2] - Vision
[3] - Software Architecture Document
[4] - Source Code
[5] - Executable
[6] - Artifact
[7] - Software Development Plan
[8] - Plan
[9] - Staffing Plan
[10] - Phase Plan
[11] - Measurement Plan
[12] - Iteration Plan
[13] - Version Control
[14] - Configuration Management
[15] - Snapshots

نکته‌ی دیگری که باید در نظر داشته باشیم، این است که بهتر است تا حد امکان سعی شود برای تحویل دستاوردها به ذینفعان نیز از ابزارهای مناسب استفاده کرده و از تحویل در قالبِ کاغذی پرهیز نمـاییم. بـدین ترتیب همواره مطمئن خواهیم بود که اطلاعات به‌روز بوده و بر اساس کارِ واقعی در پروژه هستند. در ضمن در این حالت، هزینه‌های کمتری چه به لحاظ زمانی و چه از نظرِ مالی صرف می‌شود. در شکل ۱۰–۸ ، برخی از مهم‌ترین دستاوردهای مطرح در آر.یو.پی و ارتباط میان آنها نشان داده شده است.

شکل ۱۰– ۸

دستاوردهای عمده در آر.یو.پی و ارتباط میان آن‌ها



چند نمونه از دستاوردهای[1] مهم در یک پروژه‌ی نرم‌افزاری و شکل خاص آنها، عبارتند از:

- یک مدل طراحی در قالب استاندارد یو.اِم.اِل که در ابزار Rational XDE نگهداری می‌شود،
- یک برنامه‌ی پروژه که در ابزار Microsoft Project ذخیره شده است،
- یک گزارش نقص[2] و خطا که در ابزار Rational ClearQuest نگهداری می‌شود،
- بانک اطلاعات نیازمندی‌های یک پروژه که در ابزار Rational Requisite Pro قرار دارد.

با این وجود برخی از دستاوردها باید در قالب متنی باشند، خصوصاً در حالت‌هایی که این دستاوردها بیانگر یک ورودی از بیرون برای پروژه بوده و یا در مواقعی که متن، توصیفِ گویاتر یا رسمی‌تری فراهم می‌آورد. برای نمونه می‌توان به سند چشم‌انداز پروژه و اسناد توصیفِ مواردِ کاربرد اشاره نمود.

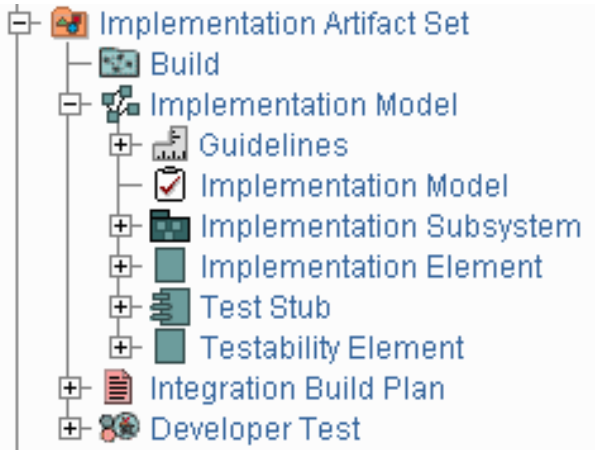
برای تأکید بر این موضوع که هر قطعه‌ی اطلاعاتی باید مسئول مشخصی داشته باشد (چه به لحاظ مسأله‌ی کیفیت آن قطعه‌ی اطلاعات و چه به منظور مدیریت بهتر آن) آر.یو.پی، هر یک از دستاوردها را به عنوان مسئولیت یک نقشِ مشخص و واحد معرفی می‌نماید. بدین ترتیب، در فرایند یک پروژه، هیچ دستاوردی وجود نخواهد داشت، مگر اینکه مسئول مشخص و واحدی داشته باشد. البته، علیرغم اینکه تنها یک شخص مالکیت و مسئولیت یک دستاورد را بر عهده دارد، افراد مختلف می‌توانند از این دستاورد استفاده نمایند و احتمالاً برای به‌روز رسانی باید مجوز لازم را از مالکِ آن دریافت کنند.

دستاوردها در آر.یو.پی با پیشوند artifact مشخص می‌شوند، مانند: «Artifact: Use Case Model»

[1] - Artifact
[2] - Defect

شکل ۱۰-۹

نمونه‌ای از یک دستاورد و اطلاعاتِ مرتبط با آن در محیط آر.یو.پی



گزارش‌ها[1]

مدل‌ها و عناصرِ آن‌ها می‌توانند دارای یک‌سری گزارش باشند. یک گزارش، اطلاعاتی درباره‌ی مدل‌ها و عناصر مربوطه‌شان از یک ابزار استخراج می‌نماید. بر خلاف دستاوردهای معمول، کنترلِ نسخه و مدیریت پیکربندی درباره‌ی گزارش‌ها مطرح نمی‌باشد. هر وقت که لازم شد می‌توانید از روی دستاوردهای موجود، گزارش‌ها را تولید نمایید.

مجموعه‌ی دستاوردها[2]

در آر.یو.پی، دستاوردها در قالب مجموعه‌های اطلاعاتی که معادل دیسیپلین‌های اصلی[3] می‌باشند، سازماندهی شده‌اند. این دسته‌بندی‌ها عبارتند از:

- مجموعه‌ی مدیریت، شامل تمامِ دستاوردهای مرتبط با مدیریتِ پروژه و کسب‌وکار نرم‌افزار، از جمله:
  - دستاوردهای مربوط به برنامه‌ریزی[4]، مانند برنامه‌ی تولید نرم‌افزار[1]، قالبِ کسب‌وکار[2]، قالبِ فرایند[3]، و مانند آن.

---

[1] - Reports
[2] - Artifact Set
[3] - Core Disciplines
[4] - Planning

- دستاوردهای عملیاتی[۴]، مانند توصیفِ نسخه‌های تحویلی[۵]، مستنداتِ استقرار، و داده‌های مربوط به نقایص[۶] و کاستی‌ها

- مجموعه‌ی نیازمندی‌ها، شاملِ تمامِ دستاوردهای مرتبط با تعریفِ نرم‌افزار، از جمله:
  - سندِ چشم‌انداز
  - نیازمندی‌ها در قالبِ نیازهای ذینفعان[۷]، مدلِ مواردِ کاربرد[۸]، و توصیف‌های تکمیلی[۹]
- مجموعه‌ی طراحی، شاملِ تمامِ توصیفی از سیستمِ در دستِ ساخت (یا ساخته شده) در قالب‌های:
  - مدلِ طراحی
  - توصیفِ معماری
- مجموعه‌ی پیاده‌سازی، شاملِ این عناصر:
  - کد و فایل‌های اجرایی
  - مجموعه‌ی فایل‌های داده‌ای[۱۰] یا فایل‌هایی که برای تولید آنها لازم می‌باشند.
- مجموعه‌ی استقرار[۱۱]، شاملِ تمامِ اطلاعاتِ تحویلی، از جمله:
  - الزامات نصب (اجزاء و اِمکاناتِ لازم)
  - مستنداتِ کاربران
  - مواد و مطالبِ آموزشی

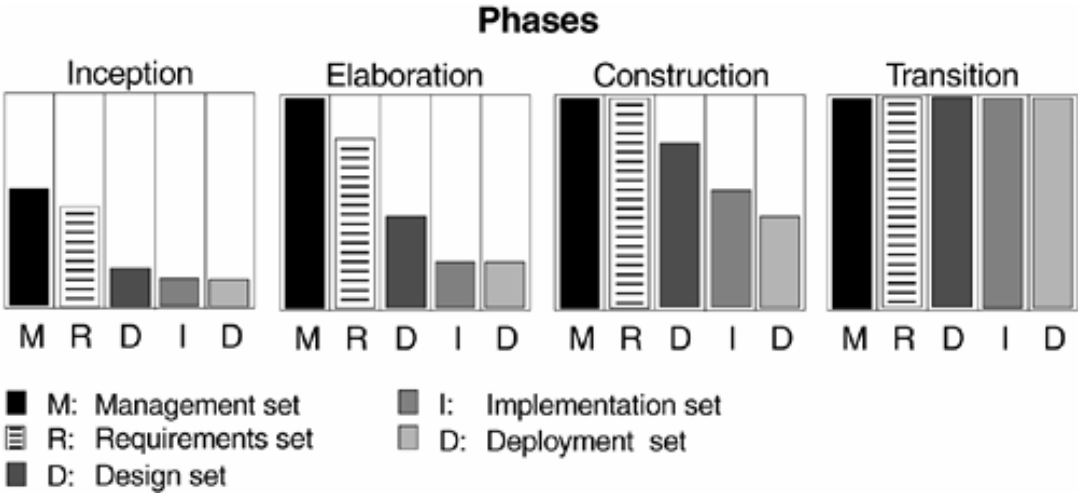
در یک رویکردِ تکرارشونده، دستاوردهای مختلف به طور یکباره در طی یک فاز کامل نشده و اصطلاحاً بسته نمی‌شوند؛ بلکه در عوض، مجموعه‌های اطلاعاتی در طولِ چرخه‌ی تولید، تکامل می‌یابند. در شکل ۱۰-۱۰ ، نمایی از روندِ تکامل دستاوردها در طول چرخه‌ی تولید نشان داده شده است.

---

[1] - Software Development Plan or SDP
[2] - Business Case
[3] - Development Case
[4] - Operational
[5] - Release Description
[6] - Defect Data
[7] - Stakeholders' Needs
[8] - Use-Case Model
[9] - Supplementary Specification
[10] - Data Files
[11] - Deployment Set

شکل ۱۰-۱۰

روندِ تکامل و رشدِ مجموعه‌های اطلاعاتی (مجموعه دستاوردها)



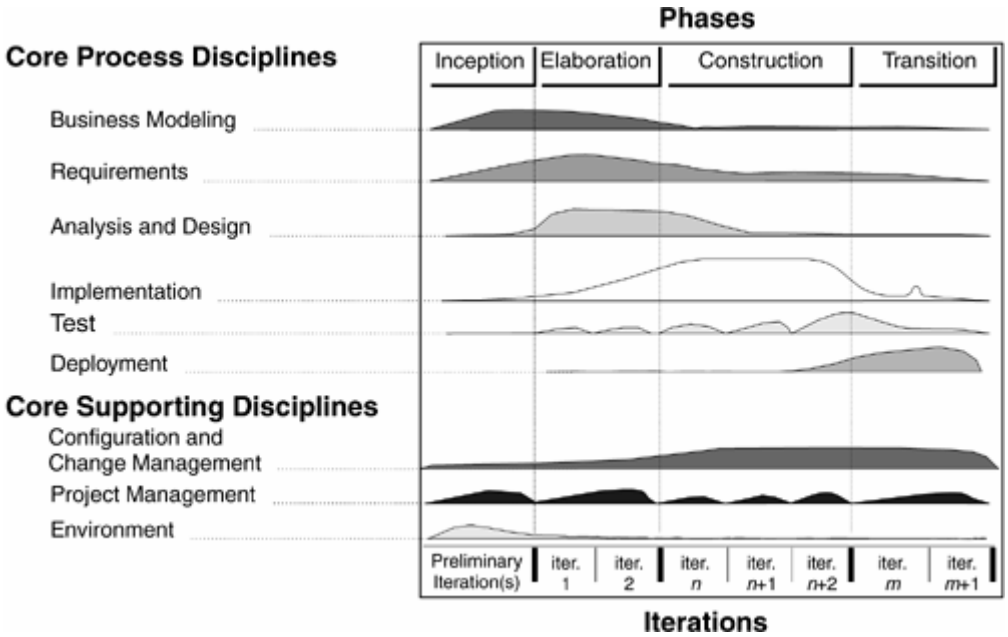
دیسیپلین‌ها[1]

دیسیپلین‌ها، ظرف‌هایی[2] هستند برای سازمان‌دهیِ فعالیت‌های فرایند. در فرایند آر.یو.پی، به طور پیش فرض، تعدادِ نُه دیسیپلین وجود دارد. این دیسیپلین‌ها بیانگر دسته‌بندی مجموعه‌ی نقش‌ها و فعالیت‌ها در قالبِ گروه‌هایی است که دارای نگرانی، دغدغه‌ها، و یا دیدگاهِ مشترکی می‌باشند. نُه دیسیپلینِ ارائه شده به دو دسته تقسیم می‌شوند، یک دسته شامل شش دیسیپلین فنی و دسته‌ی دیگر، شامل سه دیسیپلین پشتیبانی.

[1] - Disciplines
[2] - Containers

شکل ۱۰-۱۱

دیسیپلین‌های پیش‌فرض در فرایندِ آر.یو.پی



دیسیپلین‌های فنی[۱] عبارتند از:

- دیسیپلینِ مدل‌سازی سازمان[۲]
- دیسیپلینِ نیازمندی‌ها[۳]
- دیسیپلینِ تحلیل و طراحی[۴]
- دیسیپلینِ پیاده‌سازی[۵]
- دیسیپلینِ تست[۶]
- دیسیپلینِ استقرار[۷]

[1] - Technical
[2] - Business Modeling
[3] - Requirements
[4] - Analysis and Design
[5] - Implementation
[6] - Test
[7] - Deployment

دیسیپلین‌های پشتیبان[1] عبارتند از:

- دیسیپلینِ مدیریت پروژه[2]
- دیسیپلینِ مدیریت پیکربندی و تغییرات[3]
- دیسیپلینِ محیط[4]

نام‌گذاری مجموعه‌ی نقش‌ها و فعالیت‌های یکسان در قالبِ مفهوم دیسیپلین نیز جالب توجه است. می‌دانیم که مفهومِ دیسیپلین به معنای نظم و انضباط است. در واقع وجه تسمیه گروه‌بندی‌های یادشده، این است که مجموعه‌ی نقش‌ها و فعالیت‌هایی که مثلاً در دیسیپلینِ مدیریت پروژه، گروه‌بندی می‌شوند، دارای نوعی طرزِ نگرش، دیدگاه، انضباطِ کاری، و هدفِ مشترک می‌باشند که با دیدگاه و اهداف مجموعه‌ی نقش‌ها و فعالیت‌های سایر دیسیپلین‌ها، تفاوتِ اساسی دارد.

در طی فصل‌های آتی، یعنی فصل‌های ۱۱ تا ۱۹، هر یک از دیسیپلین‌های تعریف شده در آر.یو.پی را مختصراً تشریح خواهیم نمود.

---

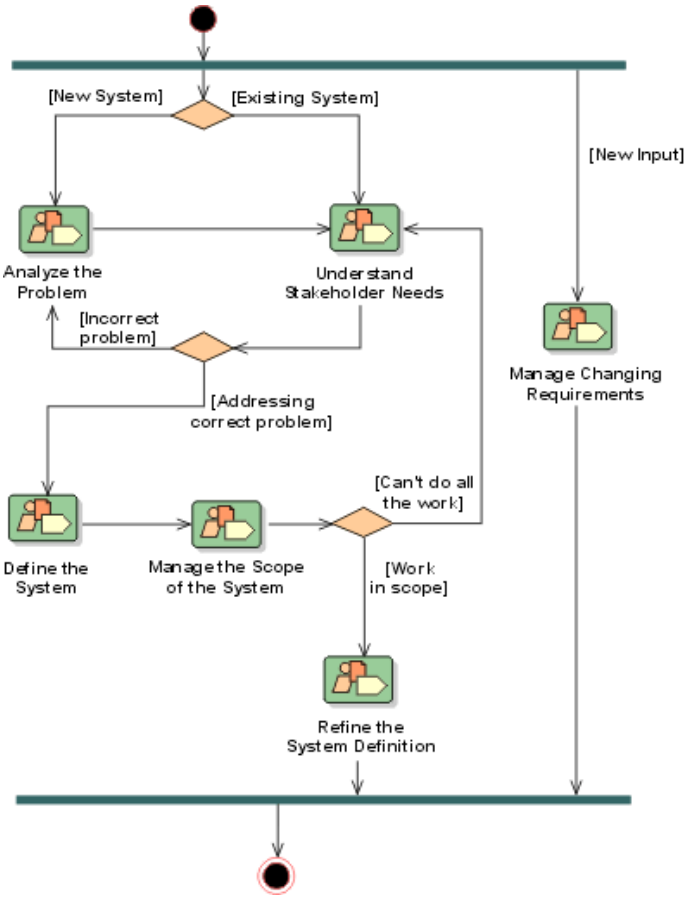
[1] - Supporting
[2] - Project Management
[3] - Configuration and Change Management
[4] - Environment

جریان‌کار[1]

جریان‌کار، یک مدلِ گرافیکی (بصری[2]) است برای بازنمایی جریانِ منطقیِ و توالی مجموعه فعالیت‌هـای هـر دیسیپلین که منجر به تولید یک نتیجه‌ی ارزشمند می‌شوند. این مدل در زبانِ مدل‌سازی یو.اِم.اِل[3] و با استفاده از دیاگرامِ فعالیت[4]، بیان شده است. در شکل ۱۰-۱۲، نمونه‌ای از یک جریان‌کار نشان داده شده است. یکی از کاربردهای مهمِ این مدل، برنامه‌ریزی تکرارها می‌باشد.

شکل ۱۰-۱۲

نمونه‌ای از یک مدلِ جریان‌کار در آر.یو.پی



[1] - Workflow
[2] - Visual
[3] - UML
[4] - Activity Diagram

باید توجه داشت که نشان دادنِ همه‌ی فعالیت‌های یک دیسیپلین در قالبِ یک دیاگرام، نه تنها به درکِ بهتر آن کمکی نمی‌کند، بلکه بر پیچیدگی مفاهیم نیز می‌افزاید. بنابراین، به جای نشان‌دادنِ فعالیت‌ها در جریان‌کارِ معادلِ یک دیسیپلین، فعالیت‌هایی را که به نحوی با هم در ارتباط هستند، در مجموعه‌ای تحتِ عنوان جزئیاتِ جریان‌کار دسته‌بندی می‌نماییم.

مفهومِ جریانِ کار در آر.یو.پی دارای سه مصداق متفاوت می‌باشد که عبارتند از:

- جریان‌های‌کارِ اصلی[۱]، متناظر با هر دیسیپلین
- جزئیاتِ جریانِ کار[۲]
- برنامه‌های تکرار[۳]

دیگر عناصرِ فرایند

نقش‌ها، فعالیت‌ها، جریان‌های کار، و دستاوردها که در قالبِ دیسیپلین‌ها سازمان‌دهی می‌شوند، زیربنایِ ساختارِ استاتیک یا محتوایی آر.یو.پی را تشکیل می‌دهند. برخی عناصر دیگر نیز در این ساختار وجود دارند که بر غِنای آن افزوده‌اند. این عناصر عبارتند از:

- راهنمایی‌ها[۴]
- قالب‌ها (الگوها)[۵]
- راهنمایِ آموزش‌دهنده‌ی ابزار[۶]
- مفاهیم[۷]

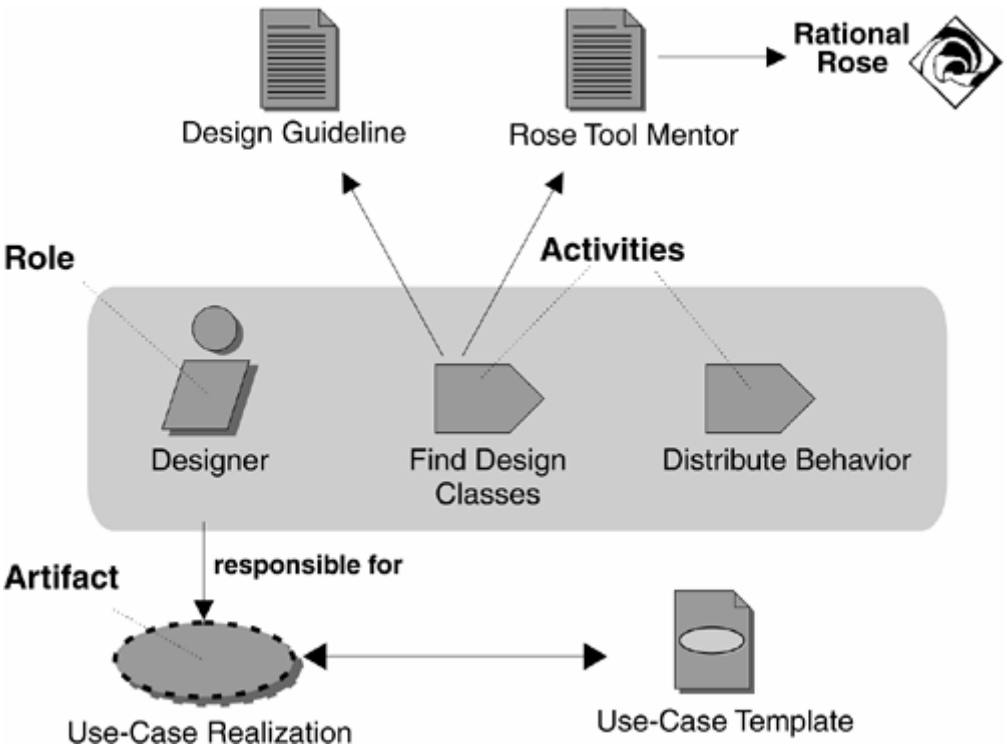
شکل ۱۰-۱۳ ، این عناصر و ارتباطشان با سایر عناصرِ محتوایی آر.یو.پی را نشان می‌دهد.

---

[1] - Core Workflows
[2] - Workflow Details
[3] - Iteration Plans
[4] - Guidelines
[5] - Templates
[6] - Tool Mentors
[7] - Concepts

شکل ۱۰-۱۳

الگوها، آموزشِ ابزار، و راهنمایی‌ها در ساختارِ استاتیکِ آر.یو.پی



به عنوانِ نکته‌ی پایانیِ این فصل، جالب است بدانید که ساختارِ استاتیکِ آر.یو.پی به وسیله‌ی گسترشِ[1] خاصی از زبانِ یو.اِم.اِل، تحتِ عنوانِ SPEM[2] (متعلق به کنسرسیومِ OMG)، مدل‌سازی شده است. بدین ترتیب، نگهداری و توسعه‌ی آر.یو.پی، یک کار مهندسی و سیستماتیک می‌باشد.

[1] - Extension
[2] - Software Process Engineering Metamodel

چکیده‌ی فصل

در این فصل با ساختارِ استاتیکِ فرایندِ آر.یو.پی آشنا شدیم. مهم‌ترین نکات ذکر شده در این فصل، عبارتند از:

- عناصر پایه‌ای در ساختار آر.یو.پی عبارتند از: نقش‌ها، فعالیت‌ها، و دستاوردها.
- دیسیپلین‌ها، گروه‌بندی طبیعی فعالیت‌ها، نقش‌ها، و دستاوردهای فرایند می‌باشد.
- جریانِ‌کار، مجموعه‌ی فعالیت‌ها، نقش‌ها، و دستاوردها را در قالب توالی‌هایی که منجر به تولید نتایج ارزشمند می‌شوند، به‌هم مرتبط می‌سازند.
- راهنمایی‌ها، قالب‌ها (الگوها)، و راهنمای آموزش‌دهنده‌ی ابزارها، مکمل توصیف فرایند بوده و راهنمایی‌هایی مفیدی را فراهم می‌نمایند.
- آر.یو.پی به وسیله‌ی گسترشی از زبانِ مدل‌سازی استانداردِ یو.اِم.اِل که SPEM نامیده می‌شود، مدل‌سازی شده است.

پرسش‌هایی برای تحقیق و مطالعه‌ی بیشتر

۱. درباره‌ی نقش تحلیل‌گرِ سیستم تحقیق‌نمایید.

۲. درباره‌ی چگونگی تغییر دیسیپلین‌ها در پروژه‌های غیر نرم‌افزاری تحقیق نمایید. برای مثال پروژه‌ی ساخت یک اتوموبیل یا نوشتن یک مقاله را در نظر بگیرید.

۳. مجموعه‌ی نقش‌های موجود در آر.یو.پی را بر حسب ضرورت یا عدمِ ضرورتِ وارد شدن‌شان در جزئیات به دو دسته‌ی تقسیم نمایید.

۴. درباره‌ی جزئیاتِ استانداردِ مدل‌سازی فرایند یعنی SPEM تحقیق نمایید.

منابع و مراجع

[1]. Ivar Jacobson, Grady Booch, James Rumbaugh, (1999). *The Unified Software Development Process*, Reading, MA: Addison-Wesley

[2]. Philippe Kruchten, (2003). *The Rational Unified Process: An Introduction*, Reading, MA: Addison-Wesley.

[3]. Per Kroll, Philippe Kruchten, (2003). *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*, Reading, MA: Addison-Wesley.

[4]. Rational Software Corporation. (2003) "Rational Unified Process 2003.06," Available at: http://www.rational.com/

[5]. OMG (2001). Object Management Group. *Software Process Engineering Metamodel (SPEM)*. OMG, doc ad/01-03-08, April 2, 2001. Available at: http://cgi.omg.org/cgi-bin/doc?ad/01-03-08.

[6]. Software Academy. (2006) "Unified Process Knowledge Base," Available at: http://www.unifiedProcess.info/

[7]. Scott W. Ambler, (2000). *The Unified Process Inception Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[8]. Scott W. Ambler, (2000). *The Unified Process Elaboration Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[9]. Scott W. Ambler, (2000). *The Unified Process Construction Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[10]. Scott W. Ambler and Larry L. Constantine, (2001). *The Unified Process Transition Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

فصل یازدهم

دیسیپلینِ مدیریتِ پروژه

مهم‌ترین موضوعات مورد بررسی در این فصل، عبارتند از:

- اهداف دیسیپلینِ مدیریت پروژه در آر.یو.پی
- فعالیت‌ها، دستاوردها، و نقش‌های کلیدی
- مفهوم و اهمیت ریسک
- انواع برنامه‌ریزی پروژه در آر.یو.پی

۱۱ دیسیپلینِ مدیریت پروژه

در این فصل، یکی از دیسیپلین‌های مهم آر.یو.پی، یعنی دیسیپلین مدیریت پروژه را بررسی خواهیم نمود. مهم‌ترین مفاهیمی را که در این فصل مورد بررسی قرار می‌دهیم، مفاهیم ریسک و مقیاس‌های میزان پیشرفت است. این دو مفهوم، عناصرِ کلیدی در برنامه‌ریزی و کنترلِ رویکرد تکرارشونده[1] می‌باشند. در این فصل چگونگی برنامه‌ریزی یک تکـرار[2] و نیز چگونگی تصمیم‌گیری درباره‌ی مدت زمان آن و نیز فعالیت‌هـای ضـروری در یـک تکـرار را مختـصراً بررسی خواهیم نمود.

شکل ۱۱-۱

نمایی از حجمِ فعالیت‌های دیسیپلینِ مدیریتِ پروژه در طولِ چرخه‌ی تولید

[1] - Iterative Development
[2] - Iteration

هدف

مدیریت یک پروژه‌ی نرم‌افزاری[۱] عبارتست از هنر برقراری توازن[۲] میان اهداف رقابتی[۳]، مدیریتِ ریسک‌ها، و غلبه بر محدودیت‌ها، به منظور تحویلِ فراورده‌ای[۴] نرم‌افزاری که مناسب و درخورِ نیازهای مشتریان و کاربران نهایی باشد. این حقیقت که نسبتاً تعداد کمی از پروژه‌های نرم‌افزاری با موفقیت کامل خاتمه می‌یابند، بیانگرِ سختی فعالیتِ مدیران پروژه‌های نرم‌افزاری می‌باشد.

در آر.یو.پی، رهنمودهایی[۵] برای انجام آسان‌تر و موفقیت‌آمیز فعالیت‌های مرتبط با دیسیپلین مدیریتِ پروژه فراهم شده است. البته این بدان معنا نیست که به این ترتیب موفقیت پروژه تضمین شود، بلکه رویکردِ مناسبی است که به طور محسوس، احتمال موفقیت در تحویلِ یک فراورده‌ی نرم‌افزاریِ دارای کیفیتِ مطلوب را افزایش می‌دهد.

سه هدفِ عمده‌ی دیسیپلین مدیریت پروژه عبارتست از:

- فراهم‌آوری چارچوبی[۶] برای مدیریت پروژه‌های نرم‌افزاری
- ارائه‌ی رهنمودهای عملی[۷] برای برنامه‌ریزی[۸]، تخصیص نیروها[۹]، اجرا[۱۰]، و نظارت[۱۱] بر پروژه‌ها
- فراهم نمودن چارچوبی برای مدیریت ریسک[۱۲]

---

[1] - Software Project Management
[2] - Balance
[3] - Competing Objectives
[4] - Product
[5] - Guidelines
[6] - Framework
[7] - Practical Guidelines
[8] - Planning
[9] - Staffing
[10] - Executing
[11] - Monitoring
[12] - Risk Management

البته، باید توجه داشت که دیسیپلینِ مدیریتِ پروژه در آر.یو.پی، همه‌ی جنبه‌های مدیریت پروژه را نمی‌پوشاند. برای مثال، مسائل و ملاحظات زیر در حیطه‌ی این دیسیپلین در آر.یو.پی، قرار نمی‌گیرند:

- مدیریت منابع انسانی: استخدام، آموزش، هماهنگی، و رهبری
- مدیریت مالی: تعریف و تخصیص[۱] بودجه
- مدیریت قراردادها[۲] با مشتری‌ها و کارپردازها[۳]

شایان ذکر است که مباحث تکمیلی مرتبط با مدیریت پروژه را می‌توان از طریق انستیتو مدیریت پروژه[۴] و مرجعی تحت عنوانِ ساختار و بدنه‌ی دانشِ مدیریت پروژه[۵] که متعلق به مؤسسه‌ی مذکور است، پیگیری نمود. آر.یو.پی، در بسیاری از موضوعات مرتبط با مدیریت عمومی پروژه‌ها، به این مرجع اشاره دارد.

تمرکز اصلی دیسیپلین مدیریت پروژه در آر.یو.پی بر موضوع مدیریتِ فرایند مبتنی بر رویکرد تکرارشونده[۶] می‌باشد. در این میان، مهم‌ترین مباحث مطرح در رابطه با این دیسیپلین، عبارتست از:

- برنامه‌ریزی یک پروژه به روش تکرارشونده در طول چرخه‌ی تولید و نیز برنامه‌ریزی هر نمونه از تکرارهای[۷] درون فازها
- مدیریت ریسک
- نظارت بر پیشرفتِ[۸] یک پروژه‌ی برنامه‌ریزی شده با رویکرد تکرارشونده و سنجش میزان پیشرفت به صورت کمّی و واقع‌بینانه

---

[1] - Allocating
[2] - Contracts
[3] - Suppliers
[4] - Project Management Institute (PMI)
[5] - Project Management Body of Knowledge (PMBOK)
[6] - Iterative Process
[7] - Iteration
[8] - Progress

برنامه‌ریزی یک پروژه بر اساس رویکردِ تکرارشونده

معمولاً توجیه‌کردن مدیران پروژه نسبت به مزیت‌های رویکرد تکرارشونده، کار چندان مشکلی نیست. اما هنگامی که یک مدیر پروژه برای اولین بار قصد استفاده از این رویکرد را دارد، اغلب با سردرگمی مُواجه می‌شود. به طور کلی، مدیریت رویکردِ تکرار شونده، پیچیده‌تر و مستلزم دیدگاه متفاوتی نسبت به رویکردِ سنتی برنامه‌ریزی می‌باشد. در بکارگیری روش تکرارشونده، مدیران پروژه با سؤالات جدیدی مواجه می‌شوند، از جمله:

- در هر یک از فازهای پروژه چند تکرار باید درنظر گرفته شود؟
- طول زمانی این تکرارها چه اندازه می‌باشد؟
- چگونه می‌توانیم اهداف و محتوای هر یک از تکرارهای را تعیین نماییم؟
- چگونه می‌توانیم میزان پیشرفت و وضعیت پروژه در هر تکرار را پیگیری و بررسی نماییم؟

برخی از اهدافِ برنامه‌ریزی پروژه عبارتند از:

- تخصیص وظایف و مسئولیت‌های تیمی از افراد در طول زمان
- نظارت بر پیشرفت نسبت به برنامه و شناسایی مشکلات بالقوه با گسترده‌تر شدن پروژه

در مقوله‌ی برنامه‌ریزی، ملاحظات مرتبط با منابع غیر انسانی مانند بودجه، تجهیزات، و تسهیلات نیز مورد توجه قرار می‌گیرد. اما از آن جایی که رویکردِ تکرارشونده تأثیر چندانی بر برنامه‌ریزی منابع مذکور ندارد، این مباحث در آر.یو.پی مطرح نشده است.

دو سطحِ برنامه‌ریزی

برخلاف برنامه‌ی بنا کردنِ یک برجِ ساختمانی، که می‌توان (و در اغلب موارد باید) پیش از شروع پروژه، برنامه‌ای با جزئیات دقیق از ساخت و ساز آن ارائه نمود، ارائه‌ی برنامه‌ای با جزئیات دقیق و حتی تلاش برای انجام این کار در ابتدای یک پروژه‌ی نرم‌افزاری (و نیز بسیاری از پروژه‌های تولید سیستم)، در اکثر موارد غیرممکن و محکوم به شکست می‌باشد.

در بسیاری از موارد، شاهد پر شدن در و دیوار شرکت‌ها و محل کار تیم‌های تولید نرم‌افزار با طرح‌ها و برنامه‌های تفصیلی هستیم. واقعیت اینست که این برنامه‌ها تنها در شرایطی که نیازمندی‌ها و معماری تثبیت شده باشند، واقع‌بینانه خواهند بود و در غیر این صورت، جز داشتن اطلاعاتی اشتباه و گمراه‌کننده که حتی می‌تواند نتایج جبران‌ناپذیری در پی داشته باشد، نتیجه‌ای در بر نخواهند داشت.

چگونه می‌توان در یک پروژه، آقای فلانی را وادار نمایید که فلان مؤلفه را در هفته‌ی بیستمِ پروژه پیاده‌سازی نماید، در حالیکه حتی نسبت به وجود یا عدم وجود این مؤلفه، اطمینانی وجود ندارید؟! بنابراین، داشتن یک برنامه‌ی تفصیلی در ابتدای پروژه، تنها در صنایعی که ریسک‌های آن‌ها به خوبی شناخته‌شده و حتی کمّ و کیف فعالیت‌ها در آن مثلاً به دلیل قوانین فیزیکی از قبل مشخص و به نوعی استاندارد شده است، کارایی دارد و البته، این موضوع شامل دنیای نرم‌افزار نمی‌شود.

بنا به توصیه‌ی آر.یو.پی (یا به عبارت دیگر بنا به تجربه‌ی موفق در صنعت نرم‌افزار)، در یک فرایند نوین مهندسی نرم‌افزار که بر مبنای رویکرد تکرارشونده بنا نهاده شده است، دو نوع برنامه بکار گرفته می‌شود:

- یک برنامه‌ی به اصطلاح درشت‌دانه[1] که برنامه‌ی فاز یا همان برنامه‌ی پروژه نام دارد،
- یک‌سری برنامه‌ی به اصطلاح ریزدانه[2] که برنامه‌ی تکرارهای مختلف می‌باشد.

در ادامه، هر یک از این دو نوع برنامه را مختصراً تشریح می‌نماییم.

---

[1] - Coarse-grained
[2] - Fine-grained

برنامه‌ی فازها[۱] یا برنامه‌ی پروژه[۲]

برنامه‌ی فاز، برنامه‌ای است درشت‌دانه[۳] که برای هر پروژه، تنها یک نمونه از آن تعریف می‌گردد. در این برنامه، پروژه در قالب یک چرخه‌ی تولید[۴] (و در برخی موارد مشتمل بر چرخه‌های متعاقبِ آن) نشان داده می‌شود. مهم‌ترین ملاحظات مرتبط با این برنامه عبارتند از:

- تاریخ و زمان رسیدن به هریک از گام‌های اصلی[۵] یا نقاطِ تصمیم‌گیری کلیدی:
  - اولین نقطه‌ی تصمیم‌گیری که در پایان فازِ آغازین[۶] (شناخت) و تحت عنوان LCO تعریف می‌شود با رسیدن به این نقطه، ابعاد و محدوده‌ی پروژه باید تعیین شده باشد.
  - دومین نقطه‌ی تصمیم‌گیری که در پایان فازِ تشریح[۷] (معماری) و تحت عنوان LCA از آن یاد می‌شود. در این نقطه، باید پابت شود که معماری و نیز نیازمندی‌های سیستم تثبیت شده است.
  - نقطه‌ی تصمیم‌گیری سوم، در پایان فازِ ساخت[۸] که تحت عنوان IOC و برای ارائه‌ی نسخه بِتای سیستم برنامه‌ریزی می‌گردد.
  - نقطه‌ی تصمیم‌گیری چهارم، در پایان فازِ اِنتقال[۹] که تحت عنوان PR مطرح می‌باشد و نقطه‌ای است که در آن چرخه‌ی تولید[۱۰] خاتمه می‌یابد.
- پروفایلِ افراد: اینکه چه نیروهایی در طول بازه‌ی زمانی پروژه در آن مشغول فعالیت خواهند بود.
- تاریخ گام‌های کوچک[۱۱] میانی یا نقاطِ آشکارشدن ملاحظاتِ فنی در هر فاز: تعیین زمان پایان هر تکرار و در صورت امکان اهداف اولیه‌ی آن.

---

[1] - Phase Plan
[2] - Project Plan
[3] - Coarse-grained
[4] - Development-Cycle
[5] - Major Milestone
[6] - Inception
[7] - Elaboration
[8] - Construction
[9] - Transition
[10] - Development Process
[11] - Minor Milestone

برنامه‌ی فاز یا برنامه‌ی پروژه، در همان اوایل فازِ آغازین (شناخت)، ایجاد شده و در ادامه‌ی فرایند و در صورت لزوم، به‌روز رسانی می‌شود. این برنامه معمولاً از حدود چند صفحه تجاوز نمی‌نماید.

جدول ۱۱-۱

وزن نسبی زمان و حجمِ تلاش‌ها در فازهای مختلفِ یک پروژه

فاز	زمان‌بندی	تلاش
آغازین (شناخت)	٪ ۱۰	٪ ۵
تشریح (معماری)	٪ ۳۰	٪ ۲۰
ساخت	٪ ۵۰	٪ ۶۵
اِنتقال	٪ ۱۰	٪ ۱۰

برنامه‌ی یک تکرار[1]

یک برنامه‌ی تکرار، برنامه‌ای است ریزدانه[2] که یک نمونه از آن در هر یک از تکرارهای چرخه‌ی تولید وجود دارد. به طور کلی مدیر پروژه در هر زمان از یک پروژه با دو برنامه‌ی تکرار فعّال، سر و کار دارد:

- برنامه‌ی تکرار جاری (برنامه‌ای که فعالیت‌های جاری در قالب آن تعریف شده است)
- برنامه‌ی تکرار بعدی (برنامه‌ای که تقریباً از اواسط تکرار فعلی و برای برنامه‌ریزی فعالیت‌های مورد نیاز در تکرار بعدی ارائه می‌گردد)

برنامه‌ی یک تکرار با همان تکنیک‌ها و ابزارهای مورد استفاده در برنامه‌ریزی سنتی (مانند گانت‌چارت و مانند آن) ایجاد شده و در قالبِ آن، فعالیت‌ها و ظایف افراد و تیم‌ها در بازه‌ی زمانی یک تکرار، تعیین می‌شود.

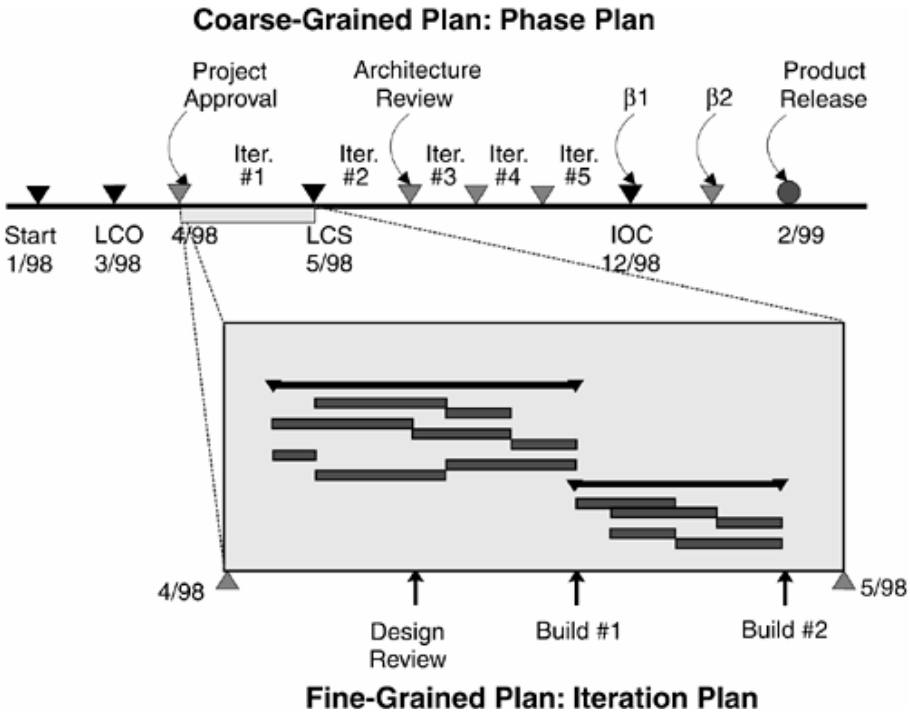
[1] - Iteration Plan
[2] - Fine-grained Plan

در این برنامه، زمان‌های مهمی مانند نسخه‌های میانی[1] اصلی، زمان رسیدن یک مؤلفه از یک بخش دیگر، و نقاط بازبینی[2] مشخص می‌شود.

همانگونه که در شکل ۱۱-۲ نشان داده شده است، برنامه‌ی یک تکرار را می‌توان همانند یک ذره‌بین در طول برنامه‌ی فاز (برنامه‌ی پروژه) تصوّر نمود.

شکل ۱۱-۲

برنامه‌ی پروژه و برنامه‌ی یک تکرار



از آنجایی که فرایندِ مبتنی بر رویکردِ تکرارشونده، فرایندی است پویا و ابـزاری مناسـب بـرای تطبیـق و سازگاری مناسب تغییرات فراهم می‌آورد، صرف هزینه‌های زیاد برای تـشریح جزئیـاتی فراتـر از افـق زمـانی برنامه‌ریزی مطلوب نمی‌باشد. تجربه نشان داده است که حداقل در دنیای پرآشوب و پویای امـروزیِ صـنعتِ نرم‌افزار، برنامه‌هایی که بازه‌ی زمانی طولانی‌تری را بپوشانند، با مشکلات عمده‌ای در نگهداری مواجه شده و

[1] - Build
[2] - Review

عملاً موانع بسیاری برای فعالیت مؤثر تیم تولیدکننده ایجاد می‌نمایند. معمولاً، چنین برنامه‌هایی خیلی زود کنار گذاشته شده و به نوعی حالت صوری و مصنوعی به خود می‌گیرند.

از دیگر نکات مهم مرتبط با برنامه‌ی یک تکرار در آر.یو.پی، می‌توان به موارد زیر اشاره داشت:

- اساس برنامه‌ریزی فعالیت‌های مختلف، مواردِ کاربرد[1] می‌باشد.
- معیارهای ارزیابی تکرار نیز در برنامه‌ی تکرار مشخص می‌شود.
- در رویکردِ تکرارشونده، اِمکان یادگیری و بهبود مستمر برنامه‌ریزی در تکرارهای مختلف وجود دارد.

طولِ زمانی یک تکرار در پروژه‌های مختلف و شرایط مختلف، متفاوت می‌باشد. در جدول ۱۱-۲ ، طولِ زمانی چند نوع پروژه‌ی مختلف نشان داده شده است.

جدول ۱۱-۲

طولِ زمانی یک تکرار در پروژه‌های با اندازه‌های مختلف

پروژه‌های دارای رویکردِ تکرارشونده		
متوسطِ طولِ زمانی تکرارها	تعدادِ افراد درگیر	تعداد خطوطِ کُد
۲ هفته	۴	۵ هزار
۱ ماه	۱۰	۲۰ هزار
۳ ماه	۴۰	۱۰۰ هزار
۶ ماه	۱۵۰	یک میلیون

[1] - Use-Case

مفهومِ ریسک[1]

امروز در شرایطی به سر می‌بریم که به اعتقاد بسیاری از متخصصان و کارشناسانِ دنیای نرم‌افزار، دیگر هیچ پروژه‌ی بدون ریسکی نمانده است! ریسک و مدیریت آن یکی از مهم‌ترین مفاهیم مطرح در دیسیپلین مدیریت پروژه می‌باشد. البته به اعتقاد آر.یو.پی و بر اساس تجارب موفق در دنیای نرم‌افزار، ریسک و مدیریت آن، به نوعی، وظیفه‌ی تک‌تکِ اعضای تیمِ تولیدکننده‌ی نرم‌افزار می‌باشد و تنها مرتبط با مدیرِ پروژه نیست. مفهوم مدیریت ریسک به طور عمده در رابطه با مدیریتِ جنبه‌ها و شرایط ناشناخته مطرح می‌شود.

ریسک چیست؟

ریسک به صورت کلیه‌ی عوامل و شرایطی که می‌توانند مانع دستیابی به موفقیت شوند، تعریف می‌گردد. بسیاری از تصمیمات در چرخه‌ی عمر یک فرایندِ مبتنی بر رویکرد تکرارشونده، بر مبنای ریسک اتخاذ می‌شود.

معمولاً ریسک‌ها بر اساس ماهیت و منشاء اصلی‌شان به دو دسته تقسیم می‌شوند:

- ریسک‌های مستقیم: ریسک‌هایی که در یک پروژه، امکان کنترل زیادی بر روی آنها داریم.
- ریسک‌های غیر مستقیم: ریسک‌هایی که کنترل زیادی بر آنها نداریم.

دو ویژگی کلیدی یک ریسک در اولویت‌دهی به آن در نظر گرفته می‌شود. این دو ویژگی که بخش مهمی از پروفایل یک ریسک را تشکیل می‌دهند، عبارتند از:

- احتمال وقوع یک ریسک
- شدّت اثر آن بر پروژه

[1] - Risk

استراتژی‌های رویارویی با ریسک

نکته‌ی کلیدی در مدیریت ریسک این است که نباید دست روی دست گذاشته و به انتظار بنشینیم تا اینکه ریسک خود را نشان داده و عملاً به یک مشکل تبدیل شود، بلکه باید از قبل راهکار و استراتژی مناسبی برای آن بیاندیشیم. استراتژی‌های متداول در مدیریت ریسک عبارتند از:

- اجتناب از ریسک[۱] : در حالتی که تشخیص می‌دهیم که ریسک، اثر چندانی بر پروژه ندارد.
- انتقالِ ریسک[۲] : سازماندهی مجدد پروژه به‌گونه‌ای که شخص یا چیز دیگری، ریسک را برعهده بگیرد.
- پذیرش ریسک[۳] : تصمیم‌گیری بر رفع ریسک.

نقش‌ها[۴] و دستاوردها[۵]

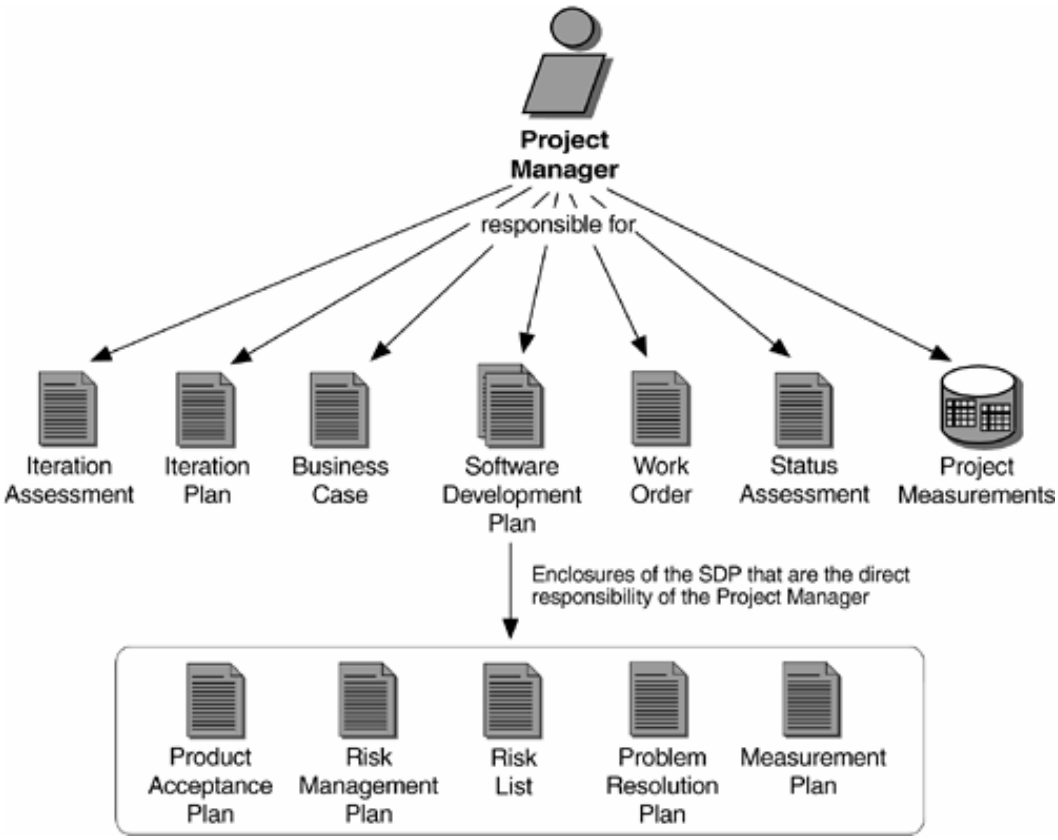
در شکل ۱۱-۳ ، مهم‌ترین نقش تعریف شده در دیسیپلین مدیریت پروژه، یعنی نقش مدیرِ پروژه[۶] و دستاوردهایی را که این نقش مسئولیتِ آنها را برعهده دارد، نشان داده شده است. نقش مهم دیگری که در این دیسیپلین تعریف شده است، نقش بازبینی‌کننده‌ی پروژه[۷] می‌باشد. این نقش، مسئولیت ارزیابی دستاوردهای مربوط به برنامه‌ریزی پروژه[۸]، دستاوردهای مرتبط با ارزیابی پروژه[۹]، و نیز ارزیابی نقاط بازبینی اصلی در پروژه را برعهده دارد.

---

[1] - Risk Avoidance
[2] - Risk Transfer
[3] - Risk Acceptance
[4] - Roles
[5] - Artifacts
[6] - Project Manager
[7] - Project Reviewer
[8] - Project Planning Artifacts
[9] - Project Assessment Artifacts

شکل ۱۱-۳

نقش مدیرِ پروژه و دستاوردهای مرتبط با آن



مهم‌ترین دستاوردهای مرتبط با دیسیپلینِ مدیریت پروژه، عبارتند از:

- برنامه‌ی تولید نرم‌افزار[۱] که خود شامل چندین دستاورد دیگر می‌باشد:
  - برنامه‌ی پذیرش محصول[۲]
  - برنامه‌ی مدیریت ریسک[۳] و ریسک‌لیست[۴]
  - برنامه‌ی حل و فصل مشکل[۵]
  - برنامه‌ی سنجش[۶]
- قالب ملاحظات کسب‌وکار[۷]

---

[1] - Software Development Plan
[2] - Product Acceptance Plan
[3] - Risk Management Plan
[4] - Risk List
[5] - Problem Resolution Plan
[6] - Measurement Plan
[7] - Business Case

- برنامه‌های تکرارهای مختلف[۱]
- ارزیابی تکرار[۲]
- ارزیابی (دوره‌ای) وضعیت[۳]
- سفارش کار[۴]
- بانک اطلاعاتی داده‌های حاصل از اندازه‌گیری پروژه[۵]

سایر برنامه‌ها و طرح‌ها نیز بخشی از برنامه‌ی تولیدِ نرم‌افزار می‌باشند، امّا به وسیله‌ی نقش‌هایی در سایر دیسیپلین‌ها تولید می‌شوند. از جمله می‌توان برنامه‌ی مدیریتِ پیکربندی[۶] و قالبِ فرایندِ تولید[۷] را نام برد.

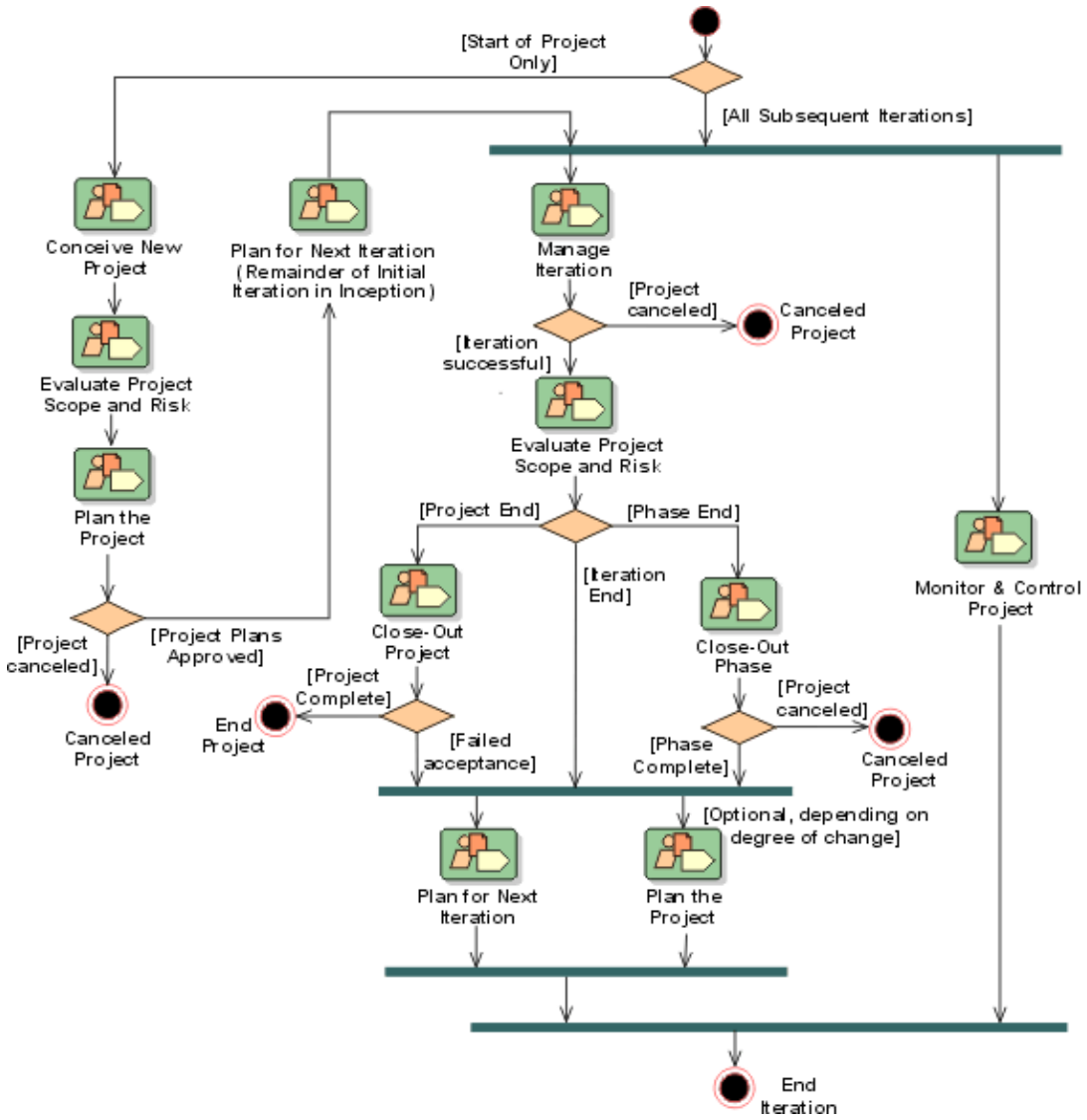
[1] - Iteration Plans
[2] - Iteration Assessment
[3] - (Periodic) Status Assessmsnt
[4] - Work Order
[5] - Project Measurement Database
[6] - Configuration Management Plan
[7] - Development Case

جریانِ کار[1]

در شکل ۱۱-۴ ، با کمک دیاگرامِ فعالیت در یو.اِم.اِل[2]، جریانِ منطقی و ارتباط میان مجموعه فعالیت‌های دیسیپلین مدیریتِ پروژه، نشان داده شده است.

شکل ۱۱-۴

جریان‌کار دیسیپلین مدیریتِ پروژه



[1] - Workflow
[2] - UML

هر یک از نمونه وضعیت‌های فعالیت[1] که به صورت یک مستطیل با لبه‌های خمیده نشان داده شده است، بیانگر به اصطلاح یک جزء از جریان‌کار[2] یا به تعبیری یک جریان‌کار جزئی در آر.یو.پی می‌باشد. هر یک از این جزئیات جریان‌کار به نوبه‌ی خود شامل یک یا چند فعالیت می‌باشد. همان‌گونه که در نمودارِ شکل ۱۱-۴ مشاهده می‌شود، برخی از جزئیات جریان کار، به زمان خاصی در طولِ چرخه‌ی تولید، مانند ابتدا و یا انتهای پروژه وابسته‌اند.

چکیده‌ی فصل

در این فصل دیسیپلین مدیریت پروژه را مختصراً مورد بررسی قرار داده و مهم‌ترین مفاهیمی را که در این دیسیپلین مطرح می‌باشند، معرفی نمودیم. مهم‌ترین مواردی که در این فصل ذکر شد، عبارتست از:

- دیسیپلین مدیریت پروژه در آر.یو.پی راهنمای خوبی است برای برقراری توازن میان اهداف رقابتی، مدیریت ریسک‌ها، و غلبه بر محدودیت‌ها به منظور تحویلِ فراورده‌ای که رضایت مشتریان و کاربران نهایی را جلب نماید.
- در فرایندی مبتنی بر رویکردِ تکرار شونده، تولید باید بر مبنای یک برنامه‌ی سطحی و درشت‌دانه[3] تحتِ عنوان برنامه‌ی فاز[4] یا برنامه‌ی پروژه و تعدادی برنامه‌ی تکرار[5]، انجام شود.
- محرک اصلی برنامه‌ریزی، ریسک می‌باشد. به طور کلی مفهومِ ریسک و اولویت‌بندی میان ریسک‌های عمده‌ی یک پروژه، شکل‌دهنده‌ی مفهوم و ماهیت فازها در آر.یو.پی می‌باشد.
- در ایجاد یک برنامه‌ی فاز (یا برنامه‌ی کلی پروژه)، باید ملاحظات و انتخاب‌های مختلف بر حسب نیروها، منابع موجود، زمانبندی، و محدوده‌ی پروژه را در نظر داشت.
- معیارهای تعریف اهداف و محدوده‌ی فعالیت‌های یک تکرار در فازهای مختلف، بر حسب اهداف و ماهیت آن فاز، متفاوت می‌باشد.

---

[1] - Activity State
[2] - Workflow Detail
[3] - Coarse-grained
[4] - Phase Plan
[5] - Iteration Plan

پرسش‌هایی برای تحقیق و مطالعه‌ی بیشتر

۱. در رابطه با معیارهای مهم در برنامه‌ریزی تکرار در آر.یو.پی تحقیق نمایید.

۲. در ارتباط با تعامل میان نقش مدیر پروژه، معمار نرم‌افزار، و مهندس فرایند در آر.یو.پی و چگونگی تاثیر این تعامل بر هماهنگی فعالیت‌های تیمی تحقیق نمایید.

۳. تفاوت‌های کلیدی میان برنامه و طرح ساختِ یک ساختمان و برنامه‌ی ساختِ یک نرم‌افزار را برشمارید.

۴. مفهوم اندازه‌گیری[1] و ارتباط آن با مفاهیم تغییر و نیز پیکربندی[2] را بررسی نمایید.

۵. مهم‌ترین موانع و چالش‌های پیش‌روی مدیران پروژه در حرکت به سمت رویکرد تکرارشونده را بررسی نمایید.

۶. حجم فعالیت‌های مدیریت پروژه را در رویکرد تکرارشونده و رویکرد آبشاری مقایسه نمایید.

۷. مهم‌ترین معیارهای سنجش پروژه را در دو رویکرد آبشاری و تکرارشونده، مقایسه نمایید.

۸. عوامل تهدید کننده‌ی رویکردِ تکرارشونده را بررسی نمایید.

[1] - Measurement
[2] - Configuration

منابع و مراجع

[1]. Ivar Jacobson, Grady Booch, James Rumbaugh, (1999). *The Unified Software Development Process*, Reading, MA: Addison-Wesley

[2]. Philippe Kruchten, (2003). *The Rational Unified Process: An Introduction*, Reading, MA: Addison-Wesley.

[3]. Per Kroll, Philippe Kruchten, (2003). *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*, Reading, MA: Addison-Wesley.

[4]. Rational Software Corporation. (2003) "Rational Unified Process 2003.06," Available at: http://www.rational.com/

[5]. Pankaj Jalote, (2002). *Software Project Management in Practice*, Reading, MA: Addison-Wesley.

[6]. Walker Royce, (1998). *Software Project Management: A Unified Framework*. Reading, MA: Addison-Wesley.

[7]. Craig Larman, (2003). *Agile and Iterative Development: A Manager's Guide*, Reading, MA: Addison-Wesley.

[8]. Software Academy. (2006) "Unified Process Knowledge Base," Available at: http://www.unifiedProcess.info/

[9]. Scott W. Ambler, (2000). *The Unified Process Inception Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[10]. Scott W. Ambler, (2000). *The Unified Process Elaboration Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[11]. Scott W. Ambler, (2000). *The Unified Process Construction Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[12]. Scott W. Ambler and Larry L. Constantine, (2001). *The Unified Process Transition Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

فصل دوازدهم

دیسیپلینِ مدل‌سازی سازمان

مهم‌ترین موضوعات مورد بررسی در این فصل، عبارتند از:

- اهداف دیسیپلینِ مدل‌سازی سازمان در آر.یو.پی
- فعالیت‌ها، دستاوردها، و نقش‌های کلیدی
- دلایل و سناریوهای مدل‌سازی سازمان
- رسیدن به مدلِ نیازمندی‌های سیستم از روی مدل‌سازمانی

دیسیپلینِ مدل‌سازی سازمان

۱۲

در این فصل، درباره‌ی دیسیپلین مدل‌سازی سازمان (کسب‌وکار) و اینکه چرا این مفهوم در یک پروژه‌ی نرم‌افزاری اهمیت دارد، صحبت خواهیم کرد. از موارد مهمی که در این فصل به تشـریح آن خواهیم پرداخت، مـی‌توان به چگونگی رسیدن از مدل‌های سازمان به مدل نیازمندی‌های نرم‌افزار، اشاره نمود. باید توجه داشته باشیم که این دیسیپلین به مفاهیمی فراتر از نرم‌افزار و عمدتاً مفاهیمی از دنیای سازمان و کسب‌وکار[1] توجه دارد.

علیرغم اینکه معنای دقیق‌تر کلمه‌ی Business فراتر از مفهومِ سازمان و شاملِ گستره‌ای از انواع کسب‌وکار می‌باشد، در این کتاب از کلمه‌ی سازمان استفاده شده است. بنابراین، دیسیپلینِ مدل‌سازی کسب‌وکار در همه‌ی انواع کسب‌وکارها چه سازمانی و چه غیر سازمانی کاربرد دارد.

شکل ۱۲-۱

نمای مقطعی از دیسیپلین مدل‌سازی سازمان در طول فازهای مختلف فرایند تولید

[1] - Business

هدف

مهم‌ترین اهدافِ مدل‌سازی سازمان یا به طور کلی، مدل‌سازی کسب‌وکار، عبارتست از:

- درک ساختار[۱] و پویایی[۲] سازمانی که باید سیستم یا سیستم‌های نرم‌افزاری در آن مستقر شوند. (اصطلاحاً سازمانِ هدف[۳])
- درک مسائل و مشکلات جاری سازمانِ هدف و شناسایی پتانسیل‌ها و نقاط بهبود،
- اطمینان از اینکه مشتریان، کاربران نهایی[۴]، و تولیدکنندگان[۵] نرم‌افزار، همگی درک مشترکی از سازمان هدف دارند،
- استنتاج[۶] و استخراج نیازمندی‌های سیستم یا سیستم‌هایی که باید سازمانِ هدف را در دستیابی به اهدافش، پشتیبانی نمایند.

برای نیل به این اهداف، دیسیپلین مدل‌سازی سازمان، چگونگی توسعه‌ی یک چشم‌انداز[۷] از سازمانِ جدید را توصیف نموده و بر اساس این چشم‌انداز، چگونگی تعریف فرایندها، نقش‌ها، و مسئولیت‌های سازمان را در یک مدل از سازمان بیان می‌نماید. مدلِ سازمان شامل دو مدل مختلف می‌باشد که عبارتند از: مدل مواردِکاربرد سازمانی[۸] و مدل اشیاء سازمان[۹].

در واقع، برای مدل‌سازی سازمان، با آن همانند یک سیستم رفتار می‌کنیم. تمام تکنیک‌ها و روش‌هایی که برای مدل‌سازیِ یک سیستم نرم‌افزاری استفاده می‌شود را می‌توان در سازمان نیز بکار برد. مدل‌های حاصل نیز تا اندازه‌ی زیادی مشابه مدل‌های سیستم می‌باشد.

---

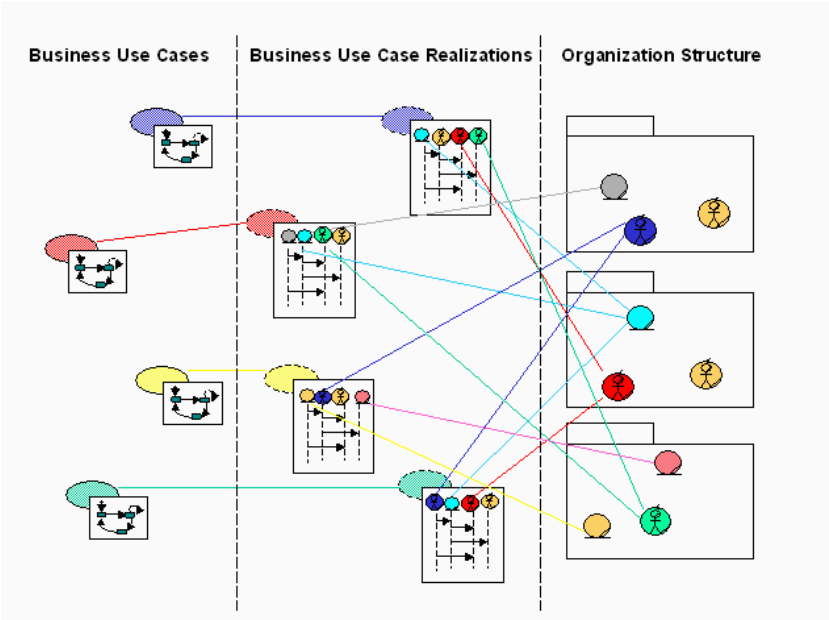
[1] - Structure
[2] - Dynamics
[3] - Target Organization
[4] - End Users
[5] - Developers
[6] - Derive
[7] - Vision
[8] - Business Use-Case Model
[9] - Business Object Model

دلایلِ مدل‌سازی سازمان

امروزه بسیاری از سیستم‌های کاربردی ماهیتی سازمانی دارند. این سیستم‌ها یا به طور مستقیم برگرفته از نیازهای سازمانی می‌باشند و یا اینکه بر آن تأثیرگذار هستند. بنابراین در بسیاری از موارد، شناختِ سازمان، مؤلفه‌ها، ساختار، و پویایی آن، برای شناخت نیازمندی‌های سیستم‌های نرم‌افزاری، امری است ضروری و اجتناب ناپذیر.

شکل ۱۲-۲

مدل‌های سازمان و ساختارِ سازمانی



با ورود سازمان‌ها به دنیای کسب و کارِ الکترونیکی[1] و در نتیجه، لزومِ خودکارسازی فرایندها و خدمات کسب‌وکار از یک سو و پیچیده‌تر شدن سازمان‌ها از سوی دیگر، شناخت و مدل‌سازی سازمان، اهمیت روز افزونی پیدا کرده است. در این میان، سازمان‌ها برای حفظ موقعیت خود در دنیای پر از رقابت، نیازمند بهبود در کسب‌وکارِ خود می‌باشند. نرم‌افزار به عنوان یک مؤلفه‌ی قابل انعطاف، نقشی کلیدی در بهبود سازمانی ایفا می‌نماید به‌گونه‌ای که دیگر هیچ استراتژی سازمانی بدون تعیین جایگاه و نقشِ نرم‌افزار در سازمان، با موفقیت مواجه نمی‌شود.

[1] - E-business

سناریوهای مدل‌سازی سازمان

در این بخش، مهم‌ترین سناریوهای مطرح در مدل‌سازی سازمان را بررسی خواهیم کرد.

سناریوی ۱. چارت سازمانی[1]

در برخی از موارد، یکی از انگیزه‌ها و دلایل تعریف مدل‌سازی سازمان، صرفاً تهیه‌ی چارت سازمانی و فرایندهای آن می‌باشد. براساس چارت سازمانی بدست آمده، شناختِ بهتری نسبت به سازمان و نیازهای آن حاصل می‌گردد. در این مورد، مدل‌سازی سازمان بخشی از پروژه‌ی مهندسی و تولیدِ نرم‌افزار می‌باشد که عمدتاً در طی فاز آغازین (شناخت) انجام می‌شود.

سناریوی ۲. مدل‌سازی قلمرو و دامنه[2]

اگر در حال ایجاد سیستم‌های کاربردی[3] با هدف مدیریت و بازنمایی اطلاعات می‌باشید (مانند سیستم‌های بانکی و مدیریت سفارشات) لازم است که بدون توجه خاص به جریان‌های کارِ سازمان[4]، مدلی از اطلاعات در سطحِ سازمانی[5] فراهم نموده و بر اساس آن مدل اطلاعاتی سیستم استخراج شود. این کار را عمدتاً مدل‌سازی دامنه می‌نامند. مدل‌سازی دامنه، بخش از پروژه‌ی مهندسی و تولید نرم‌افزار بوده که در طی فازهای آغازین (شناخت) و تشریح (معماری) انجام می‌شود.

سناریوی ۳. یک سازمان و چندین سیستم[6]

اگر درصدد ایجاد یک سیستمِ بزرگ[7] یا یکسری سیستم‌های مربوطِ به هم[8] می‌باشید و یا اگر با یک سازمان بزرگ با سیستم‌های متعدد روبرو هستید، مدل‌سازی سازمان به عنوان ورودی برای فعالیت‌های مهندسی نرم‌افزار و پروژه‌ی تولید نرم‌افزار، ضروری خواهد بود. با استفاده از مدلِ سازمان خواهید توانست نیازمندی‌های وظیفه‌مندی سیستم‌ها را یافته و چارچوب کلی معماری مجموعه‌ی سیستم‌ها را ارائه نمایید.

---

[1] - Organization Chart
[2] - Domain Modeling
[3] - Application
[4] - business Workflow
[5] - Business Level
[6] - One Business, Many Systems
[7] - Large System
[8] - A Family of Applications

سناریوی ۴. یک مدل سازمانی عام و کلی[۱]

در مواردی که قصد داشته باشید که یک سیستم کاربردی برای استفاده‌ی چندین سازمان تهیه نمایید، مدل‌سازی سازمان نقش مهمی در درک نقاط مشترک و تفاوت‌های میان سازمان‌های مختلف دارد. به عنوان مثال می‌توان به تولید یک سیستم آموزش الکترونیکی نمونه یا یک سیستم کتابخانه‌ی دیجیتالی برای سازمان‌های مختلف اشاره نمود.

سناریوی ۵. یک سازمان جدید[۲]

اگر یک سازمان درصدد ایجاد یک کسب‌وکار جدید باشد و تصمیم به توسعه‌ی یک سیستم اطلاعاتی مناسب برای آن داشته باشد، باید اقدام به مدل‌سازی سازمان نماید. در این حالت هدفِ مدل‌سازی سازمان، تنها به یافتن نیازمندی‌های سیستم مورد نیاز ختم نمی‌شود، بلکه امکان‌پذیری[۳] کسب‌وکار جدید نیز مستلزم بررسی می‌باشد. معمولاً، در این حالت مدل‌سازی سازمان به صورت یک پروژه‌ی جداگانه مطرح می‌شود.

سناریوی ۶. نوسازی[۴]

در شرایطی که یک سازمان تصمیم بر تحول و اصطلاحاً مهندسی مجددِ فرایندهای سازمان[۵] خود می‌گیرد، مدل‌سازی سازمان به عنوان یکی از مهم‌ترین فعالیت‌ها مطرح می‌شود. مهندسی مجدد فرایندهای سازمان، معمولاً دارای چندین مرحله می‌باشد: ارائه‌ی تصویری از سازمان جدید[۶]، مهندسی معکوسِ سازمان فعلی[۷]، مهندسی سازمان جدید[۸]، استقرار، و راه‌اندازی سازمان جدید.

---

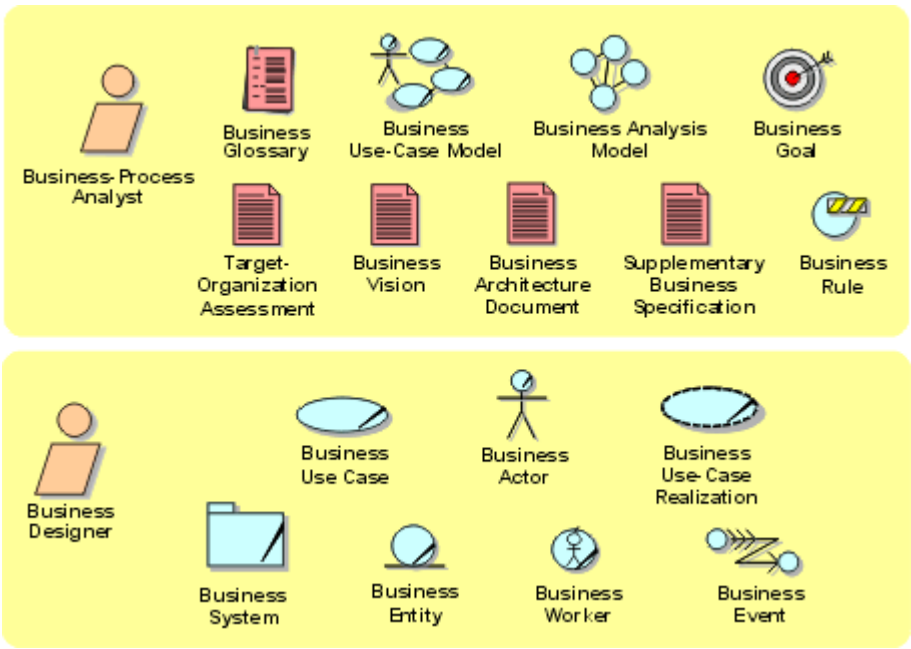
[1] - Generic Business Model
[2] - New Business
[3] - Feasibility
[4] - Revamp
[5] - Business-Process Reengineering (BPR)
[6] - Envision New Business
[7] - Reverse Engineering
[8] - Forward Engineering

نقش‌ها[1] و دستاوردها[2]

همان‌گونه که پیش از این نیز بیان گردید، آر.یو.پی، بخشِ چگونگی[3] فرایند را در قالب نقش‌هـا، دسـتاوردها، فعالیت‌ها، و جریان‌های کار، تعریف می‌نماید. شکل ۱۲-۳ ، نقش‌ها و دسـتاوردهای مـدل‌سـازی سـازمان را نشان می‌دهد.

شکل ۱۲-۳

نقش‌ها و دستاوردهای مدل‌سازی سازمان



مهم‌ترین نقش‌های موجود در دیسیپلین مدل‌سازی سازمان، عبارتند از:

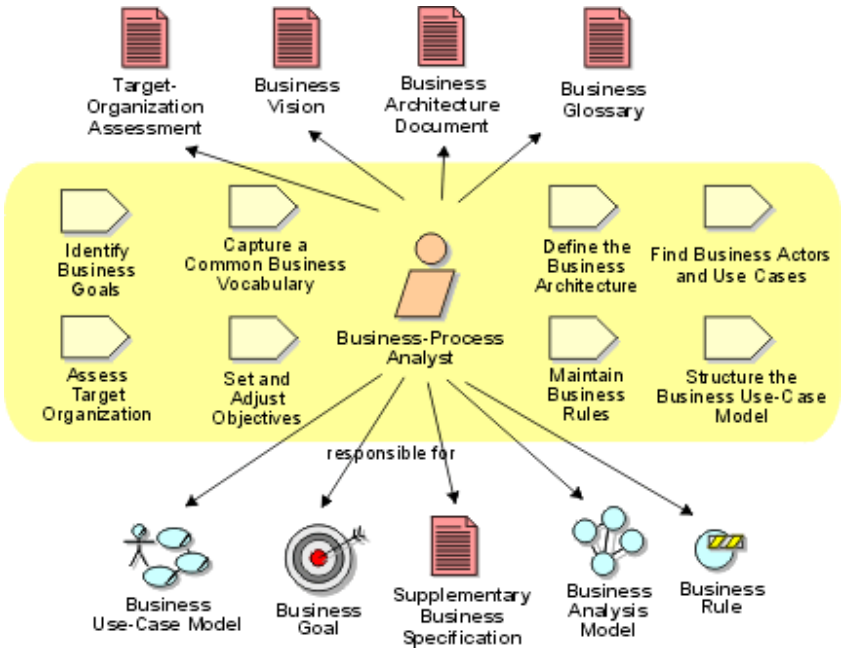
– تحلیل‌گر فرایندهای سازمان[4] : این نقش وظیفه‌ی هدایت و هماهنگی مدل‌سازی مواردِ کاربردِ سازمانی[5] را برعهده دارد. برای مثال، تدوین چشم‌انداز سازمان[6] جدید، بدست‌آوردن اهدافِ سازمانی[7]، و شناسایی اَکتورهای

[1] - Roles
[2] - Artifacts
[3] - How
[4] - Business Process Analyst
[5] - Business Use Cases
[6] - Business Vision
[7] - Business Goals

سازمانی و مواردِ کاربردِ سازمانی و چگونگی تعامل میان آنها، از جمله مهم‌ترین مسئولیت‌های این نقش می‌باشد. در شکل ۱۲-۴ ، مجموعه دستاوردها و فعالیت‌های مرتبط با این نقش نشان داده شده است.

شکل ۱۲-۴

فعالیت‌ها و دستاوردهای مرتبط با تحلیل‌گر فرایندهای سازمان

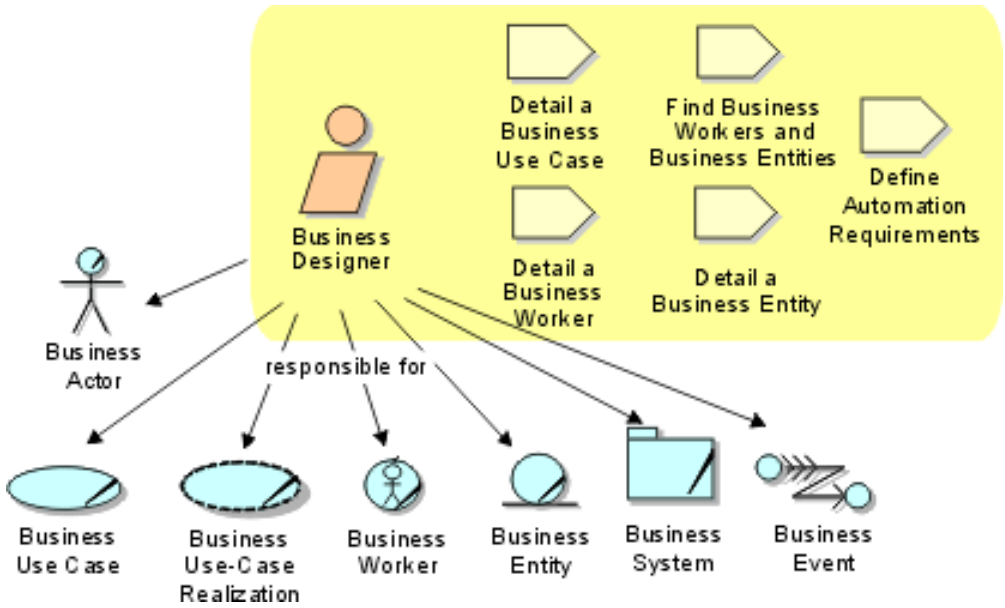


- طراح سازمان[1] : مسئولیت تشریح جزئیات یک بخش از سازمان را با توصیفِ یک یا چند موردِ کاربردِ سازمانی بر عهده دارد. چگونگیِ تحقق[2] مواردِ کاربردِ سازمانی را به وسیله‌ی شناسایی عاملین انجام دهنده‌ی فعالیت‌های سازمان[3] و اقلام و عناصر سازمان[4] و نیز توصیف نحوه‌ی تعامل میان آن‌ها، نشان می‌دهد. طراح سازمان وظیفه‌ی تعریف مسئولیت‌ها[5]، عملیات[6]، ویژگی‌ها[7]، و روابط[8] میان یک یا چند عاملِ انجام‌دهنده‌ی کارها در سازمان و عناصر سازمان را نیز برعهده دارد. در شکل ۱۲-۵ ، مجموعه دستاوردها و فعالیت‌های مرتبط با این نقش نشان داده شده است.

[1] - Business Designer
[2] - Realization
[3] - Business Worker
[4] - Business Entity
[5] - Responsibility
[6] - Operation
[7] - Attributes
[8] - Relationship

شکل ۱۲-۵

فعالیت‌ها و دستاوردهای مرتبط با نقشِ طراحِ سازمان



از دیگر نقش‌هایی که به نحوی در این دیسیپلین درگیر می‌شوند، ذینفعان[1] و نیز ارزیاب یا بازبینی کننده‌ی سازمان[2] را می‌توان نام برد.

مهم‌ترین دستاوردهای[3] دیسیپلینِ مدل‌سازی سازمان عبارتند از:

- سند چشم‌اندازِ سازمان[4] : تعریف‌کننده‌ی اهداف[5] و مقصودهای[6] برنامه‌ی مدل‌سازی سازمان
- مدل مواردِ کاربرد سازمان[7] : مدلی از مجموعه‌ی وظیفه‌مندی‌ها و سرویس‌های سازمان از منظر بیرونی
- مدل تحلیلی سازمان[8] : مدلی از اشیاء درون سازمان، بیانگر نحوه‌ی تحقق مواردِ کاربرد سازمان

[1] - Stakeholders
[2] - Business Reviewer
[3] - Artifacts
[4] - Business Vision Document
[5] - Goals
[6] - Objectives
[7] - Business Use Case Model
[8] - Business Analysis Model

سایر دستاوردهای دیسیپلینِ مدل‌سازی سازمان عبارتند از:

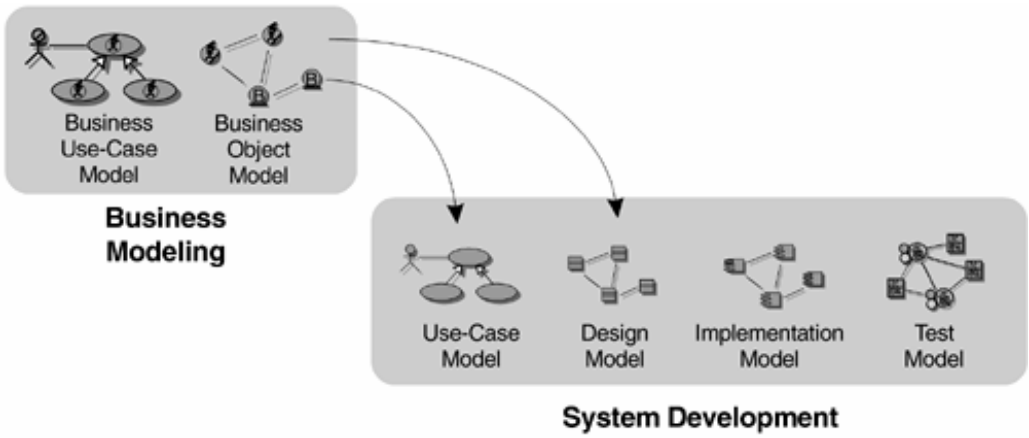
- ارزیابی سازمانِ هدف[1]
- قوانین سازمانی[2]
- مجموعه توصیف‌های تکمیلی سازمان[3]
- فرهنگ واژه‌های سازمان[4]
- سندِ معماری سازمان[5] : دربرگیرنده‌ی منظرهای مختلفِ سازمان و تصمیم‌های کلیدی مرتبط با آنها

استخراج مدل‌های سیستم از مدل‌های سازمان

همان‌گونه که در شکل ۱۲-۶ ، نشان داده شده است، مدل‌های سیستم بر اساس مدل‌های سازمان استخراج می‌شوند. توضیحات کاملتر مرتبط با این بحث را در آر.یو.پی، پی‌گیری نمایید.

شکل ۱۲-۶

اِنتقال از مدل‌های سازمان به مدل‌های سیستم

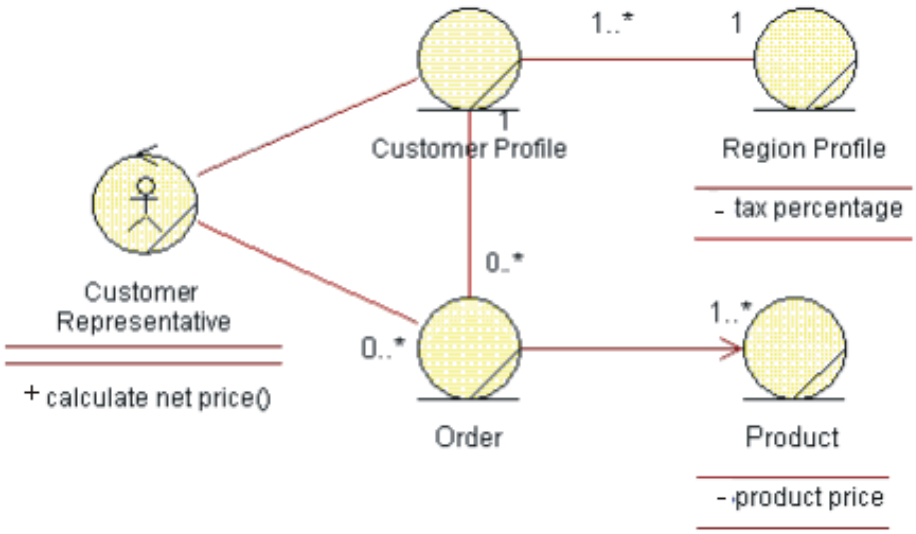


شکل ۱۲-۷ ، نمونه‌ای از یک مدلِ تحلیلِ سازمانی را نشان می‌دهد.

[1] - Target-Organization Assessment
[2] - Business Rules
[3] - Supplementary Business Specifications
[4] - Business Glossary
[5] - Business Architecture Document

شکل ۱۲-۷

نمونه‌ای از یک مدل تحلیلِ سازمانی

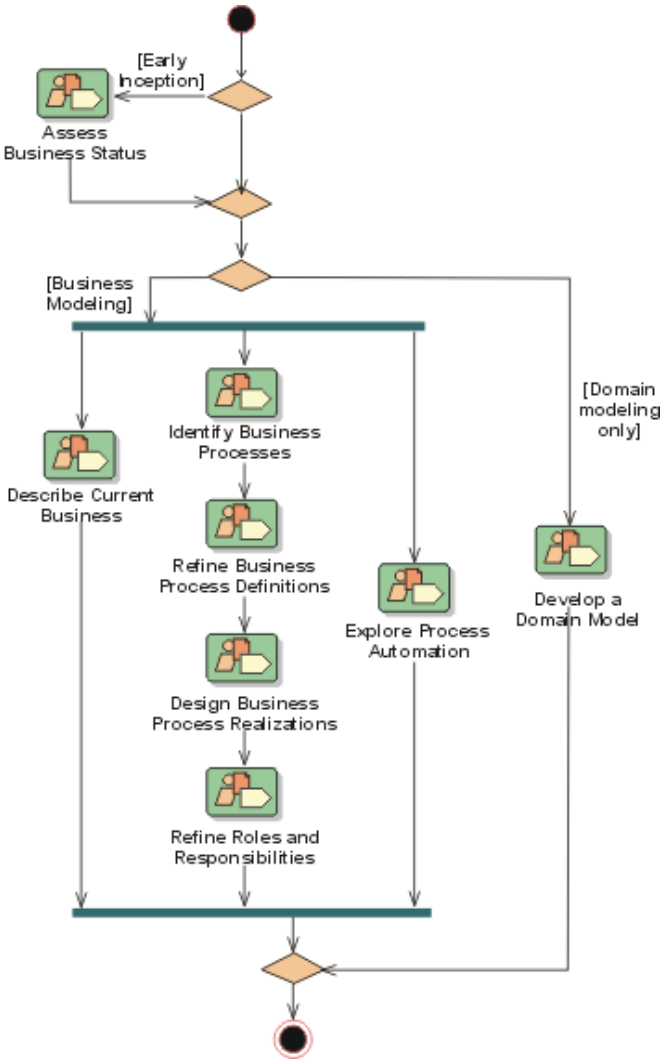


جریان‌کار[1]

در شکل ۱۲–۸ ، دیاگرام جریان‌کار مربوط به دیسیپلینِ مدل‌سازی سازمان، نشان داده شده است. همانگونه که در دیاگرام جریان‌کارِ این دیسیپلین مشخص است، بر حسب هدفِ مدل‌سازی و نیز حال و هوای فاز، چندین مسیر برای پیمایش جریان کار مدل‌سازی سازمان وجود دارد.

شکل ۱۲–۸

جریان کار در دیسیپلینِ مدل‌سازی سازمان



[1] - Workflow

چکیده‌ی فصل

در این فصل یکی دیگر از دیسیپلین‌های آر.یو.پی را بررسی نمودیم. این دیسیپلین، یعنی دیسیپلین مدل‌سازی سازمان[1]، برخلاف سایر دیسیپلین‌های آر.یو.پی که عمدتاً با نرم‌افزار و ملاحظات مرتبط با آن سروکار دارند، با مباحثی نظیرِ سازمان[2]، فرایندهای کسب و کار، اهداف، قوانین، ساختار، پویایی، بهبود، و به طور کلی مدل‌سازی یک سازمان و یا کسب‌وکار، مرتبط می‌باشد و لذا از این نظر تفاوت عمده‌ای با بقیه‌ی دیسیپلین‌های آر.یو.پی دارد. رویکرد و متدولوژی فضای مدل‌سازی سازمان، شباهت بسیار زیادی با رویکرد تحلیل و طراحی سیستم‌های نرم‌افزاری دارد.

مهم‌ترین مباحث مطرح شده در این فصل:

- مدل‌سازی سازمان عمدتاً با هدفِ درک پویایی و ساختار آن، اطمینان از درک مشترک همه‌ی ذینفعان و بدست‌آوردن نیازمندی‌های سیستم یا سیستم‌های مورد نیاز برای پشتیبانی سازمان، انجام می‌شود.
- مدل‌سازی سازمان را می‌توان با استفاده از تکنیک‌های مهندسی نرم‌افزار انجام داد. به تجربه ثابت شده است که این تکنیک‌ها بسیار اثربخش و مؤثر می‌باشند.
- بر حسب ویژگی‌های سازمان، و نیازها و اولویت‌های آن، سناریوهای مختلفی برای مدل‌سازی سازمان مطرح می‌شود.
- در بسیاری از موارد، بخش اَعظمِ نیازمندی‌های نرم‌افزار از روی مدل‌های سازمانی استخراج می‌شود. بنابراین بسیار ارزشمند است که سازمان همواره مدل از وضع موجود[3] خود را داشته باشد تا کمترین دوباره‌کاری در رابطه با مدل‌سازی سازمان صورت پذیرد. در واقع، بر اساس این مدل‌ها، سازمان

[1] - Business Modeling Discipline
[2] - Business
[3] - AS-IS

می‌تواند نیازهای فناوری، تغییرات، و برنامه‌های دیگر خود را در هر لحظه تدوین نموده و نسبت بـه رفع آنها و دستیابی به مدل وضع مطلوب[1]، اقدام نماید.

پرسش‌هایی برای تحقیق و مطالعه‌ی بیشتر

۱. در رابطه با روش‌های مختلف مدل‌سازی سازمان تحقیق نمایید.
۲. مهم‌ترین انگیزه‌های بکارگیری تکنیک‌های مهندسی نرم‌افزار و خصوصاً زبان یو.اِم.اِل و متدولوژی شیءگرا را در مدل‌سازی سازمان برشمارید.
۳. شکل‌های مختلف کسب و کار الکترونیکی[2] را بررسی نمایید.
۴. درباره‌ی چگونگی رسیدن از مدل‌های سازمان به تعریف سیستم‌های مختلف تحقیق نمایید.
۵. با بررسی نمونه‌های مطرح شده در دیسیپلین مدل‌سازی سازمان، چگونگی مدل‌سازی اهـداف سـازمان و ارتباط آن با استراتژی‌های سازمان را توصیف نمایید.
۶. مدلی از مواردِ کاربردِ سازمانی در یک سازمان کوچک ارائه نمایید.
۷. درباره‌ی تفاوت‌های مفاهیم Enterprise ، Business و Organization تحقیق نمایید.
۸. معماری کسب و کار[3] را بررسی نموده و در رابطه با ارتباط میان اجزاء آن بحث نمایید.
۹. نقش مدل‌سازی سازمان در تولید سیستم‌های کسب و کار اِلکترونیکی چگونه است؟
۱۰. درباره‌ی تفاوت‌های میان دو رویکرد و متدولوژی اصلی در مدل‌سازی کسب و کـار، یعنـی رویکردهـای Process-Oriented و Use-Case Driven ، تحقیق نموده و این دو رویکرد را با هم مقایـسه نمایید.

---

[1] - To-Be
[2] - E-Business
[3] - Business Architecture

منابع و مراجع

[1]. Ivar Jacobson, Grady Booch, James Rumbaugh, (1999). *The Unified Software Development Process*, Reading, MA: Addison-Wesley

[2]. Philippe Kruchten, (2003). *The Rational Unified Process: An Introduction*, Reading, MA: Addison-Wesley.

[3]. Per Kroll, Philippe Kruchten, (2003). *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*, Reading, MA: Addison-Wesley.

[4]. Rational Software Corporation. (2003) "Rational Unified Process 2003.06," Available at: http://www.rational.com/

[5]. Eriksson, Hans-Erik, and Magnus Penker, (1999). *Business Modeling with UML*, New York: John Wiley & Sons.

[6]. Software Academy. (2006) "Unified Process Knowledge Base," Available at: http://www.unifiedProcess.info/

[7]. Scott W. Ambler, (2000). *The Unified Process Inception Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[8]. Scott W. Ambler, (2000). *The Unified Process Elaboration Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[9]. Scott W. Ambler, (2000). *The Unified Process Construction Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[10]. Scott W. Ambler and Larry L. Constantine, (2001). *The Unified Process Transition Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

فصل سیزدهم

دیسیپلینِ نیازمندی‌ها

مهم‌ترین موضوعات مورد بررسی در این فصل، عبارتند از:

- اهداف دیسیپلینِ نیازمندی‌ها در آر.یو.پی
- فعالیت‌ها، دستاوردها، و نقش‌های کلیدی
- مفهومِ نیازمندی
- مدلِ واردِکاربرد به عنوانِ یک راهکارِ موفق در مدیریت نیازمندی‌ها

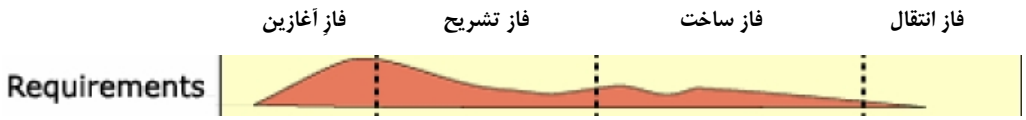
دیسیپلینِ نیازمندی‌ها

۱۳

در این فصل، به بررسی یکی از دیگر از دیسیپلین‌های آر.یو.پی و مفاهیم کلیدی مرتبط با آن خواهیم پرداخت. دیسیپلینِ نیازمندی‌ها یا دیسیپلینِ مدیریتِ نیازمندی‌ها، مجموعه‌ی فعالیت‌ها، دستاوردها، و نقش‌هایی است که با بهـره‌گیری از روش‌ها و تکنیک‌های موفقی مانند مدل‌سازی نیازمندی‌ها به وسیله‌ی مواردِکاربرد[۱]، ما را در جمع‌آوری[۲]، سازماندهی، و مدیریتِ نیازمندی‌های[۳] یک سیستم، یاری می‌دهند.

شکل ۱۳-۱

نمایی مقطعی از دیسیپلینِ نیازمندی‌ها در چرخه‌ی تولید



[1] - Use-Case
[2] - Capture
[3] - Requirement

هدف

اهدافِ دیسیپلینِ نیازمندی‌ها به شرح زیر می‌باشد:

- برقراری و حفظِ توافق میان مشتری و سایر ذینفعان با تیم تولید درباره‌ی چیستی[۱] و چرایی[۲] سیستم
- فراهم‌نمودن درکِ مناسبی از نیازمندی‌های سیستم برای تولیدکنندگان آن
- تعریف و مدیریت مرزهای سیستم[۳]
- فراهم نمودن مبنایی مناسب برای برنامه‌ریزی تکرارهای[۴] مختلف (برنامه‌ریزی تکرارها در آر.یو.پی عمدتاً بر اساس مواردِکاربرد انجام می‌شود)
- فراهم نمودن مبنایی مناسب برای تخمین هزینه و زمان تولیدِ سیستم
- تعریف یک واسطِ کاربر که بر نیازها و اهداف کاربران متمرکز می‌باشد.

برای دستیابی به این اهداف، دیسیپلین نیازمندی‌ها چگونگی تعریفِ یک چشم‌انداز[۵] برای سیستم و ترجمه‌ی آن به مدلِ مواردِکاربرد[۶] و مجموعه مشخصه‌های تکمیلی[۷] که جزئیات نیازمندی‌های سیستم را در بر خواهند داشت، تشریح می‌نماید. این دیسیپلین، چگونگی استفاده از خصیصه‌های نیازمندی‌ها[۸] را برای کمک به مدیریت محدوده‌ی سیستم و نیز چگونگی مدیریت تغییرات نیازمندی‌ها را تعریف می‌نماید.

---

[1] - What
[2] - Why
[3] - System Boundary
[4] - Iteration
[5] - Vision
[6] - Use Case
[7] - Supplementary Specification
[8] - Requirements Attributes

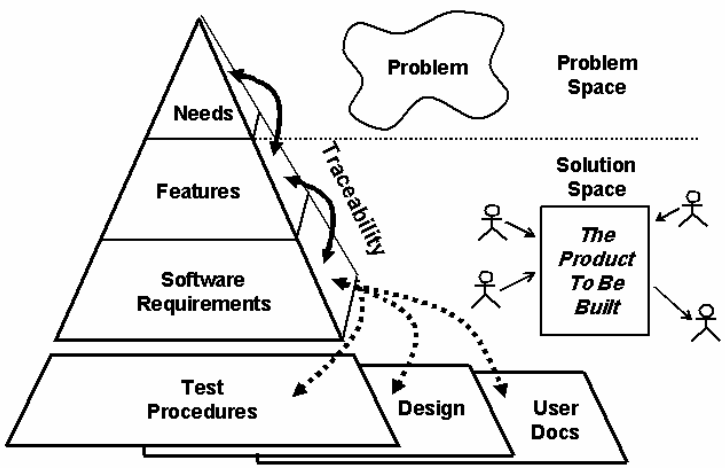
تعریفِ مفهومِ نیازمندی[1]

در اینجا یک «نیازمندی» را به صورتِ حالت[2] یا قابلیتی[3] که باید سیستم بر آن پایبند باشد، تعریف می‌نماییم. به عبارت دیگر، نیازمندی‌های نرم‌افزار عبارتست از مجموعه‌ی معیارهای کِیفی[4] مانند وظیفه‌مندی[5]، قابلیت استفاده[6]، قابلیت اعتماد[7]، کارایی[8]، و قابلیت پشتیبانی[9]. از این منظر، شاخص‌های مطرح در نیازمندی‌ها منطبق با شاخص‌های مطرح در مقوله‌ی کیفیتِ سیستم می‌باشند.

باید توجه داشت که در آر.یو.پی، نیازمندی‌های نرم‌افزار را از نیازها[10] و مجموعه قابلیت‌ها و ویژگی‌های[11] سیستم تفکیک می‌نماییم. شکل ۱۳-۲، ارتباط میان نیازها، قابلیت‌ها، و نیازمندی‌های نرم‌افزار را نشان می‌دهد.

شکل ۱۳-۲

ارتباط میان نیازها، قابلیت‌ها، و نیازمندی‌های نرم‌افزار



[1] - Requirement
[2] - Condition
[3] - Capability
[4] - Quality Criteria
[5] - Functionality
[6] - Usability
[7] - Reliability
[8] - Performance
[9] - Supportability
[10] - Needs
[11] - Feature

مسلماً، آنچه در قالب نرم‌افزار تحقق پیدا می‌کند و پیاده‌سازی می‌شود، باید با نیازهای واقعی کاربران تطابق داشته باشد. امّا، آنچه کاربران درخواست[۱] می‌نمایند لزوماً نیازهای واقعی آنان نمی‌باشد. آر.یو.پی درخواست کاربران را درخواست ذینفعان[۲] می‌نامد. بر اساس این درخواست و با کمک تکنیک‌هایی که در دیسیپلینِ نیازمندی‌ها مطرح شده است، نیازهای واقعی[۳] کاربران و سایر ذینفعان تحلیل شده و بر اساس آن، سرویس‌ها یا مواردِکاربرد[۴] سیستم بدست می‌آیند.

انواع نیازمندی‌های نرم‌افزار را می‌توانیم به دو دسته تقسیم نماییم:

۱. نیازمندی‌های وظیفه‌مندی[۵]

به طور معمول، وقتی که به نیازمندی‌های یک سیستم فکر می‌کنیم، اولین چیزهایی که به ذهنمان خطور می‌کند، مجموعه‌ی وظایف و خدماتی است که سیستم برای کاربران انجام می‌دهد. این مجموعه خدمات را نیازمندی‌های رفتاری، وظیفه‌مندی، یا کارکِردیِ سیستم می‌نامیم. نیازمندی‌های وظیفه‌مندی و یا کارکردی به منظورِ بیانِ رفتارِ یک سیستم به وسیله‌ی توصیف شرایط ورودی و خروجی آن، توصیف می‌شود.

۲. نیازمندی‌های غیر وظیفه‌مندی[۶]

به منظور ارائه‌ی یک فراورده‌ی با کیفیت مطلوب به کاربران نهایی آن، سیستم باید علاوه بر برآورده نمودنِ نیازمندی‌های کارکردی و وظیفه‌مندی، با برخی ویژگی‌های کیفی و ملاحظات غیر کارکردی نیز تطابق داشته باشد. این شرایط و معیارهای کیفی و غیرِ کارکردی، اصطلاحاً نیازمندی‌های غیرِ وظیفه‌مندی نامیده می‌شوند.

البته در برخی از منابع، نیازمندی‌های غیر وظیفه‌مندی نیز به دو دسته تقسیم می‌شوند: محدودیت‌های طراحی[۱] شامل ملاحظاتی مانند تکنولوژی، بستر[۲]، و استانداردهای فنی و نیازمندی‌های غیرِ وظیفه‌مندی مانند کارایی و قابلیت اعتماد.

---

[1] - Request
[2] - Stakeholder Request
[3] - Real Needs
[4] - Use-Case
[5] - Functional Requirements
[6] - Non-Functional Requirements

برخی از مهم‌ترین نیازمندی‌های غیرِ وظیفه‌مندی عبارتند از:

- قابلیتِ بکارگیری[۳] : این دسته از نیازمندی‌ها عمدتاً ملاحظات و فاکتورهای انسانی[۴] را مورد خطاب قرار می‌دهد. از جمله، می‌توان به ملاحظات زیبایی شناسی[۵]، سهولتِ یادگیری[۶]، سهولتِ بکارگیری[۷]، هماهنگی و سازگاری میانِ واسطهای کاربر[۸]، دستنامه‌ی کاربران[۹]، و موادِ آموزشی[۱۰] اشاره نمود.

- قابلیتِ اعتماد[۱۱] : این دسته از نیازمندی‌های غیرِ وظیفه‌مندی به مواردی مانند تناوب و شدتِ اثرِ خرابی[۱۲] سیستم، قابلیت بهبود و بازیافت[۱۳]، قابلیت پیش‌بینی[۱۴]، و صحتِ[۱۵]کارکِرد، اشاره دارد.
- کارایی[۱۶] : نیازمندی‌های کارایی، یکسری شرایط خاص را روی نیازمندی‌های وظیفه‌مندی تحمیل می‌نمایند. برای مثال، نیازمندی‌هایی که نرخ تراکنش[۱۷]، سرعت[۱۸]، دسترسی[۱۹]، صحت[۲۰]، زمان پاسخ[۲۱]، زمان بازیافت[۲۲]، یا میزان مصرفِ حافظه‌ای[۲۳] را که بر اساس آن یک عمل باید انجام شود، تعیین می‌نمایند، نیازمندی‌های غیرِ وظیفه‌مندی و از نوعِ کارایی می‌باشند.
- قابلیتِ پشتیبانی[۲۴] : این نیازمندی شامل مواردی مانند قابلیت تست[۲۵]، قابلیت نگهداری[۱]، و دیگر معیارها و شرایط کیفی است که سیستم پس از تحویل و برای به‌روز[۲] ماندن، به آنها نیاز دارد. از

---

[1] - Design Constraints
[2] - Platform
[3] - Usability
[4] - Human Factors
[5] - Aesthetics
[6] - Ease of Learning
[7] - Ease of Use
[8] - User-Interface Consistency
[9] - User Manual
[10] - Training Material
[11] - Reliability
[12] - Frequency and Severity of Failure
[13] - Recoverability
[14] - Predictability
[15] - Accuracy
[16] - Performance
[17] - transaction Rate
[18] - Speed
[19] - Availability
[20] - Accuracy
[21] - Response Time
[22] - Recovery Time
[23] - Memory Usage
[24] - Supportability
[25] - Testability

آنجایی که این دسته از نیازمندی‌ها بیش از هر چیز به فرایندِ تولید[۳] مرتبط می‌باشد، تا حدودی با بقیه‌ی نیازمندی‌های غیر وظیفه‌مندی، متفاوت است. البته فراموش نکنیم که قابلیت پشتیبانی صرفاً با مستندسازی کاملِ همه‌ی جزئیات حاصل نمی‌شود. بخش عمده‌ای از این قابلیت، در قالبِ معماری سیستم نِمود پیدا می‌کند.

توصیف نیازمندی‌های[۴] سیستم به کمک مواردِکاربرد[۵]

مدل‌سازی به کمک مواردِکاربرد، یکی از قوی‌ترین تکنیک‌های توصیفِ رفتارهای سیستم می‌باشد. در این تکنیکِ مدل‌سازی که در تطابق با مفهومِ مشتری‌مداری[۶] و اولویت‌دهی به خواسته‌های مشتری در تولید فراورده می‌باشد، سیستم از منظر کاربران خارج از آن و به زبان آنان توصیف می‌گردد.

یکی از ویژگی‌های کلیدی این تکنیک، سادگی درک آن برای همه‌ی ذینفعان[۷] و در عین حال جامعیّت و نگاهِ تجریدی آن به مسأله می‌باشد. بنابراین مدل‌سازی نیازمندی‌ها به وسیله‌ی مواردِ کاربرد، پل ارتباطی مناسبی میان سطوح مختلف خواسته‌ها و نیازهای ذینفعان از یک سو و نیازمندی‌های نرم‌افزار از دیدِ تولیدکنندگان از سوی دیگر، می‌باشد.

مدلِ مواردِ کاربرد، مدلی است از سیستم و رفتارهای مطلوبِ آن از منظرِ بیرونی[۸]، یا در واقع، مدلی است که رفتارهای سیستم را از منظر عناصر و موجودیت‌های خارج از آن، توصیف می‌نماید. این مدل دربرگیرنده‌ی مجموعه‌ای از مواردِکاربرد و سرویس می‌باشد که تعریف‌کننده‌ی چگونگی بکارگیری سیستم به وسیله‌ی

---

[1] - Maintainability
[2] - Up-to-date
[3] - Development Process
[4] - Requirements
[5] - Use-Case
[6] - Customer-Oriented
[7] - Stakeholders
[8] - External View

اَکتورها می‌باشد. اَکتور[1] عبارت است از شخص یا چیزی بیرون از سیستم و در تعامل با آن. یک موردِ کاربرد[2]، دیالوگی است میان اَکتور و سیستم.

از منظر این مدل، سیستم همانندِ یک جعبه‌ی سیاه[3] دیده می‌شود. در این مدل تنها به بررسی چیستی و چراییِ سرویس‌ها و نیازمندی‌ها می‌پردازیم. به عبارت دیگر، مدلِ مواردِ کاربرد برای توصیف نیازمندی‌های وظیفه‌مندی[4] سیستم بکار می‌روند. البته برخی ملاحظاتِ مرتبط با نیازمندی‌های غیر وظیفه‌مندی در توصیفِ متنی مواردِکاربرد بیان می‌شود.

معمولاً یکی از مهم‌ترین مشکلات تیم‌های تولیدکننده‌ی نرم‌افزار، لغزش محدوده‌ی تعریفِ سیستم[5] می‌باشد. این مشکل در بسیاری از موارد منجر به عدم موفقیت پروژه‌های نرم‌افزاری می‌شود. تکنیک مواردِکاربرد در کنار سایر ملاحظات و تکنیک‌های مدیریت نیازمندی‌ها، راهکار موفقی در مدیریت دامنه‌ی تعریف سیستم و جلوگیری از لغزش، فراهم می‌نماید.

البته با وجودی‌که درک مدل مواردِ کاربرد بسیار آسان می‌باشد، مدل‌سازی به کمک این تکنیک، ملاحظات و نکات خاصی دارد که در زمانِ مدل‌سازی باید به آنها توجه داشت. برخی از تجارب موفق و نکات ارزشمند در رابطه با مدل‌سازی موارد کاربرد، در قالب راهنمایی‌های آر.یو.پی، فراهم شده است.

---

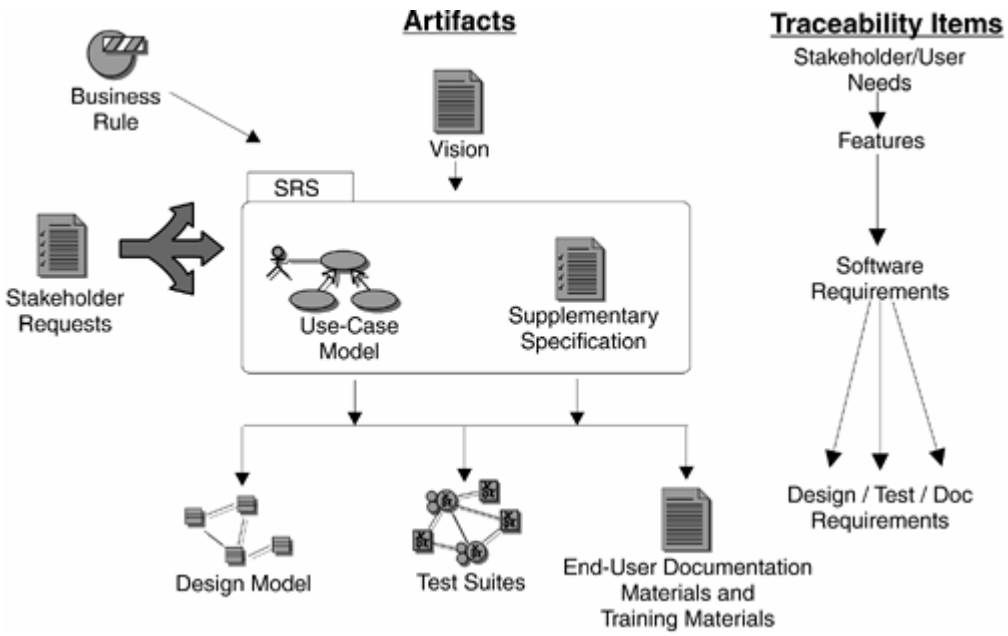
[1] - Actor
[2] - Use-Case
[3] - Black-box
[4] - Functional Requirements
[5] - Scope Creep

دریافت[1] و مدیریتِ نیازمندی‌ها

در شکل ۱۳-۳، نمایی از ساختار ارتباطی میان خواسته‌ها و نیازهای کاربران، دستاوردهای مرتبط با نیازمندی‌ها، و سایر دستاوردهای پروژه نشان داده شده است. در ابتدا درخواست‌های ذینفعان[2] جمع‌آوری می‌شود. این درخواست‌ها شامل لیستی از خواسته‌های همه‌ی کاربران، مشتریان، بخش بازاریابی، و دیگر ذینفعان می‌باشد.

شکل ۱۳-۳

انواع نیازمندی‌ها و ارتباطشان با سایر دستاوردها



این مجموعه درخواست‌های ذینفعان، برای ایجاد سند چشم‌انداز[3] بکار می‌رود. این سند، شامل مجموعه‌ای از نیازهای[4] کلیدی ذینفعان و کاربران و نیز ویژگی‌های[5] سطح بالای سیستم می‌باشد. این ویژگی‌ها بیانگر سرویس‌هایی است که باید در قالب سیستم و برای برآورده نمودن نیازهای ذینفعان تحویل

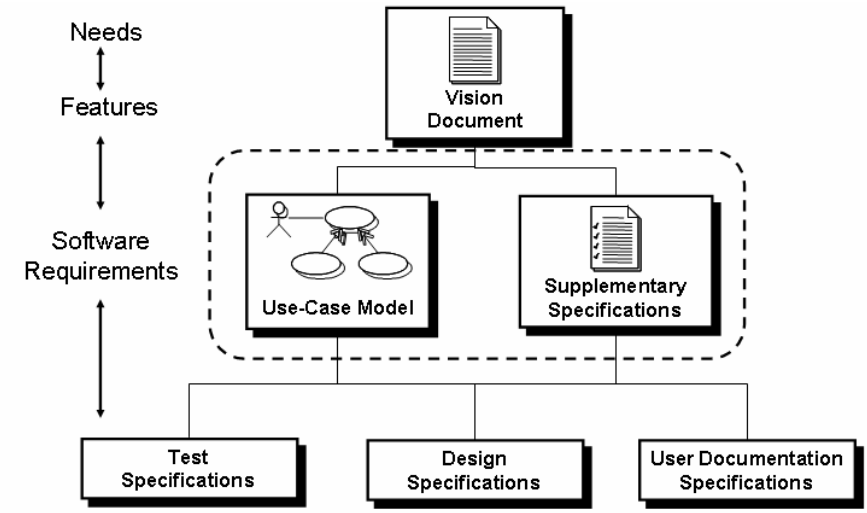
[1] - Acquisition
[2] - Stakeholder Request
[3] - Vision Document
[4] - Need
[5] - Feature

گردد. این ارتباط میان درخواست‌ها، نیازها، ویژگی‌ها، سرویس‌ها، و نیز سایر دستاوردهای سیستم را قابلیت ردگیری[1] می‌نامند.

پیش از آنکه بتوانیم سیستم را بسازیم، لازم است مجموعه‌ی ویژگی‌های سطح بالای ارائه شده را به تفصیل بیان نماییم. این تفصیل به وسیله‌ی مدلِ مواردِکاربرد[2] و سایر توصیف‌های مکمل[3]، انجام می‌شود.

شکل ۱۳-۴

دسته‌بندی انواع نیازمندی‌های نرم‌افزار و ارتباط آن‌ها با برخی از مستندات



[1] - Traceability
[2] - Use-Case Model
[3] - Supplementary Specification

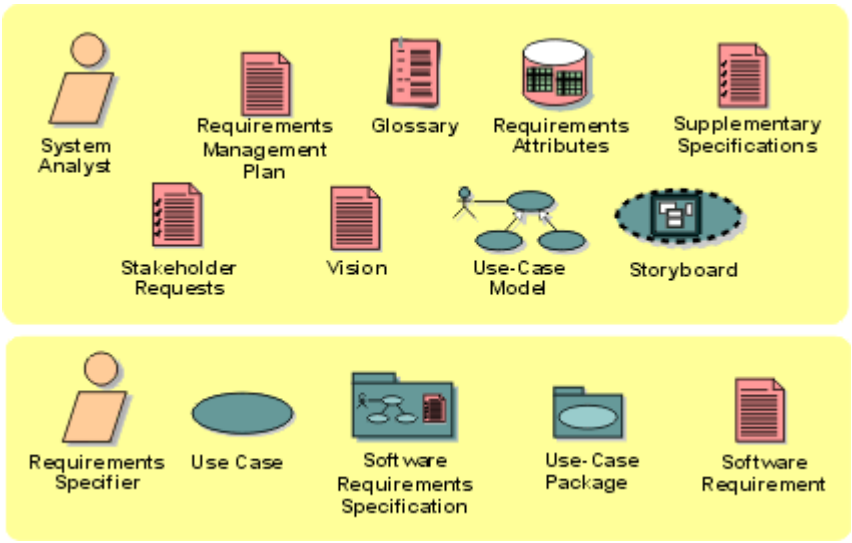
نقش‌ها[1] و دستاوردهای[2] دیسیپلین نیازمندی‌ها

شکل ۱۳- ۵، نقش‌های اصلی دیسیپلین نیازمندی‌ها را نشان می‌دهد. همانگونه که پیش از این نیز بیان گردید، رفتارهای نقش‌های تعریف شده در آر.یو.پی در قالبِ انجامِ فعالیت‌های مختلف، تفسیر می‌شود. نقش‌های دیسیپلینِ نیازمندی‌ها، عبارتند از:

- تحلیل‌گرِ سیستم[3] : هدایت و هماهنگی فعالیت‌های مرتبط با جمع‌آوری نیازمندی‌ها و مدل‌سازی مواردِ کاربرد[4] را برعهده دارد.
- توصیف‌گر نیازمندی‌ها[5] : مسئولیت تشریح جزئیات نیازمندی‌های تمام یا بخشی از یک سیستم را با توصیف جزئیات یک یا چند موردِکاربرد، برعهده دارد.

شکل ۱۳-۵

نقش‌ها و دستاوردهای دیسیپلین نیازمندی‌ها



[1] - Roles
[2] - Artifacts
[3] - System Analyst
[4] - Use-Case
[5] - Requirements Specifier

تحلیل‌گرِ سیستم[۱]، ضمن کار با ذینفعان پروژه به تحلیل مسأله پرداختـه و نیازهـای واقعـیِ ذینفعـان[۲] را درک می‌نماید. سپس با توجه به این شناخت، چیستی[۳] و چراییِ[۴] سیستم را توصیف می‌نماید. این نقش، آنچه را که در قالب سیستم می‌گنجد و آنچه را که خارج از حوزه‌ی آن قرار می‌گیرد، به خوبی شناسایی مـی‌نمایـد. پس از آن، تحلیل‌گرِ سیستم چشم‌اندازِ[۵] پروژه را تدوین می‌نماید. ایـن چـشم‌انـداز کـه مـوردِ توافـقِ همـه‌ی ذینفعان و دربرگیرنده‌ی انتظارات کلیدی آنان می‌باشد، مبنای بیان مدلِ مواردِکاربرد، قرار می‌گیـرد. در ادامـه، یک یا چند موردِکاربرد و نیز مجموعه مشخصه‌های تکمیلی[۶]، در اختیار توصیف‌گرِ نیازمندی‌ها[۷] برای تشریح و توصیفِ جزئیات، قرار می‌گیرد. این نقش به کمک سایر نقش‌ها، از جمله تحلیل‌گرِ سیستم و نیز طراحِ واسـطِ کاربر[۸]، مجموعه فعالیت‌ها و مسئولیت‌های خویش را انجام می‌دهد.

برخی از مهم‌ترین دستاوردهای دیسیپلین نیازمندی‌ها عبارتند از:

- سندِ چشم‌انداز[۹] : فراهم‌کننده‌ی چشم‌اندازی مشترک میان تمام ذینفعان پروژه می‌باشد. در این سند، انتظارات، شرایط و معیارهای کیفی، جایگاه ذینفعان، شرح مسأله، و توصیف نکات کلیدی راهکار، از منظر مشتری بیان می‌شود.
- مدلِ مواردِ کاربرد[۱۰] : باید به عنوان یک رسانه‌ی ارتباطی و نوعی قرارداد میان مشتری، کـاربران، و تولیدکنندگان سیستم، به منظورِ تعریف وظیفه‌مندی‌ها و کارکردهای سیستم، تلقی شود.
- مجموعه مشخصه‌هـای تکمیلـی[۱۱] : بـه عنـوان مکمـلِ بـسیار مهمـی بـرای مـدلِ مواردِکـاربرد و دربرگیرنده‌ی نیازمندی‌های غیرِ وظیفه‌مندی[۱۲] می‌باشد.

---

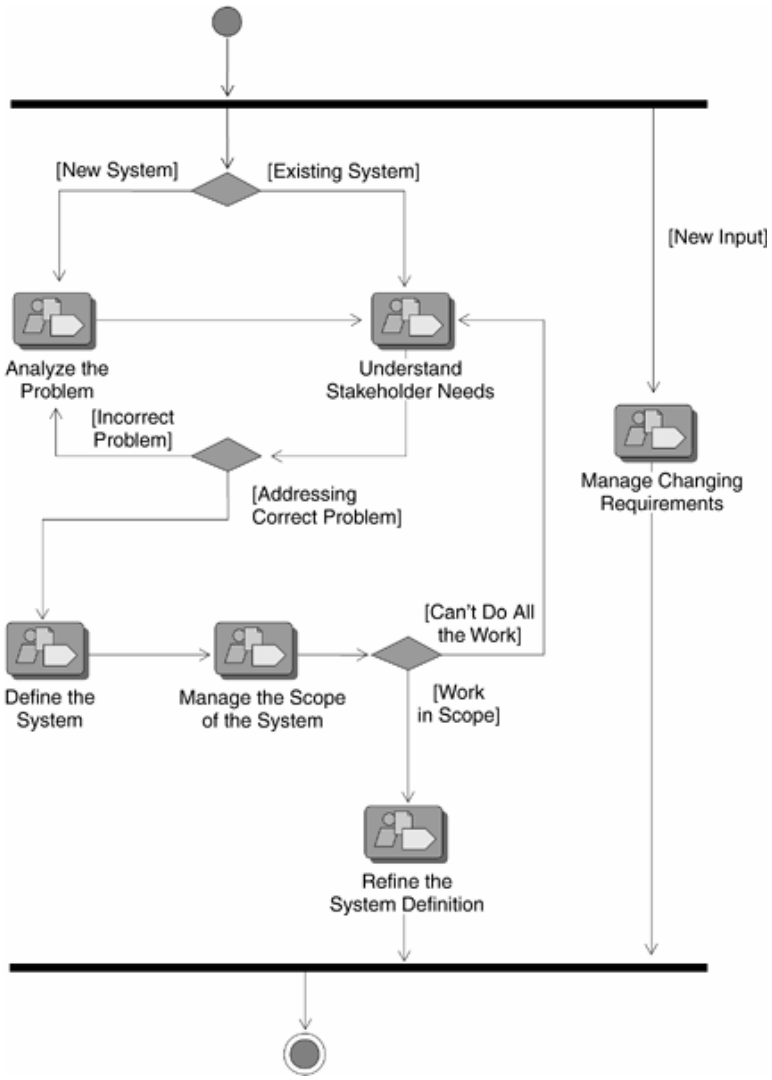
[1] - System Analyst
[2] - Stakeholder
[3] - What
[4] - Why
[5] - Vision
[6] - Supplementary Specifications
[7] - Requirements Specifier
[8] - User-Interface Designer
[9] - Vision Document
[10] - Use-Case Model
[11] - Supplementary Specifications
[12] - Non-Functional Requirements

جریان کار نیازمندی‌ها[1]

در شکل ۱۳-۶ ، جریانِ‌کارِ دیسیپلین نیازمندی‌ها که بیانگرِ ترتیبِ منطقی انجامِ مجموعه‌ی فعالیت‌های ایـن دیسیپلین است، نشان داده شده است.

شکل ۱۳-۶

نمودار جریان‌کارِ دیسیپلین نیازمندی‌ها



[1] - Requirements Workflow

چکیده‌ی فصل

در طی این فصل، با یکی از مهم‌ترین دیسیپلین‌های آر.یو.پی، یعنی دیسیپلینِ نیازمندی‌ها[۱] آشنا شدیم. این دیسیپلین در تناظر با یکی از راهکارهای موفق در زمینه‌ی مهندسی نرم‌افزار، یعنی مدیریتِ نیازمندی‌ها[۲] ، نیز می‌باشد.

چکیده‌ای از مهم‌ترین مباحث مطرح شده در این فصل به شرح زیر می‌باشد:

- مدیریت نیازمندی‌ها، مستلزم یک کار تیمی است که با هدف برقراری و حفظ توافق[۳] میان ذینفعان[۴] و تیم تولید[۵]، درباره‌ی آنچه[۶] سیستم باید انجام دهد، انجام می‌شود.
- به طور معمول، در یک پروژه، انواع مختلفی از نیازمندی‌ها وجود دارد. برخی بسیار سطح بالا بوده و در قالب ویژگی‌های[۷] سیستم بیان می‌شوند؛ برخی نیز نیازمندی‌های وظیفه‌مندی تفصیلی[۸] و نیازمندی‌های غیر وظیفه‌مندی[۹] می‌باشند.
- برای مدیریتِ مؤثر محدوده[۱۰] و ابعاد پروژه و نیز مدیریت تغییرات نیامندی‌ها در طول پروژه، نگهداری خصیصه‌های نیازمندی‌ها[۱۱] و نیز برقراری قابلیتِ ردگیری[۱۲] میان آن‌ها بسیار ضروری است.
- امروزه، مدل‌سازی نیازمندی‌ها به کمک مدلِ مواردِکاربرد، یکی از راهکارها و تکنیک‌های موفق در مدیریت نیازمندی‌ها است. این تکنیک در عین سادگی، دارای قدرت فراوانی در توصیف نیازمندی‌های سیستم و مدیریت حوزه‌ی تعریف آن دارد.

---

[1] - Requirements Discipline
[2] - Requirement Management
[3] - Agreement
[4] - Stakeholders
[5] - Development Team
[6] - What
[7] - Features
[8] - Detailed Functional Requirements
[9] - Non-Functional
[10] - Scope
[11] - Requirement Attributes
[12] - Traceability

پرسش‌هایی برای تحقیق و مطالعه‌ی بیشتر

۱. درباره‌ی نقش و جایگاه ابزار در دیسیپلین نیازمندی‌ها تحقیق نمایید.

۲. در آر.یو.پی چه ابزاری برای مدیریت نیازمندی‌ها معرفی شده است؟ قابلیت‌ها و ویژگی‌های کلیدی این ابزار را بررسی نمایید.

۳. تفاوت‌های کلیدی دیدگاه مبتنی بر مواردِکاربرد[1] و دیدگاه تابعی[2] در توصیف نیازمندی‌ها را بررسی نمایید.

۴. چگونگی مدل‌سازی و مدیریت مواردِکاربرد در یک سیستم بزرگ که متشکل از چندین زیرسیستم می‌باشد را بررسی نمایید.

۵. درباره‌ی چگونگی تحقّق تکنیک ردگیری[3] تحقیق نمایید.

۶. در تکنیکِ مدل‌سازی مواردِ کاربرد، مدیریت حوزه و دامنه‌ی مسأله چگونه انجام می‌شود؟

۷. درباره‌ی تکنیک‌های جمع‌آوری نیازمندی‌ها، تحقیق نمایید.

۸. تکنیک‌های تحلیل مسأله را در آر.یو.پی بررسی نمایید.

---

[1] - Use-Case
[2] - Functional
[3] - Traceability

منابع و مراجع

[1]. Ivar Jacobson, Grady Booch, James Rumbaugh, (1999). *The Unified Software Development Process*, Reading, MA: Addison-Wesley

[2]. Philippe Kruchten, (2003). *The Rational Unified Process: An Introduction*, Reading, MA: Addison-Wesley.

[3]. Per Kroll, Philippe Kruchten, (2003). *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*, Reading, MA: Addison-Wesley.

[4]. Rational Software Corporation. (2003) "Rational Unified Process 2003.06," Available at: http://www.rational.com/

[5]. Software Academy. (2006) "Unified Process Knowledge Base," Available at: http://www.unifiedProcess.info/

[6]. Leffingwell, Dean, and D. Widrig, (1999). *Managing Software Requirements: A Unified Approach*, Reading, MA: Addison-Wesley.

[7]. Cockburn, Alistair, (2001). *Writing Effective Use Cases*, Boston: Addison-Wesley.

[8]. Kurt Bittner and Ian Spence, (2003). *Use Case Modeling*. Boston: Addison-Wesley.

[9]. Scott W. Ambler, (2000). *The Unified Process Inception Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[10]. Scott W. Ambler, (2000). *The Unified Process Elaboration Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[11]. Scott W. Ambler, (2000). *The Unified Process Construction Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[12]. Scott W. Ambler and Larry L. Constantine, (2001). *The Unified Process Transition Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

فصل چهاردهم

دیسیپلینِ تحلیل و طراحی

مهم‌ترین موضوعات مورد بررسی در این فصل، عبارتند از:

- اهداف دیسیپلینِ تحلیل و طراحی در آر.یو.پی
- فعالیت‌ها، دستاوردها، و نقش‌های کلیدی
- مقایسه‌ی مدل‌های مواردِکاربرد، تحلیل، و طراحی

دیسیپلینِ تحلیل و طراحی

۱۴

در این فصل، مباحث و مفاهیم کلیدی مرتبط با دیسیپلین تحلیل و طراحی را بررسی می‌نماییم. پس از بررسی اهدافِ این دیسیپلین، تعریف مدل‌های تحلیل و طراحی و تشریح ارتباط آن‌ها با سایر مدل‌ها، به بررسی مهم‌ترین دستاوردهای طراحی، یعنی کلاس‌ها[1]، زیرسیستم‌ها[2]، و همکاری‌ها[3] خواهیم پرداخت.

شکل ۱۴-۱

نمایی مقطعی از دیسیپلینِ تحلیل و طراحی در طول فازهای چرخه‌ی تولید

[1] - Class
[2] - Subsystem
[3] - Collaboration

هدف

هدف اصلی دیسیپلینِ تحلیل و طراحی، ترجمه‌ی نیازمندی‌ها[۱] به توصیف‌هایی است که چگونگی پیاده‌سازی[۲] سیستم را نشان می‌دهند. همانگونه که پیش از این نیز بیان گردید، مدلِ نیازمندی‌ها عمدتاً بیانگرِ چیستی[۳] و نیز شرایط عملکرد (وظیفه‌مندی‌های) سیستم است. در این مدل، سیستم (راه حل) به عنوان یک جعبه‌ی سیاه[۴] از منظرِ کاربران و ذینفعان آن، توصیف می‌شود. در مقابل، در مدل‌های تحلیل و طراحی به بررسی چگونگیِ[۵] تحققِ[۶] نیازمندها توجه داریم. برای نیل به این مقصود، ابتدا باید نیازمندی‌ها را به خوبی درک نموده و پس از آن با انتخابِ بهترین استراتژی پیاده‌سازی، نیازمندی‌های را به طراحی سیستم[۷] ترجمه نماییم.

در اوایلِ پروژه، باید یک معماری مستحکم[۸] ایجاد شود، به گونه‌ای که طراحی سیستم در چارچوب آن قابل درک بوده و بتوان آن را به راحتی و در چارچوب هزینه‌ها و محدودیت‌های موجود، ساخت و کامل نمود. سپس باید طراحی را با توجه به محیط پیاده‌سازی تطبیق داده و با توجه به ملاحظات کیفی[۹] فراورده، نظیر کارایی[۱۰]، استحکام[۱۱]، مقیاس‌پذیری[۱۲]، و قابلیت گسترش[۱۳] که در قالبِ نیازمندی‌های غیروظیفه‌مندی[۱۴] بیان شده‌اند، طراحی را کامل‌تر نماییم.

---

[1] - Requirements
[2] - Implementation
[3] - What
[4] - Black-Box
[5] - How
[6] - Realization
[7] - System Design
[8] - Robust Architecture
[9] - Quality
[10] - Performance
[11] - Robustness
[12] - Scalability
[13] - Extensibility
[14] - Non-Functional Requirements

تحلیل در مقابلِ طراحی

به طور کلی، هدفِ تحلیل[۱] عبارت است از انتقال نیازمندی‌های سیستم، از زبان و دنیای مشتری به زبان و دنیای کارشناسانِ تیمِ تولیدِ سیستم. مدل نیازمندی‌ها، سیستم را مانند یک جعبه‌ی سیاه[۲] معرفی می‌نماید. در این مدل، سعی می‌نماییم تا آنجا که امکان دارد از پرداختن جزئیات مربوط به پیاده‌سازی‌های خاص سیستم، اجزاء داخلی آن، و حتی واسط کاربر پرهیز نموده و عمدتاً به چیستی[۳] و چرایی[۴] سیستم توجه نماییم. مدلِ مواردِ کاربرد به خوبی بیانگرِ این دیدگاه می‌باشد.

در مقابل، تحلیل به توصیفِ چگونگی تحققِ[۵] وظیفه‌مندی‌ها با معرفی سیستم به صورت یک جعبه‌ی خاکستری[۶] از اجزاء درونی آن، متمرکز می‌باشد. در واقع، مدلِ تحلیل بدون وارد شدن به جزئیات خاصِ پیاده‌سازی و ملاحظاتِ غیرِ وظیفه‌مندی، چگونگیِ تحققِ مواردِ کاربردِ سیستم بررسی می‌شود. توجه داشته باشید که مدل‌سازی[۷]، مصداق بارز مفهوم تجرید[۸] است. این مفهوم از مهم‌ترین تکنیک‌های مهندسی و به منظور غلبه بر پیچیدگی[۹] استفاده می‌شود. با رفتن از نیازمندی‌ها به تحلیل، طراحی، و پس از آن، پیاده‌سازی، به تدریج به جزئیات مورد نیاز اضافه شده و میزان تجرید کمتر می‌شود.

در مدلِ تحلیل، اولین گام را به توصیف درون سیستم برمی‌داریم، امّا واردِ جزئیات پیاده‌سازی‌های مختلف و نیز بسیاری نیازمندی‌های غیرِ وظیفه‌مندی[۱۰] نخواهیم شد. بنابراین، مدلِ تحلیل مدلی عام[۱۱] از چگونگی رفتارهای سیستم ارائه می‌دهد، بدون آنکه مثلاً در آن به زبان جاوا، مبتنی بر وب بودن، یا یک بانک اطلاعاتی خاص، اشاره‌ای داشته باشیم. به عبارت دیگر، مدل تحلیل بیانگر تصویری ایده‌آل از سیستم

---

[1] - Analysis
[2] - Black-Box
[3] - What
[4] - Why
[5] - Realization
[6] - Gray-Box
[7] - Modeling
[8] - Abstraction
[9] - Complexity
[10] - Non-functionl Requirements
[11] - Generic Model

می‌باشد. به کمک این مدل، می‌توان ناسازگاری‌ها[۱] و افزونگی‌های[۲] موجود میان نیازمندی‌ها را تشخیص داده و رفع نمود.

جدول ۱۴-۱

مقایسه‌ی مدلِ تحلیل و مدلِ نیازمندی‌ها

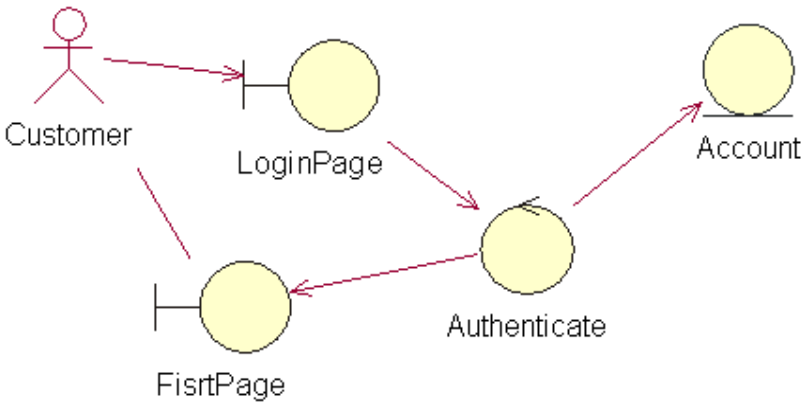
مدلِ تحلیل	مدلِ نیازمندی‌ها
توصیف به زبان کارشناسان تیمِ تولید	بیان شده به زبان مشتری
ارائه‌ی منظر درونی سیستم	ارائه‌ی منظر بیرونی سیستم
سازماندهی و ساختاردهی بر اساس بسته‌ها و کلاس‌های کلیشه‌ای	سازماندهی بر اساس مواردِ کاربرد
عمدتاً توسط افراد تیم توسعه و برای درک سیستمی که باید طراحی و در نهایت پیاده‌سازی شود، به کار می‌رود.	عمدتاً قراردادی میان مشتری و تیم توسعه در رابطه با چیستی سیستم و بایدها و نبایدهای آن
افزونگی و ناسازگاری جایز نیست (افزونگی و ناسازگاری‌های موجود در نیازمندی‌ها به وسیله‌ی مدل تحلیل حذف می‌شود)	ممکن است با افزونگی (Redundancy) و ناسازگاری (Inconsistency) همراه باشد.
بیانگر نحوه‌ی تحقق (Realization) وظیفه‌مندی‌های سیستم	بیان وظیفه‌مندی‌های سیستم (Functionality)
دربرگیرنده‌ی نحوه‌ی تحققِ موارد کاربرد (Use-Case Realization)	دربرگیرنده‌ی تعریفِ مواردکاربرد (Use Case Definition)

در مدل تحلیل، برای بیان نحوه‌ی تحقّق موارد کاربرد[۳] تنها از سه نوع کلاس کلی استفاده می‌نماییم. این کلاس‌های به اصطلاح کلیشه‌ای[۴]، عبارتند از: کلاس‌های مرزی[۵]، کلاس‌های کنترلی[۶]، و کلاس‌های موجودیت[۷]. لازم به ذکر است که مفهوم کلیشه، از تکنیک‌های گسترش زبانِ مدل‌سازی یو.اِم.اِل[۸] می‌باشد. شکل ۲-۱۴، نمونه‌ای است از یک دیاگرامِ کلاس‌ها در مدل تحلیل.

[1] - Inconsistency
[2] - Redundancy
[3] - Use-Case Realization
[4] - Stereotype
[5] - Boundary Classes
[6] - Control Classes
[7] - Entity Classes
[8] - UML : Unified Modeling Language

شکل ۱۴-۲

نمونه‌ای از یک دیاگرام در مدلِ تحلیل



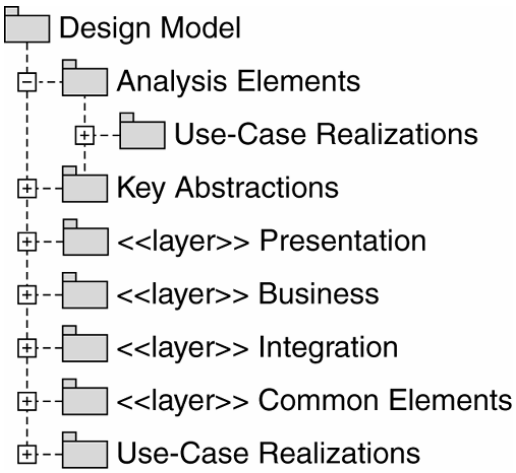
در مقابل، هدفِ طراحی عبارت است از تطبیق[۱] نتایج تحلیل بر اساس محدودیت‌ها[۲] و شرایط تحمیل شده توسط نیازمندی‌های غیرِ وظیفه‌مندی[۳] و محیط پیاده‌سازی[۴]. به عبارت دیگر، مدلِ طراحی در مقایسه با مدل تحلیل، یک توصیف با جزئیات تفصیلی از پیاده‌سازی و شرایط خاصِ آن ارائه می‌دهد و بنابراین آن را می‌توانیم یک جعبه‌ی سفید[۵] از سیستم تلقی کنیم.

مدل طراحی، اصلی‌ترین دستاوردِ دیسیپلینِ تحلیل و طراحی می‌باشد. مدلِ طراحی شامل مجموعه‌ای از همکاری‌های[۶] میان عناصرِ مدل[۷] می‌باشد که مجموعاً رفتارهای سیستم را توصیف می‌نمایند. این رفتارها از روی مواردِ کاربرد و نیازمندی‌های غیرِ وظیفه‌مندی استنتاج می‌شود.

[1] - Adapt
[2] - Constraints
[3] - Non-Functional
[4] - Implementation Environment
[5] - White-Box
[6] - Collaborations
[7] - Model Elements

شکل ۱۴-۳

نمونه‌ای از ساختار مدل طراحی

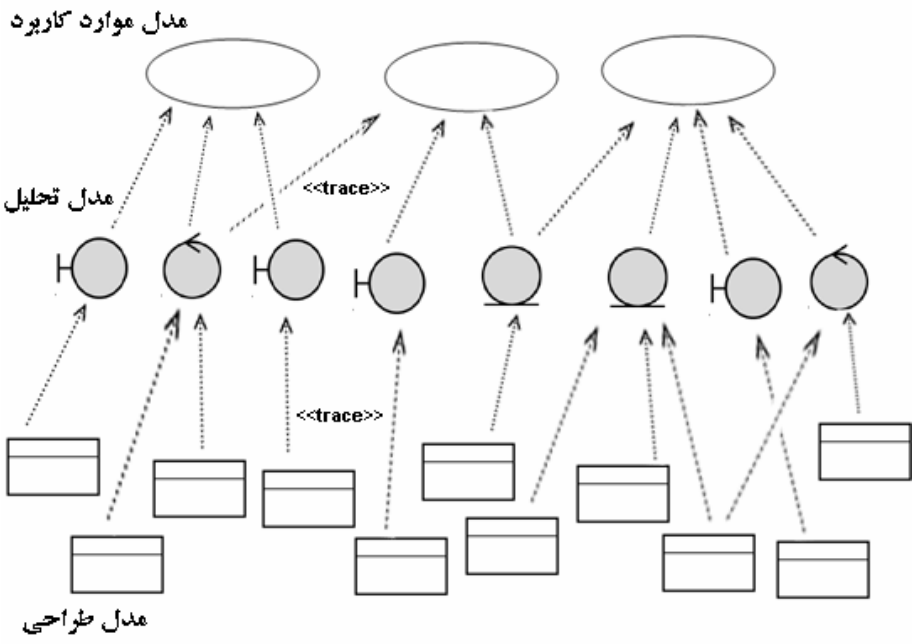


مدل طراحی شامل توصیفی از همکاری‌های میان کلاس‌ها می‌باشد که ممکن است در قالب بسته‌ها و زیرسیستم‌ها در کنارِ هم تجمیع[۱] شوند. یک کلاس[۲] توصیف‌کننده‌ی مجموعه‌ای از اشیاء[۳] با مسئولیت‌ها[۴]، روابط[۵]، عملیات‌ها[۶]، ویژگی‌ها[۷]، و معنای[۸] مشابه می‌باشد. یک بسته[۹]، مکانیزمی است برای سازماندهی و کنترلِ پیچیدگی و نیز گروه‌بندی منطقی عناصر یک مدل، از جمله کلاس‌ها. زیرسیستم[۱۰] بسته‌ای است شامل مجموعه‌ای از کلاس‌های فراهم کننده‌ی رفتارهای[۱۱] خاص.

[1] - Aggregate
[2] - Class
[3] - Object
[4] - Responsibility
[5] - Relationship
[6] - Operation
[7] - Attributes
[8] - Semantics
[9] - Package
[10] - Subsystem
[11] - Behavior

شکل ۱۴-۴

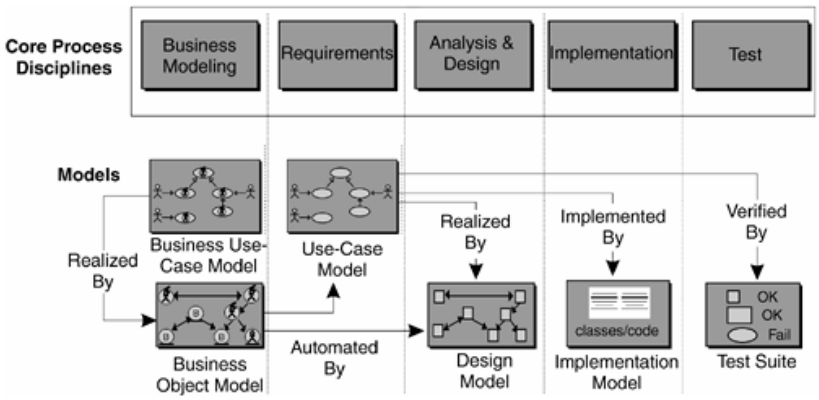
ارتباط میان مدل‌های نیازمندی‌ها، تحلیل، و طراحی



همانگونه که شکل ۱۴-۵ نشان می‌دهد، مدل طراحی (به عنوان دستاوردِ اصلی دیسیپلین تحلیل و طراحی) برای تحققِ مدلِ مواردِکاربرد و نیز بیانِ چگونگی خودکارسازی[1] مدلِ اشیاء سازمانی به کار می‌رود.

شکل ۱۴-۵

ارتباط میان مدل‌های مختلف در فرایند تولیدِ نرم‌افزار



[1] - Automation

در جدولِ ۱۴-۲ ، مدل‌های تحلیل و طراحی با هم مقایسه شده‌اند.

جدول ۱۴-۲

مقایسه‌ی مدلِ طراحی و مدلِ تحلیل

مدلِ طراحی	مدلِ تحلیل
مدلی فیزیکی (نقشه‌ی پیاده‌سازی)	مدلی مفهومی (بدون ملاحظات پیاده‌سازی)
خاص و مربوط به یک پیاده‌سازی مشخص	مدلی عام و کلی برای هر طراحی (Design-Generic) قابل استفاده برای چندین موردِ کاربرد
تعداد نامحدودی از کیشه‌های (فیزیکی) از کلاس‌ها که به پیاده‌سازی‌های مختلف وابسته است.	سه کلیشه‌ی مفهومی از کلاس‌ها: کلاس‌های Control ، Boundary ، و Entity
نسبتاً رسمی‌تر می‌باشد (More Formal)	کمتر حالت رسمی دارد (Less Formal)
پویا (Dynamic) و با تمرکز بیشتری بر توالی (Sequence)	پویا (Dynamic) و با تاکید کمی بر توالی (Sequence)
طراحی سیستم و منظرهای معماری را آشکار می‌نماید (Manifest)	سرفصل‌های طراحی و نیز معماری را بیان می‌کند (Outlines)
عمدتاً با برنامه‌نویسی یا مدل‌سازی بصری در محیطی با قابلیت مهندسی رفت و برگشتی ایجاد می‌شود	عمدتاً از طریق برگزاری کارگاه‌ها و مکانیزم‌های مشابه ایجاد می‌شود
باید در سرتاسر چرخه‌ی عمر یک نرم‌افزار نگهداری شود	ممکن است نگهداری آن در طول چرخه‌ی عمر یک نرم‌افزار ضرورتی نداشته باشد

معمولاً، مدل خاصی از طراحی نیز وجود دارد که شاملِ عناصر داده‌ای و ملاحظات مرتبط با بانک‌های اطلاعاتی مانند جدول‌ها[1]، محدودیت‌ها[2]، Views و Stored Procedures ، Trigger، Schema، و Index می‌باشد. این مدل را مدل داده‌ها[3] می‌نامند.

[1] - Table
[2] - Constraint
[3] - Data Model

نقش‌ها[۱] و دستاوردها[۲]

آر.یو.پی، فرایندِ تحلیل و طراحی را در قالب مجموعه‌ای از نقش‌ها، دستاوردها، فعالیت‌ها، و جریانِ‌کار بیان می‌نماید. در شکل ۱۴-۶ ، نقش‌ها و دستاوردهای این دیسیپلین، نشان داده شده است. مهم‌ترین نقش‌های فعال در دیسیپلینِ تحلیل و طراحی عبارتند از:

- معمار نرم‌افزار[۳] :

معمارِ نرم‌افزار، مسئولیت رهبری و هدایت مجموعه فعالیت‌ها و دستاوردهای فنی را در سرتاسر پروژه برعهده دارد. در واقع معمارِ نرم‌افزار، نقش مدیریت فنی تیمِ تولید نرم‌افزار را ایفا می‌نماید. این نقش، ساختار کلی[۴] سیستم را برای هریک از منظرهای معماری بنا می‌کند: تجزیه‌ی منظرها[۵]، گروه‌بندی عناصر، و واسط میان گروه‌بندی‌های عمده. در مقایسه با نگاه سایرِ نقش‌ها، یک معمار باید نگاهی وسیع، کل‌نگر، و در عین حال سطحی[۶] داشته باشد.

- طراح[۷] :

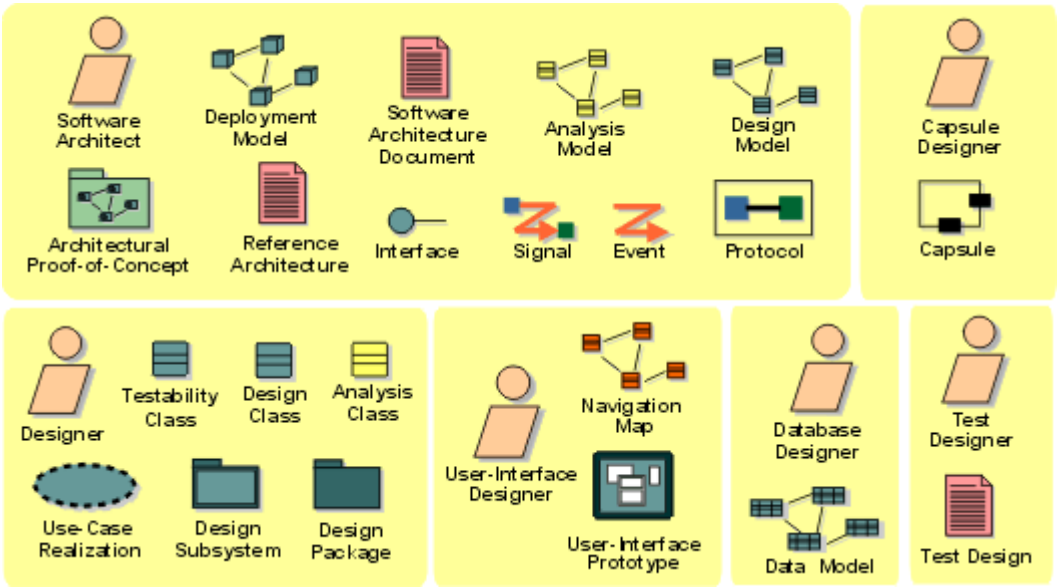
طراحِ نرم‌افزار، مسئولیت‌ها[۸]، عملیات‌ها[۹]، خصیصه‌ها[۱۰]، و روابط یک یا چند کلاس را تعریف کرده و چگونگی تنظیم و تحققِ آنها را در محیط پیاده‌سازی، تعیین می‌نماید.

---

[1] - Roles
[2] - Artifacts
[3] - Software Architect
[4] - Overall Structure
[5] - Decomposition of Views
[6] - Breath View
[7] - Designer
[8] - Responsibilities
[9] -Operations
[10] - Attributes

شکل ۱۴-۶

نقش‌ها و دستاوردها در دیسیپلین تحلیل و طراحی



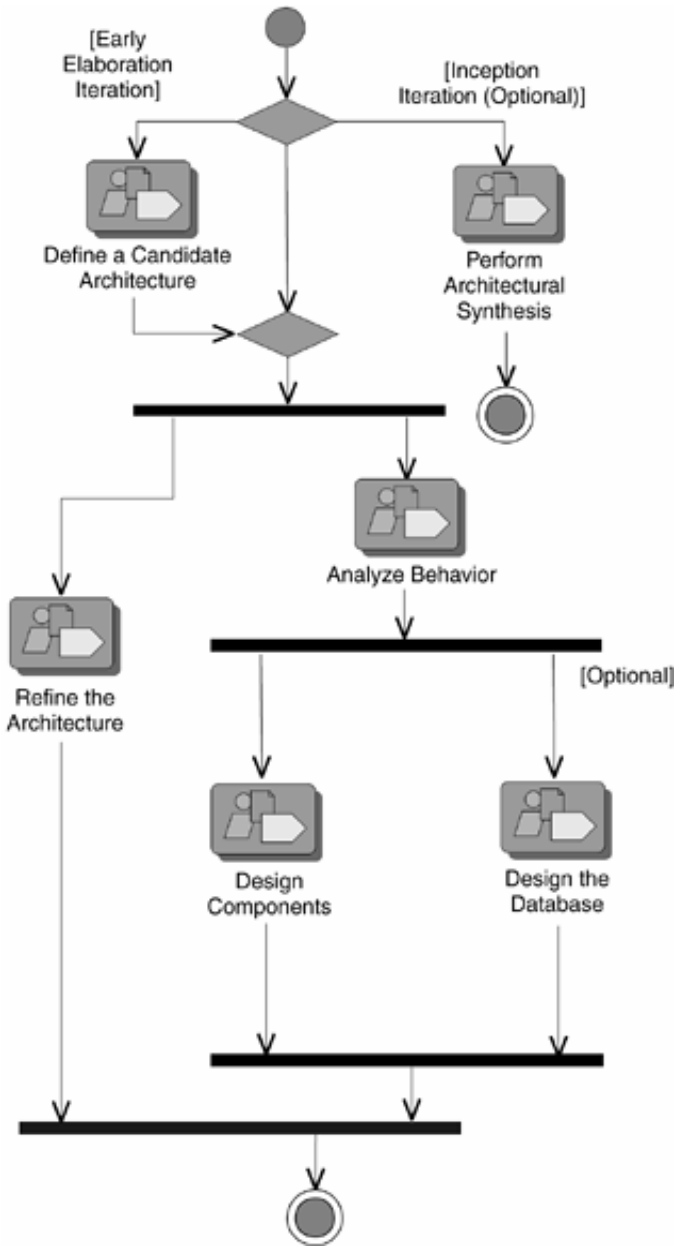
جریان کار[1]

در شکل ۱۴-۷ ، یک تکرار[2] را در دیسیپلین تحلیل و طراحی، در قالب دیاگرام فعالیت[3] که از دیاگرام‌های زبان استانداردِ مدل‌سازی، یعنی یو.اِم.اِل می‌باشد، نشان می‌دهد. هریک از حالت‌های فعالیت[4]، بیانگر یک جزء جریان کار[5] و شامل چندین فعالیتِ به هم مرتبط در آر.یو.پی می‌باشد. به عنوان مثال، تعیین یک معماری کاندیدا[6] یکی از اجزاء جریان کار و شامل مجموعه‌ای از فعالیت‌های به هم مرتبط و در راستای تعریف یک معماری انتخابی می‌باشد. همانگونه که در این شکل نشان داده شده است، برخی از اجزاء جریان کار به فازهای خاصی از فرایند وابسته می‌باشند. به عنوان مثال، در تکرارهای فاز تشریح[7]، معماری سیستم تعریف شده و سپس پالایش می‌شود و بخش‌های مهم و دارای ریسک از سیستم، تحلیل و طراحی می‌شوند.

[1] - Workflow
[2] - Iteration
[3] - Activity Diagram
[4] - Activity State
[5] - Workflow Detail
[6] - Define an Candidate Architecture
[7] - Elaboration

شکل ۱۴-۷

جریان‌کار در دیسیپلین تحلیل و طراحی



چکیده‌ی فصل

در این فصل دیسیپلینِ تحلیل و طراحی[۱] و ملاحظات مرتبط با آن معرفی گردید. مهم‌ترین مباحث مطرح شده در ارتباط با دیسیپلین تحلیل و طراحی عبارتند از:

- دیسیپلینِ تحلیل و طراحی پُلی است میان نیازمندی‌ها[۲] و پیاده‌سازی[۳]. این دیسیپلین، با استفاده از مدلِ مواردِکاربرد[۴]، مجموعه‌ی اشیاء موجود در سیستم را که در نهایت در قالب کلاس‌ها[۵]، زیر سیستم‌ها[۶]، و بسته‌ها[۷] پیاده‌سازی خواهند شد، شناسایی می‌نماید.
- مسئولیت‌های تحلیل و طراحی در میان نقش‌های معمارِ نرم‌افزار[۸] (ارائه‌ی یک تصویر کلی[۹])، طراح[۱۰] (تشریح جزئیات[۱۱]) و طراح بانک‌اطلاعاتی[۱۲] (ملاحظات مرتبط با مانایی[۱۳] اشیاء) توزیع می‌شود.
- نتیجه‌ی نهایی تحلیل و طراحی، ارائه‌ی مدلِ طراحی[۱۴] است. این مدل را می‌توان با استفاده از سه منظرِ معماری[۱۵]، بیان نمود. منظرِ منطقی[۱۶] یا مفهومی بیانگر تجزیه‌ی سیستم در قالب مجموعه‌ای از عناصر مفهومی (کلاس‌ها، زیرسیستم‌ها، بسته‌ها، و همکاری‌ها[۱۷]) می‌باشد. منظرِ پردازه‌ای[۱۸]، نگاشتی است از عناصر مفهومی و منطقی ذکرشده در قالب پردازه‌ها و نخ‌کشی‌ها[۱۹] در سیستم. منظرِ استقرار[۲۰]، بیانگر نگاشتِ پردازه‌ها به مجموعه‌ای از گره‌ها[۲۱] است که باید پردازه‌های سیستم، روی آن‌ها اجرا شود.

---

1 - Analysis and Design Discipline
2 - Requirements
3 - Implementation
4 - Use Cases
5 - Classes
6 - Subsystems
7 - Packages
8 - Software Architect
9 - Big Picture
10 - Designer
11 - Details
12 - Database Designer
13 - Persistence
14 - Design Model
15 - Architectural Views
16 - Logical View
17 - Collaborations
18 - Process View
19 - Threads
20 - Deployment View
21 - Nodes

- طراحی واسط کاربر[1] به موازات سایر فعالیت‌ها پیش‌رفته و نتیجه‌ی آن ارائه‌ی یک پیش‌الگو[2] از واسط کاربر خواهد بود.
- در برخی از موارد، به منظور درکِ کامل‌تر نیازمندی‌ها و انتقال مفاهیم و فضای نیازمندی‌ها (زبان مشتری) به مفاهیم و فضای طراحی (زبان تیم توسعه)، مدلِ تحلیلِ جداگانه‌ای از سیستم ضروری می‌باشد؛ از جمله در مواردی که قصد داشته باشیم چند نوع طراحی مختلف را از سیستم ارائه دهیم، داشتن یک مدل کلی و عام[3] یا همان مدلِ تحلیل، می‌تواند سودمند باشد.

پرسش‌هایی برای تحقیق و مطالعه‌ی بیشتر

۱. مدل طراحی اشیاء (Object Model) و مدل طراحی داده (Data Model) را با هم مقایسه کنید.
۲. در رابطه با مدل معماری ۱+۴ در آر.یو.پی و ارتباط آن با مفهوم معماری مبتنی بر مدل (Model Driven Architecture) متعلق به OMG ، تحقیق نمایید.
۳. مدل تحلیل چگونه می‌تواند پالایش نیازمندی‌ها و حذف ناسازگاری میان نیازمندی‌های نرم‌افزار را تسهیل نماید؟
۴. درباره‌ی ابزارهای مورد نیاز برای انجام فعالیت‌های مختلف دیسیپلین تحلیل و طراحی، تحقیق نمایید.
۵. رویکرد توسعه‌ی مبتنی بر مواردِ کاربرد در آر.یو.پی (Use-Case Driven Development) را با رویکرد توسعه‌ی مبتنی بر مدل (OMG's Model Driven Development) مقایسه نمایید.

---

[1] - User-Interface Design
[2] - Prototype
[3] - General

منابع و مراجع

[1]. Ivar Jacobson, Grady Booch, James Rumbaugh, (1999). *The Unified Software Development Process*, Reading, MA: Addison-Wesley

[2]. Philippe Kruchten, (2003). *The Rational Unified Process: An Introduction*, Reading, MA: Addison-Wesley.

[3]. Per Kroll, Philippe Kruchten, (2003). *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*, Reading, MA: Addison-Wesley.

[4]. Shalloway, Alan, and James Trott, (2002). *Design Patterns Explained: A New Perspective on Object-Oriented Design*, Reading, MA: Addison-Wesley.

[5]. Rational Software Corporation. (2003) "Rational Unified Process 2003.06," Available at: http://www.rational.com/

[6]. Peter Eeles, Kelli Houston, Wojtek Kozaczynski, (2002). *Building J2EE™ Applications with the Rational Unified Process*, Reading, MA: Addison-Wesley.

[7]. Software Academy. (2006) "Unified Process Knowledge Base," Available at: http://www.unifiedProcess.info/

[8]. Scott W. Ambler, (2000). *The Unified Process Inception Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[9]. Scott W. Ambler, (2000). *The Unified Process Elaboration Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[10]. Scott W. Ambler, (2000). *The Unified Process Construction Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[11]. Scott W. Ambler and Larry L. Constantine, (2001). *The Unified Process Transition Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[12]. Luke Hohmann, (2003). *Beyond Software Architecture: Creating and Sustaining Winning Solutions*, Reading, MA: Addison-Wesley.

فصل پانزدهم

دیسیپلینِ پیاده‌سازی

مهم‌ترین موضوعات مورد بررسی در این فصل، عبارتند از:

- اهداف دیسیپلینِ پیاده‌سازی در آر.یو.پی
- فعالیت‌ها، دستاوردها، و نقش‌های کلیدی
- مفهومِ نسخه‌های میانی[۱]
- مفهومِ پیش‌الگو[۲] و انواعِ آن
- مفهومِ یکپارچه‌سازی[۳]

[1] - Builds
[2] - Prototype
[3] - Integration

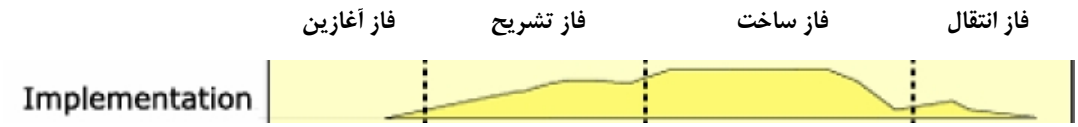
دیسیپلینِ پیاده‌سازی

۱۵

در این فصل، دیسیپلینِ پیاده‌سازی را بررسی خواهیم نمود. پس از معرفی اهداف این دیسیپلین، مهم‌ترین مفاهیم مرتبط با آن، یعنی نسخه‌های میانی، پیش‌الگو، و یکپارچه‌سازی تدریجی را بررسی خواهیم کرد. در شکل ۱۵-۱ نمایی مقطعی از حجمِ فعالیت‌های این دیسیپلین در طول چرخه‌ی تولید نشان داده شده است. همانگونه ملاحظه می‌نمایید، به طـورِ معمول، حجم فعالیت‌های این دیسیپلین در فازِ ساخت بیشتر از سایر فازها است.

شکل ۱۵-۱

نمایی مقطعی از دیسیپلین پیاده‌سازی در طول چرخه‌ی تولید



به تفاوت معنایی مفاهیم پیاده‌سازی[1] و ساخت[2] در آر.یو.پی توجه داشته باشید. بخش عمده‌ی فعالیت‌های دیسیپلینِ پیاده‌سازی به نوعی به برنامه‌نویسی[3] مرتبط می‌باشد. امّا، در فازِ ساخت برای دستیابی به اَهداف تعیین شده، فعالیت‌ها یا دیسیپلین‌های دیگری (علاوه بر پیاده‌سازی) نیز دخیل می‌باشند. بنابراین، پیاده‌سازی

[1] - Implementation
[2] - Construction
[3] - Programming

یکی از اقدامات مهم و عمده‌ی فازِ ساخت است، نه همه‌ی آن. توجه داشته باشید که دیسیپلین پیاده‌سازی ارتباط نزدیکی با دیسیپلینِ تست دارد.

هدف

چهار هدفِ اصلی دیسیپلین پیاده‌سازی، عبارتند از:

- تعریفِ چگونگی سازماندهی[۱] و ساختارِ کدهای برنامه در قالب زیرسیستم‌های[۲] هر لایه[۳] از معماریِ سیستم
- پیاده‌سازی کلاس‌ها[۴] و اشیاء[۵] در قالب یکسری مؤلفه[۶]
- تست مؤلفه‌های پیاده‌سازی شده به صورت واحد[۷]
- یکپارچه‌سازی و مجتمع‌سازی[۸] مؤلفه‌ها در قالب یک سیستم قابلِ اجرا[۹]

در دیسیپلین پیاده‌سازی، مفهومِ تست، محدود به تستِ واحدِ[۱۰] هر یک از مؤلفه‌ها به صورت جداگانه و مستقل می‌باشد. تست‌هایی مانند تستِ سیستم[۱۱] و تستِ یکپارچگی[۱۲] در دیسیپلینِ تست[۱۳] قرار دارند. البته به خاطر داشته باشید که در عمل، جریان‌های کار[۱۴] به هم آمیخته‌اند و در اجرا مرز کاملاً مشخصی میان آنها وجود نخواهد داشت.

---

[1] - Organization
[2] - Subsystem
[3] - Layer
[4] - Class
[5] - Object
[6] - Component
[7] - Unit
[8] - Integration
[9] - Executable System
[10] - Unit Test
[11] - System Test
[12] - Integration Test
[13] - Test Discipline
[14] - Workflows

در ادامه، مفاهیمِ کلیدی زیر را بررسی خواهیم کرد:

- نسخه‌های میانی[۱]
- یکپارچه‌سازی[۲]
- پیش‌الگو[۳]

نسخه‌های میانی[۴]

یک نسخه میانی نسخه‌ای است عملیاتی[۵] از سیستم یا قسمتی از آن که زیرمجموعه‌ای از قابلیت‌هایی را که باید محصول نهایی داشته باشد، به معرض نمایش می‌گذارد. لازم به یادآوری است که در یک فرایند با رویکردِ تکرارشونده[۶]، نتیجه‌ی حاصل از یک تکرار[۷]، معمولاً، یک نسخه‌ی قابل اجرا[۸] از نرم‌افزار می‌باشد. این نسخه‌ی نرم‌افزار را اصطلاحاً Release می‌نامند.

در طی یک تکرار، به منظور فراهم کردن نقاطی برای بازبینی زودهنگام، شناسایی و رفع به موقعِ مشکلات و معضلات مرتبط با یکپارچه‌سازی، و اطمینان از پیشرفت کار، یکسری نسخه‌های میانی تولید می‌شود. این نسخه‌های قابل اجرا را نسخه‌های میانی (معادلی برای واژه‌ی Build) نامیده‌اند. مفهومِ گونه[۹] یا نسخه‌نهایی نرم‌افزار را معادلِ اصطلاحِ Version بکار برده‌ایم. این مفهوم به نسلی از نرم‌افزار که در طی یک چرخه‌ی تولید، متولد شده یا تکامل می‌یابد، اشاره دارد.

---

[1] - Builds
[2] - Integration
[3] - Prototype
[4] - Builds
[5] - Operational Version
[6] - Iterative
[7] - Iteration
[8] - Executable
[9] - Version

در طی فرایندِ مبتنی بر توسعه‌ی تکرارشونده[1]، نسخه‌های میانی متعددی ایجاد می‌شود. هر یک از این نسخه‌های میانی، با فراهم‌آوری یک نقطه‌ی بازبینی زودهنگام[2]، امکان شناسایی و تشخیص به موقعِ مشکلات و معضلات مرتبط با یکپارچه‌سازی را به وجود می‌آورند.

ایجاد نسخه‌های میانی، جزء جدایی‌ناپذیرِ از رویکرد تکرار شونده می‌باشد. با ایجاد نسخه‌های میانی، می‌توانیم سرویس‌ها و قابلیت‌هایی را که تا یک زمان مشخص ایجاد می‌شوند، به معرض نمایش گذاشته و به این وسیله، تکامل تدریجی سیستم را مشاهده نماییم.

هر یک از نسخه‌های میانی، تحت کنترل پیکربندی[3] قرار دارند به گونه‌ای که همواره باید امکان برگشت به نسخه‌های قبل از یک مؤلفه وجود داشته باشد. بنابراین ایجاد نسخه‌های میانی متعدّد در طول تکرارهای مختلف از فرایند تولید، بدون داشتن ابزارهای مناسب برای مدیریت پیکربندی، امکان‌پذیر نمی‌باشد. این موضوع را در دیسیپلین مدیریت پیکربندی و تغییرات، پیگیری نمایید.

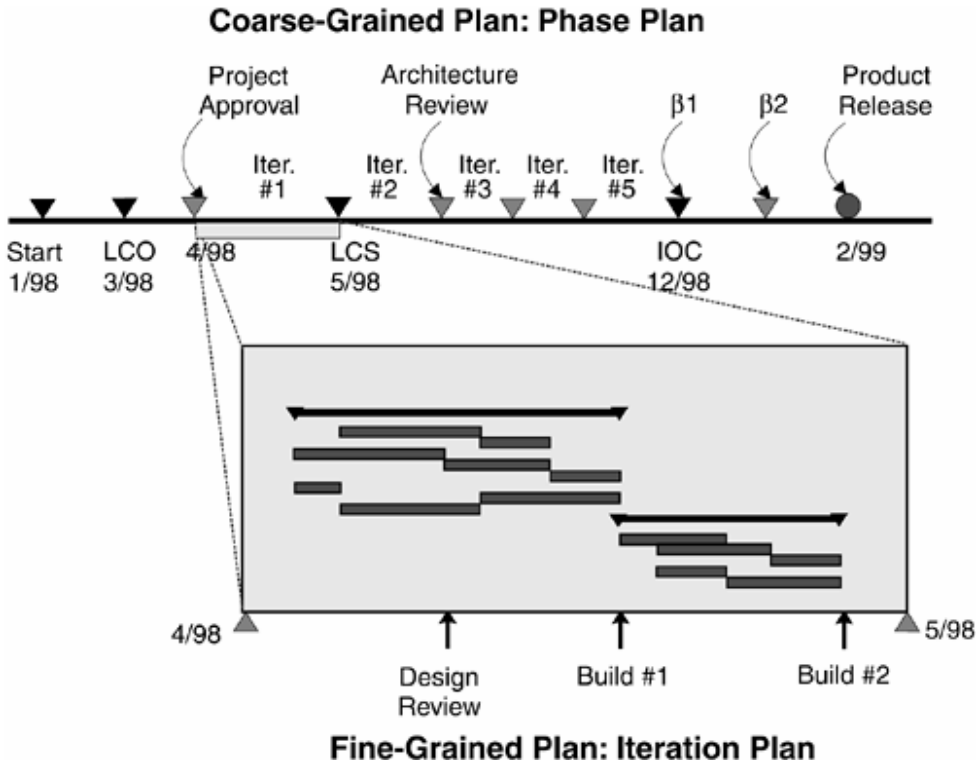
به طور معمول، سعی بر این است که با فراهم‌آوری زیرساخت مناسب، اِمکان ایجاد نسخه‌های میانی به صورت روزانه[4] فراهم گردد. اما در صورتی‌که این کار امکان‌پذیر نباشد، داشتن حداقل ایجادِ هفتگی این نسخه‌های میانی توصیه می‌شود.

---

[1] - Iterative Development
[2] - Early Review Point
[3] - Configuration
[4] - Daily Builds

شکل ۱۵-۲

ایجاد نسخه‌های متعدّد در یک تکرار



یکپارچه‌سازی[1]

واژه‌ی یکپارچه‌سازی یا مجتمع‌سازی به فعالیتی از مجموعه‌ی فعالیت‌های تولید نرم‌افزار اتلاق می‌شود که در آن مؤلفه‌های[2] مجزای نرم‌افزار با هم ترکیب شده و یک موجودیتِ بزرگتر را ایجاد می‌نمایند. یکپارچه‌سازی در سطوح[3] و مراحل[4] مختلفی از پیاده‌سازی انجام می‌شود و اهداف کلی آن عبارتست از:

- یکپارچه‌سازی کارِ یک تیم روی پیاده‌سازی بخش‌های مختلف یک زیرسیستم، پیش از آن‌که آن زیر سیستم تحویل شود،
- یکپارچه‌سازی زیرسیستم‌ها و تشکیل سیستمِ کامل.

[1] - Integration
[2] - Component
[3] - Levels
[4] - Stages

رویکردِ آر.یو.پی در یکپارچه‌سازی عبارتست از یکپارچه‌سازی تدریجی[1] نرم‌افزار. این دیدگاه به معنی پیاده‌سازی و تست قطعه‌کدهای کوچکِ سیستم و ترکیبِ تدریجی آنها برای تشکیل یک قطعه‌ی بزرگ‌تر و در نهایت تشکیل کل سیستم می‌باشد.

رویکردی که مقابل این روش قرار می‌گیرد و در روش آبشاری شاهد آن بودیم، داشتن یک فازِ مجزا تحت عنوان فاز یکپارچه‌سازی[2] است. این رویکرد بر یکپارچه‌سازی چندین مؤلفه (جدید یا تغییر یافته) در یک زمانِ مشخص در اِنتهای پروژه، متکی است. مهم‌ترین معضلِ این رویکرد، این است که در این روش با مطرح شدن متغیرهای متعدد در زمان یکپارچه‌سازی، عملاً شناسایی محل خطاها[3] بسیار مشکل می‌شود. یک خطا می‌تواند در درون هریک از مؤلفه‌های جدید، در تعامل[4] میان مؤلفه‌های جدید، یا در تعامل میان مؤلفه‌های جدید و مؤلفه‌های هسته‌ی[5] سیستم (مجموعه‌ی مؤلفه‌هایی که از قبل با هم تلفیق شده‌اند)، واقع باشد.

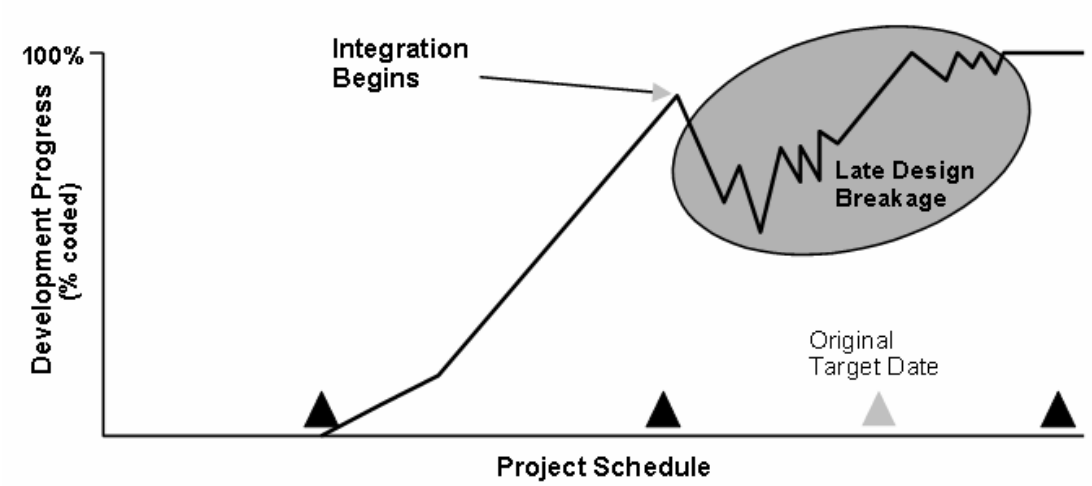
پیش از این، در فصل دوم کتاب که موضوع دلایل و ریشه‌های شکست پروژه‌های نرم‌افزاری مطرح گردید، اشاره نمودیم که علّت تأخیر بسیاری از پروژه‌ها، به تعویق افتادنِ عملیاتِ یکپارچه‌سازی می‌باشد. این مشکل که ویژگی ذاتی رویکرد آبشاری است، باعث می‌شود که به محض اینکه در فاز به اصطلاحْ یکپارچه‌سازی سیستم قرار گرفتیم، بسیاری از خطاهای پنهان آشکار شده و عملاً تحویل به موقع فراورده با مشکل روبرو خواهد شد. این موضوع در شکل ۱۵-۳ نشان داده شده است.

---

[1] - Incremental Integration
[2] - Integration Phase
[3] - Errors
[4] - Interaction
[5] - Core

شکل ۱۵-۳

معضل مهمّ عملیاتِ یکپارچه‌سازی در رویکرد آبشاری



یکپارچه‌سازی تدریجی که عملاً با رویکرد توسعه‌ی تکرارشونده[1] تحقّق می‌یابد، دارای مزایای زیر است:

- پیداکردن محل نواقص کار چندان مشکلی نخواهد بود. زمانی‌که در حین یکپارچه‌سازی تدریجی، مشکل یا نقص جدیدی رخ می‌دهد، مؤلفه‌ی جدید یا تغییر یافته (یا تعامل آن با مؤلفه‌های از قبل یکپارچه شده) اولین جایی است که احتمالاً باید در جستجوی علت آن مشکل بود. رویکردِ یکپارچه‌سازی تدریجی، این احتمال را نیز تقویت می‌نماید که خطاها و نواقص به صورت تدریجی و هر کدام در یک زمان خاصی کشف شوند و همین موضوع، موجبات شناسایی دقیق‌تر علّت مشکل را فراهم می‌آورد.
- هر یک از مؤلفه‌ها به صورتی کامل‌تر تست خواهند شد. مؤلفه‌های جدید به محض تولید، با مؤلفه‌های موجود تلفیق شده و تست می‌شوند. توجه داشته باشید که بسیاری از ملاحظات تستِ یک مؤلفه، در هنگام یکپارچه‌سازی آن با سایر مؤلفه‌ها و قرار گرفتن آن در لایه‌ی مناسبی از معماری، قابل بررسی خواهد بود. به این ترتیب، امکان بررسی و آزمون مکرّر مؤلفه‌های مختلف فراهم می‌گردد. در حالی‌که، در رویکرد آبشاری یا سایر رویکردهایی که با یکپارچه‌سازی به صورت یک فاز برخورد می‌شد، این امکان عملاً وجود نداشت. با توجه به اینکه توسعه‌ی مؤلفه‌های زیرساختی زودتر

[1] - Iterative Development

انجام می‌شود، بنابراین این مؤلفه‌ها که جزء بنیادی و پایه‌ای سیستم می‌باشند، بیشتر تست شده و در نتیجه استحکام سیستم نیز تضمین می‌شود.

- امکان راه‌اندازی زودترِ بخش‌هایی از سیستم فراهم می‌گردد. با یکپارچه‌سازی تدریجی، در طول زمان اِمکان ارائه‌ی نسخه‌های اجرایی از سیستم میسّر می‌شود. این موضوع هم بر رضایت و احساسِ پیشرفت اعضای تیم و نیز مشتری تاثیرگذار می‌باشد و هم دریافت بازخوردهای[1] سریع‌تر را از طراحی‌ها، ابزارها، قوانین، و حتی نوع فرایند، امکان پذیر می‌سازد.

باید توجه داشت که یکپارچه‌سازی در هر تکرار، حداقل یک‌بار اتفاق می‌اُفتد. در برنامه‌ی یک تکرار[2]، موارد کاربردی که باید در طول بازه‌ی زمانی آن تکرار، طراحی شده و کلاس‌هایی که باید پیاده‌سازی شوند، تعریف و تعیین می‌شود. استراتژی یکپارچه‌سازی که در قالبِ همین برنامه می‌باشد، تعیین می‌نماید که چه کلاس‌هایی و با چه ترتیبی پیاده‌سازی و سپس با هم تلفیق خواهند شد.

یکپارچه‌سازی تدریجی نیز همانند ایجاد نسخه‌های میانی، بدون داشتن ابزارهای مناسب برای پیکربندی و یکپارچه‌سازی، امکان‌پذیر نمی‌باشد.

پیش‌الگو[3]

مهم‌ترین انگیزه‌ی تولید پیش‌الگوها در طی یک فرایند تولید یک فراورده، کاهش ریسک می‌باشد. با ارائه‌ی یک پیش‌الگو می‌توان ریسک‌ها و ابهامات مرتبط با موارد زیر را کاهش داد:

- سودمندی و ارزش تجاری[4] فراورده
- استحکام[5] یا کارایی[6] فناوری‌های کلیدی
- بودجه و تعهدات پروژه (از طریق ساخت یک پیش‌الگو برای به اصطلاح اثبات مفهوم[7])

---

[1] - Feedback
[2] - Iteration Plan
[3] - Prototype
[4] - Business Viability
[5] - Stability
[6] - Performance
[7] - Proof-of-Concept

- درک نیازمندی‌ها
- شکل و شمایلِ[1] فراورده و در مجموع، قابلیت استفاده‌ی[2] مناسب از آن

در طول چرخه‌ی تولید، همواره باید به ماهیت و هدف یک پیش‌الگو توجه داشته باشیم. بنابراین وقتی یک پیش‌الگوی رفتاری را به منظور آزمایش واسط کاربر، تولید می‌کنید، انتظار نداشته باشید که این پیش‌الگو، قسمتی از فراورده‌ی نهایی شما باشد.

انواعِ پیش‌الگو

به طور کلی، انواع پیش‌الگوها را به دو طریق می‌توان بررسی نمود: با چیزی که این پیش‌الگوها کاوش[3] می‌نمایند (یا در واقع، هدف از ایجاد یک پیش‌الگو) و یا به وسیله‌ی چگونگی تکامل‌شان در طول چرخه (نتیجه‌ی حاصل از آنها).

از منظر اولین دیدگاه، دو نوع پیش‌الگوی اصلی خواهیم داشت:

- یک پیش‌الگوی رفتاری[4]، که تمرکز بیشتری بر کشف رفتار یک سیستم دارد.
- یک پیش‌الگوی ساختاری[5]، که مرتبط با کاوش در نگرانی‌ها و ملاحظات مرتبط با معماری و فناوری می‌باشد.

از دیدگاه دومین منظر نیز، دو نوع پیش‌الگو وجود دارد:

- پیش‌الگوی مکاشفه‌ای[6]، که پیش‌الگوی دورانداختنی[7] نیز نامیده می‌شود بعد از تکمیل و دست‌یابی به نتیجه‌ی مورد انتظار، دور انداخته می‌شود.

---

[1] - Look and Feel
[2] - Usability
[3] - Explore
[4] - Behavioral Prototype
[5] - Structural Prototype
[6] - Exploratory Prototype
[7] - Throwaway Prototype

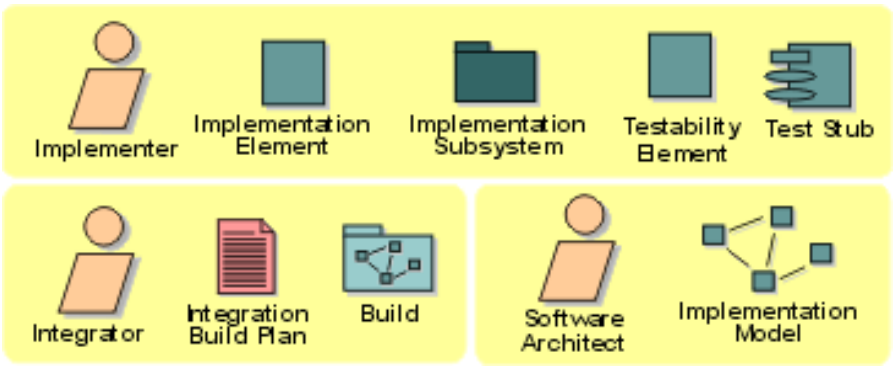
- پیش‌الگوی تکاملی[1]، که برای تبدیل شده به سیستم نهایی، به تدریج تکامل می‌یابد. به عنوان مثال، پیش‌الگوی معماری که از دستاوردهای مهم در فازِ تشریح می‌باشد، از این نوع است.

نقش‌ها[2] و دستاوردها[3]

در شکل ۱۵-۴ ، نقش‌ها و دستاوردهای مرتبط با دیسیپلینِ پیاده‌سازی نشان داده شده است.

شکل ۱۵-۴

نقش‌ها و دستاوردها در دیسیپلینِ پیاده‌سازی



نقش‌های اصلی درگیر در دیسیپلین پیاده‌سازی عبارتند از:

- پیاده‌ساز[4] (برنامه‌نویس[5])، که مسئول پیاده‌سازی و تولید مؤلفه‌ها و دستاوردهای مرتبط با آنها و نیز انجام تستِ واحد[6] روی این مؤلفه‌ها می‌باشد.
- یکپارچه‌ساز سیستم[7]، که مسئول ایجاد نسخه‌های میانی[8] می‌باشد.

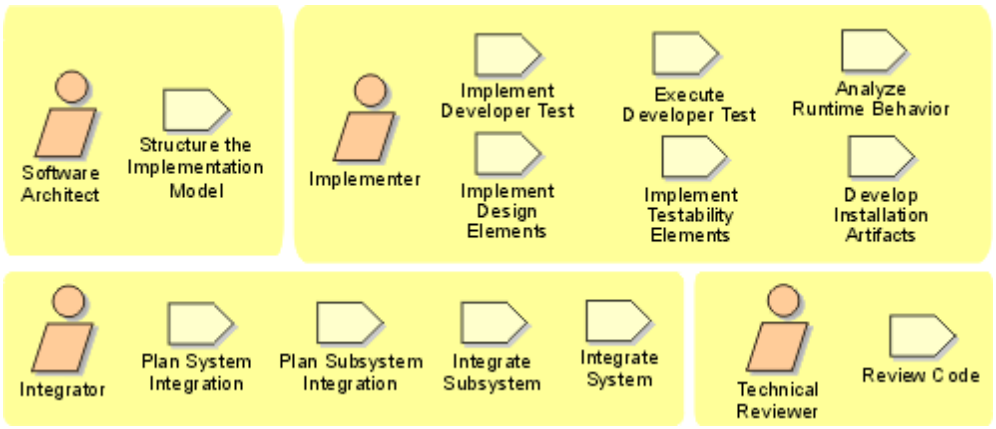
[1] - Evolutionary Prototype
[2] - Role
[3] - Artifact
[4] - Implementer
[5] - Programmer
[6] - Unit Test
[7] - System Integrator
[8] - Build

دیگر نقش‌های مرتبط با دیسیپلین پیاده‌سازی، عبارتند از:

- معمارِ نرم‌افزار[1]، که مسئول تعریف ساختار مدل پیاده‌سازی (لایه‌ها و زیرسیستم‌ها) می‌باشد.
- بازبینی‌کننده‌ی فنی[2]، که پیاده‌سازی‌های انجام شده را به منظور حصول اطمینان از کیفیت و مطابقت با استانداردها، بازبینی می‌نماید.

شکل ۱۵-۵

مجموعه‌ی فعالیت‌های دیسیپلینِ پیاده‌سازی



دستاوردهای[3] اصلی دیسیپلینِ پیاده‌سازی عبارتند از:

- زیرسیستم پیاده‌سازی[4]: مجموعه‌ای از عناصرِ پیاده‌سازی[5] و دیگر زیرسیستم‌های آن می‌باشد. از زیرسیستم پیاده‌سازی به منظور ساختاردهی به مدل پیاده‌سازی و تقسیم آن به بخش‌های کوچک‌تر، استفاده می‌شود.
- عناصر پیاده‌سازی: یک قطعه‌ی پیاده‌سازی شده از نرم‌افزار (کُد[6]، قالب باینری[7]، یا اجرایی[8]) یا یک فایل محتوی اطلاعات (برای مثال یک فایل راه‌انداز یا یک فایل به اصطلاح readme). یک

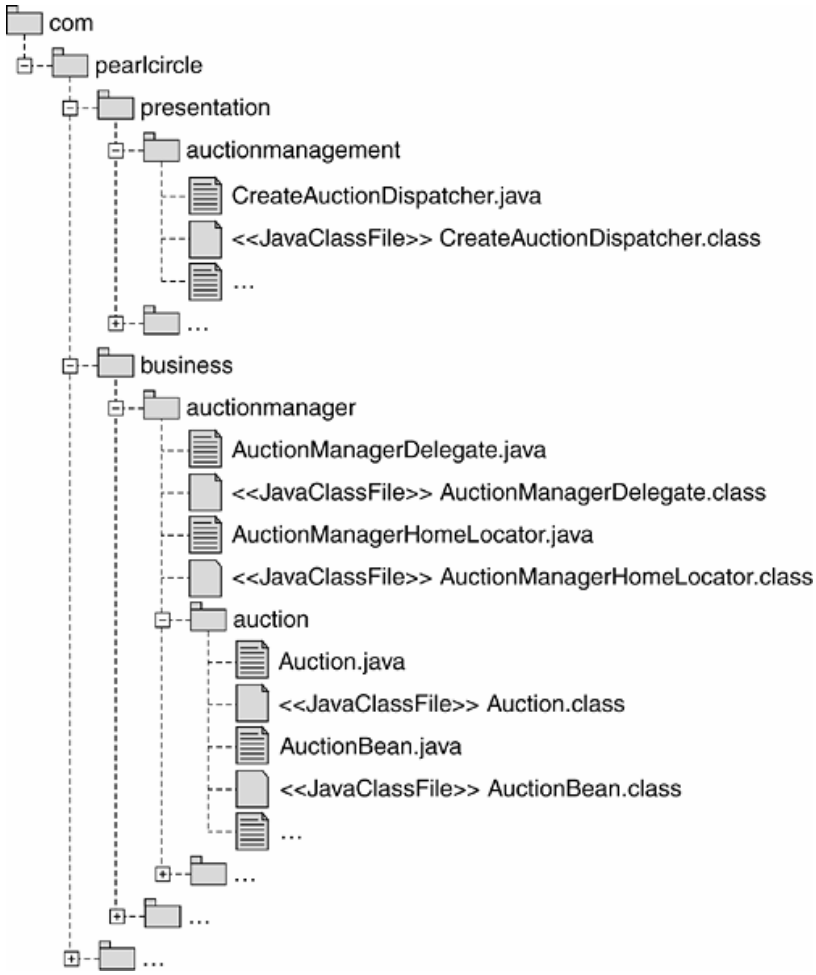
[1] - Software Architect
[2] - Technical Reviewer
[3] - Artifact
[4] - Implementation Subsystem
[5] - Implementation Element
[6] - Source Code
[7] - Binary
[8] - Executable

عنصر پیاده‌سازی می‌تواند تجمیع[1] یکسری از عناصر پیاده‌سازی دیگر باشد، برای نمونه، یک سیستمِ کاربردی[2] شامل چندین فایلِ قابل اِجرا می‌باشد.

- طرحِ یکپارچه‌سازی ایجاد یک نسخه‌ی میانی[3] : این سند شامل تعریف ترتیب و توالی یکپارچه‌سازی عناصر و زیرسیستم‌ها می‌باشد.

شکل ۱۵-۶

نمونه‌ای از ساختار مدلِ پیاده‌سازی



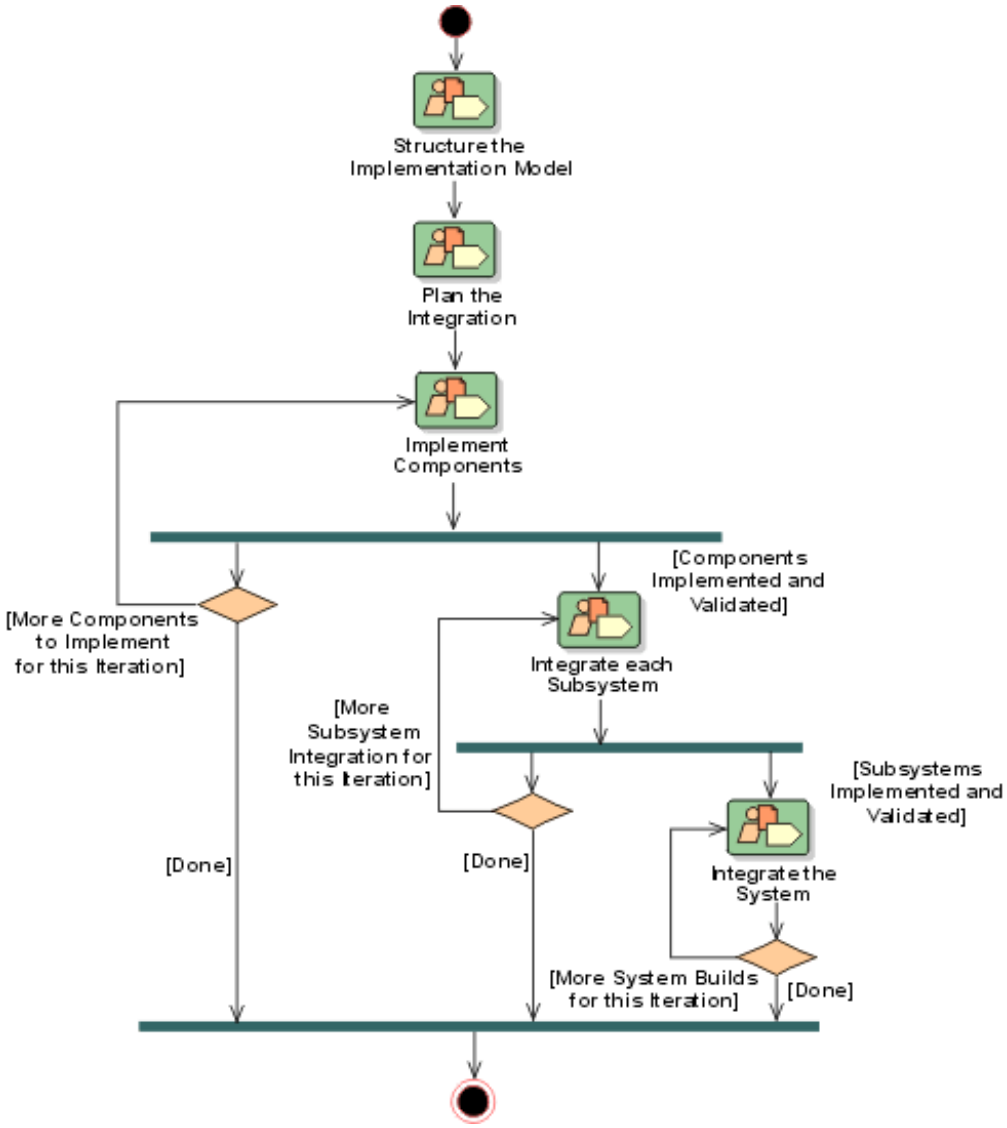
[1] - Aggregation
[2] - Application
[3] - Integration Build Plan

جریان‌کار[1]

جریان‌کارِ دیسیپلینِ پیاده‌سازی مطابق شکل ۱۵-۷ می‌باشد. کارهای اصلی به منظور ساختاردهی به مدل پیاده‌سازی[2]، در اوایلِ فازِ تشریح انجام می‌شود.

شکل ۱۵-۷

جریان‌کار در دیسیپلینِ پیاده‌سازی



[1] - Workflow
[2] - Structure the Implementation Model

هدفِ مجموعه فعالیت‌های مرتبط با این جزء از جریان‌کار[1]، سازماندهی مدلِ پیاده‌سازی است به گونه‌ای که پیاده‌سازی مجموعه مؤلفه‌ها و زیرسیستم‌ها، با کمترین اِختلال و به بهترین شیوه‌ی ممکن، انجام شود. به طور کلی، پیاده‌سازی ارتباط نزدیکی با طراحی دارد، به طوری که همواره باید میان عناصرِ طراحی[2] و عناصر پیاده‌سازی[3] امکان رَدگیری[4] و تناظر، وجود داشته باشد.

چکیده‌ی فصل

در این فصل دیسیپلینِ پیاده‌سازی، اهداف آن و برخی از مفاهیم مرتبط با آن را بررسی نمودیم. مهم‌ترین مباحثی که در این فصل معرفی گردید، عبارتند از:

- یکی از ویژگی‌های کلیدی آر.یو.پی، رویکرد یکپارچه‌سازی تدریجی[5] آن در سرتاسر چرخه‌ی تولید است.
- در طول فاز تشریح (معماری)، یک پیش‌الگوی ساختاری[6] و تکاملی[7] از سیستم ایجاد شده و در پایان فازِ ساخت به سیستم نهایی منجر می‌شود.
- در طول چرخه‌ی تولید، برخی از مسائل مطرح، مانند مانند واسطهای کاربران[8] را می‌توان به طور موازی، با تولید پیش‌الگوهای دور انداختنی، بررسی و حل نمود.

---

[1] - Workflow Detail
[2] - Deign Elements
[3] - Implementation Elements
[4] - Trace
[5] - Incremental Integration
[6] - Structural
[7] - Evolutionary
[8] - User Interface

پرسش‌هایی برای تحقیق و مطالعه‌ی بیشتر

۱. یکی از تکنیک‌های موفق در پیاده‌سازی، روش برنامه‌نویسی زوجی یا Pair Programming می‌باشد، درباره‌ی این روش و جایگاه آن در دیسیپلین پیاده‌سازی تحقیق نمایید.

۲. درباره‌ی نقش پیاده‌ساز (برنامه‌نویس) و چگونگی مشارکت آن در کار تیمی بحث نمایید.

۳. درباره‌ی مهارت‌های یک برنامه‌نویس در تست مؤلفه‌های نرم‌افزاری تحقیق نمایید.

۴. درباره‌ی تفاوت رویکردهای توسعه‌ی تدریجی (Incremental) و رویکردهای تکرارشونده (Iterative) از منظر پیاده‌سازی و یکپارچه‌سازی، بحث نمایید.

منابع و مراجع

[1]. Ivar Jacobson, Grady Booch, James Rumbaugh, (1999). *The Unified Software Development Process*, Reading, MA: Addison-Wesley

[2]. Philippe Kruchten, (2003). *The Rational Unified Process: An Introduction*, Reading, MA: Addison-Wesley.

[3]. Per Kroll, Philippe Kruchten, (2003). *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*, Reading, MA: Addison-Wesley.

[4]. Shalloway, Alan, and James Trott, (2002). *Design Patterns Explained: A New Perspective on Object-Oriented Design*, Reading, MA: Addison-Wesley.

[5]. Rational Software Corporation. (2003) "Rational Unified Process 2003.06," Available at: http://www.rational.com/

[6]. Eeles, Peter, Kelli Houston, and Wojtek Kozaczynski, (2003). *Building J2EE Applications with the Rational Unified Process*, Boston: Addison-Wesley.

[7]. Software Academy. (2006) "Unified Process Knowledge Base," Available at: http://www.unifiedProcess.info/

[8]. Scott W. Ambler, (2000). *The Unified Process Inception Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[9]. Scott W. Ambler, (2000). *The Unified Process Elaboration Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[10]. Scott W. Ambler, (2000). *The Unified Process Construction Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[11]. Scott W. Ambler and Larry L. Constantine, (2001). *The Unified Process Transition Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[12]. Martin Fowler, et al, (1999). *Refactoring: Improving the Design of Existing Code*, Reading, MA: Addison-Wesley.

فصل شانزدهم

دیسیپلینِ تست

مهم‌ترین موضوعات مورد بررسی در این فصل، عبارتند از:

- اهداف دیسیپلینِ تست در آر.یو.پی
- فعالیت‌ها، دستاوردها، و نقش‌های کلیدی
- مفهومِ کیفیت از منظرِ آر.یو.پی
- انواعِ تست

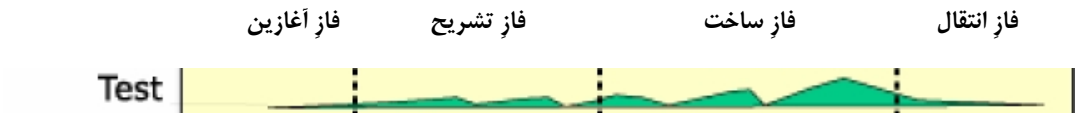
دیسیپلینِ تست

۱۶

در این فصل، با تشریح دیسیپلینِ تست، مفهوم کیفیت را از منظر آر.یو.پی بررسی خواهیم نمود. ارتباط میان دیسیپلینِ تست و سایر دیسیپلین‌های فرایند، از دیگر مباحث مهم مطرح در این فصل می‌باشد. در شکلِ ۱۶-۱ ، نمایی مقطعی از توزیع حجمِ فعالیت‌های مرتبط با این دیسیپلین در طولِ چرخه‌ی تولید نشان داده شده است. همان‌گونه که ملاحظه می‌نمایید، فعالیت‌های تست تقریباً در تمامِ طولِ چرخه توزیع شده است.

شکل ۱۶-۱

نمایی مقطعی از حجم فعالیت‌های مرتبط با دیسیپلینِ تست در طول چرخه‌ی تولید



هدف

از بسیاری جنبه‌ها، دیسیپلینِ تست به نوعی ارائه‌دهنده‌ی خدمتْ به دیگر دیسیپلین‌ها می‌باشد. مجموعه فعالیت‌های این دیسیپلین با بهره‌گیری از اصول زیر، بر ارزیابی یا سنجش[۱] کیفیتِ فراورده[۲] متمرکز می‌باشند:

- یافتن و مستندسازی خطاها[۳]، نقایص[۴]، و مشکلات فراورده.
- توصیه به داشتن مدیریت روی کیفیت درک شده[۵] و مورد قبول از نرم‌افزار.
- سنجش و ارزیابی فرضیات در نظر گرفته شده در طراحی و توصیف نیازمندی‌ها[۶] از طریق ارائه‌ی نمایش‌های عینی[۷] و واقعی (ارائه‌ی نسخه‌ی قابل اجرا[۸] و تست شده از نرم‌افزار).
- تأیید[۹] اینکه فراورده‌ی نرم‌افزاری، مطابق طراحی انجام شده، کار می‌کند.
- تأیید اینکه نیازمندی‌ها به درستی و به صورتی مناسب پیاده‌سازی شده‌اند.

یک تفاوت جالب میان دیسیپلینِ تست و سایر دیسیپلین‌های آر.یو.پی، این است که تست ضرورتاً با پیداکردن و نشان دادن نقاط ضعف و کمبودهای فراورده‌ی نرم‌افزاری در ارتباط می‌باشد. برای دست‌یابی به این هدف، باید نوعی دیدگاه، فلسفه، و ذهنیت خاص و متفاوت با دیسیپلین‌های نیازمندی‌ها، تحلیل و طراحی، و پیاده‌سازی داشته باشیم. در حالی‌که این سه دیسیپلین بر کامل‌بودن[۱۰]، سازگاری و هماهنگی[۱۱]، و صحتِ[۱۲] کار متمرکز می‌باشند، دیسیپلینِ تست بر پیداکردن اشتباهات، ناسازگاری‌ها، و نقایص و کمبودها تأکید دارد.

---

[1] - Evaluation
[2] - Product Quality
[3] - Failures
[4] - Defects
[5] - Perceived Quality
[6] - Requirement Specifications
[7] - Concrete Demonstrations
[8] - Executable
[9] - Validate
[10] - Completeness
[11] - Consistency
[12] - Correctness

معمولاً در مجموعه فعالیت‌های تست، در پی پاسخگویی به سؤالات زیر می‌باشیم:

- چطور این نرم‌افزار را می‌توان شکست؟ (یا اینکه چگونه می‌توان آن را از کار انداخت؟)
- در چه موقعیت‌هایی، این نرم‌افزار مطابق پیش‌بینی عمل نمی‌کند؟

دیسیپلینِ تست، با فرضیات[۱]، ریسک‌ها[۲]، و شرایطِ غیرقطعی[۳] که از کار دیگر دیسیپلین‌ها به ارث رسیده، دست به گریبان است و این نگرانی‌ها را با سنجش[۴] و ارزیابی بی‌طرفانه[۵] و نیز نمایش‌های عینی[۶] برطرف می‌نماید.

بر اساس برآوردها و آمارهای مختلف، تستِ نرم‌افزار به طور معمول در حدود ۳۰ تا ۵۰ درصد از هزینه‌ی تولید نرم‌افزار را به خود اختصاص می‌دهد. بنابراین تعجب نخواهید کرد اگر بدانید که بسیاری بر این عقیده‌اند که یک نرم‌افزار رایانه‌ای قبل از تحویل، به خوبی تست نمی‌شوند. دلایل عمده‌ی این مسأله عبارتند از:

- مشکل بودنِ فعالیت‌های تست نرم‌افزار (خصوصاً اینکه برخی از جنبه‌های کیفی قابل کمّی شدن نیستند)
- به طور معمول، تست بدون داشتن یک متدولوژی و رویکرد مشخص انجام می‌شود که نتیجه‌ی آن به دست آوردن نتایج متفاوت در پروژه‌های و سازمان‌های مختلف می‌باشد. در بسیاری از موارد، موفقیت به مهارت‌های فردی کارشناسانِ تیم تست بستگی دارد.
- نرم‌افزارهای خودکارسازی تست یا استفاده نمی‌شوند و یا در بسیاری از موارد، به درستی مورد بهره‌برداری قرار نمی‌گیرند. این موضوع موجب می‌شود که اولاً، انجام بسیاری از تست‌های ضروری در سیستم‌های امروزی (مانند تستِ کارایی و بارگذاری) امکان‌پذیر نباشند و دوّم اینکه، اِمکان مدیریتِ تست فراهم نمی‌شود و لذا حجمِ عظیم داده‌های تست و نیز نتایجِ حاصل، قابلِ مدیریت نخواهد بود.

---

[1] - Assumptions
[2] - Risks
[3] - Uncertainty
[4] - Evaluation
[5] - Impartial
[6] - Concrete Demonstrations

- قابلیت انعطاف در استفاده[۱] و نیز پیچیدگی[۲] نرم‌افزار، عملاً تکمیل و انجام همه‌ی تست‌ها را به نوعی، به یک هدف غیرممکن، مبدل نموده است. استفاده از استراتژی مناسب و شفاف و نیز ابزارهای مناسب، می‌تواند موجبات بهره‌وری[۳] و مؤثرتر بودنِ[۴] تستِ نرم‌افزار را فراهم آورد.

تجربه‌ی پروژه‌های موفّق بیانگر این موضوع است که داشتنِ یک رویکردِ مستمر[۵] به مقوله‌ی کیفیت که از همان اوایل چرخه‌ی تولید شروع می‌شود، می‌تواند به طور چشمگیری هزینه‌های تکمیل و نگهداری سیستم را کاهش داده و منجر به کاهش ریسکِ استقرار نرم‌افزاری با کیفیتِ پایین[۶]، گردد.

تست در چرخه‌ی توسعه‌ی مبتنی بر رویکردِ تکرارشونده[۷]

از منظر آر.یوپی، تست نه یک فعالیتِ منفرد[۸] و نه یک فاز جداگانه‌ای از پروژه است که در طی آن کیفیت فراورده ارزیابی و یا تضمین شود؛ ما به داشتن بازخوردهای به موقع از روند تکاملِ کیفیت فراورده نیاز داریم و بنابراین تست باید در سرتاسر چرخه‌ی تولید انجام شود و نه فقط در یک فاز خاص از آن!

به این ترتیب، قادر خواهیم بود پیش‌الگوهای ارائه شده در مراحل و فازهای اولیه را نیز تست کنیم. علاوه بر این، ما می‌توانیم تستِ ثبات[۹]، پوشش[۱۰] (جامعیت)، و کارایی[۱۱] معماری را در زمانی که فرصتِ کافی برای رفع نقایص و مشکلات آن با هزینه‌ی معقولی داریم، انجام دهیم. همچنین می‌توانیم فراورده‌ی نهایی را برای اطمینان از آماده‌ی تحویل بودن[۱۲] آن، تست کنیم.

بنابراین، دیگر این مفهوم که تست یک مرحله‌ی پایانی از کار و برای اطمینان از اینکه همه‌چیز به درستی کار می‌کند (روشی که مورد توجه علاقمندان به رویکرد آبشاری است) را باید فراموش نماییم زیرا این

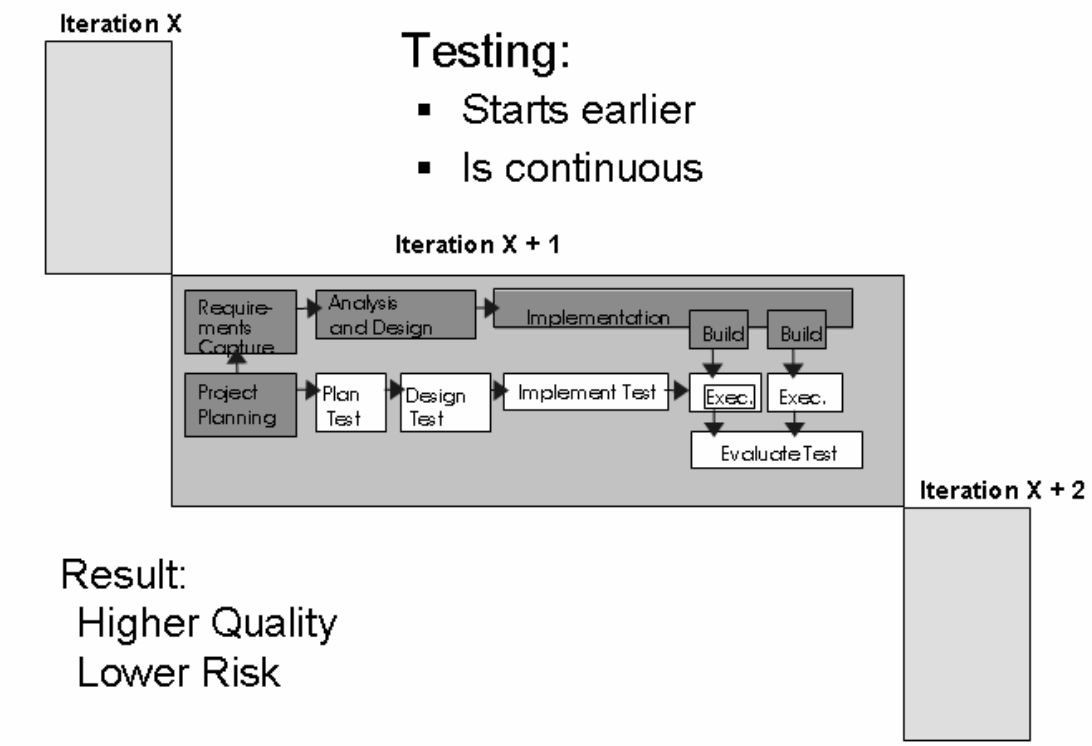
---

[1] - Flexibility of Use
[2] - Complexity
[3] - Productivity
[4] - Effectiveness
[5] - Continuous Approach
[6] - Poor-Quality Software
[7] - Test in Iterative Development
[8] - Single Activity
[9] - Stability
[10] - Coverage
[11] - Performance
[12] - Readiness for Delivery

دیدگاه مهم‌ترین و با ارزش‌ترین امتیاز را که داشتن تست‌های مکرّر در رویکرد تکرارشونده است، نـدارد. ایـن امتیاز و مزیت عبارتست از: فراهم نمودن بازخورد[1] در حالی که هنوز زمان و منابع کـافی بـرای اینکـه کـاری روی آن انجام دهیم، وجود دارد و البته، مهم‌تر از آن، فرصت یادگیری مستمر.

شکل ۱۶-۳

تداومِ تست در طول تکرارهای مختلف و ماهیت آن در درون یک تکرار

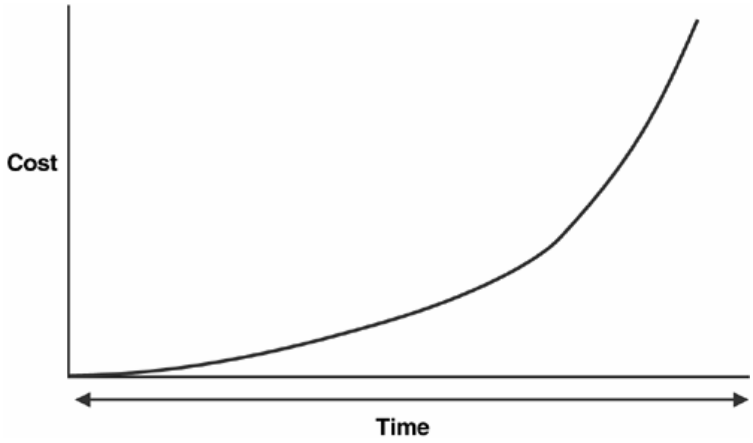


شکل ۱۶-۴ ، بیانگرِ افزایش نمایی هزینه‌ی رفع یک نقص یا مشکل در طول زمان می‌باشد. بنابراین باید تا جایی که اِمکان دارد، تست را زودتر شروع نمود. استفاده از ابزارهای اتوماسیون یا خودکارسازی فعالیتهای تست نیز تأثیر زیادی در بهبودِ و بهینه‌سازی تست دارد. البته باید در انتخاب و تهیه‌ی ابزارهای مناسب، دقت لازم را مبذول داشت. شکل ۱۶-۵ ، بیانگرِ مثالی از تأثیر اتوماسیون بر تست می‌باشد.

[1] - Feedback

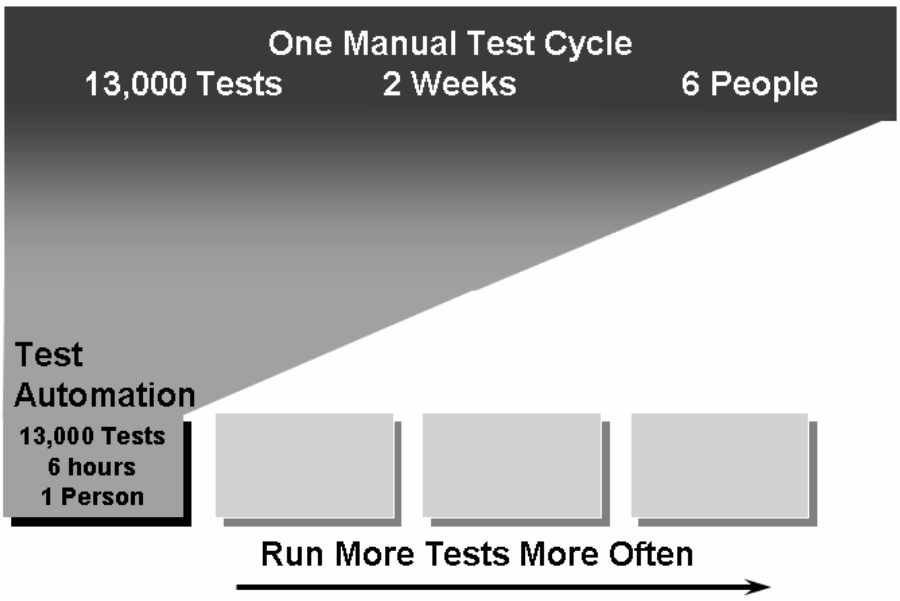
شکل ۱۶-۴

افزایش نمایی هزینه‌ی رفع یک نقص یا مشکل در طول زمان



شکل ۱۶-۵

مثالی از خودکارسازی تست و نتایج حاصل از آن



کیفیت[1]

واژه‌ی کیفیت معمولاً به مفاهیم مختلفی اشاره دارد: اصولاً کیفیت به معنای عدم وجود نقایص[2] می‌باشد و البته مهم‌تر از آن، کیفیت به معنای شایستگی[3] برای یک هدف مطلوب است. باید توجه داشت، چیزی که هیچ نقصی ندارد، ولی کاری را که ما خواستار انجام آن هستیم، انجام نمی‌دهد، به اندازه‌ی یک محصول ناقص، بِلااستفاده است. بنابراین، کیفیت مقوله‌ی بسیار پیچیده‌ای است، امّا برای بسیاری از اَبعاد آن معیارهایی وجود دارد که شناخت آنها ضروری است. برخی از این معیارها که در میان بسیاری از پروژه‌ها عمومیت دارند، عبارتند از:

- قابلیتِ اعتماد[4] : سیستم دارای عملکردی قابل پیش‌بینی[5] می‌باشد. به عبارت دیگر، سیستم نسبت به ازکارافتادن[6] در حین اجرا، دارای مقاومت می‌باشد: مثلاً با کمبود حافظه مواجه نمی‌شود یا قفل نمی‌کند.
- وظیفه‌مندی[7] : سیستم قادر است همه‌ی مواردِکاربرد مورد نیاز یا رفتارهای مطلوب را بر حسب مورد، اجرا نماید.
- کارایی[8] : اجرا و پاسخ‌دهی نرم‌افزار و سیستم در محدوده‌ی یک بازه‌ی زمانی مشخص بوده و هنگامی که در شرایط عملیاتی واقعی[9] (مانند بار[10]، استرس[11]، و بازه‌ی طولانی یک عملیات[12]) قرار می‌گیرد، به خوبی صورت می‌پذیرد. هدف اصلی تستِ کارایی عبارت است از: اطمینان از اینکه وظیفه‌مندی‌های سیستم، ضمنِ تطابق با نیازمندی‌های غیروظیفه‌مندی[13]، قابل ارائه می‌باشند.

---

1 - Quality
2 - Absence of Defects
3 - Fitness
4 - Reliability
5 - Predictable
6 - Failure
7 - Functionality
8 - Performance
9 - Real-World Operational Characteristics
10 - Load
11 - Stress
12 - Lengthy Periods of Operation
13 - Non-Functional Requirements

- قابلیت استفاده[۱] : استفاده از نرم‌افزار برای همه‌ی کاربران نهایی آن آسان می‌باشد. در اینجا عمدتاً عواملی مانند فاکتورهای انسانی، زیبایی‌شناسی و مانند آن، مطرح می‌باشد.

همانگونه که ملاحظه نمودید، معیارهای کیفی در تطابق با مهم‌ترین نیازمندی‌های غیرِ وظیفه‌مندی[۲] می‌باشد. برای دستیابی به هر یک از ابعادِ کیفیت، اجرای یک یا چند نوع تست، در سطوح مختلف، ضروری است. علاوه بر این، باید توجه داشت که معیارهای دیگری هم برای کیفیت مطرح می‌باشد که بیشتر به نظر اشخاص بستگی داشته[۳] و کمتر حالت کمّی دارند، از جمله: قابلیتِ پشتیبانی[۴]، قابلیتِ نگهداری[۵]، قابلیت گسترش[۶]، و انعطاف‌پذیری[۷]. البته بهتر است که تا حد امکان، این معیارها نیز کمّی شوند.

همواره به یاد داشته باشید که تولید یک فراورده‌ی با کیفیت[۸]، مسئولیت همه‌ی اعضای تیم پروژه می‌باشد، نه مسئولیت بخش تضمین کیفیت[۹] سازمان یا حتی بدتر از آن یک سازمان دیگر! اگر کیفیت از آغاز در بطنِ همه‌ی فعالیت‌ها قرار نگیرد، با هیچ ترفندی تحقق واقعی آن ممکن نخواهد بود.

نقشِ تست، تضمین کیفیت نیست، بلکه نقش اصلی تست عبارتست از ارزیابی[۱۰] کیفیت و فراهم نمودنِ بازخوردهایی در طول زمان، به منظور رفع مشکلات و مسائلِ کیفی با هزینه‌ی زمانی و مالی مقرون‌به‌صرفه. بنابراین، نقش انجام دهنده‌ی تست[۱۱] نیز ارزیابی کیفیت و ارائه‌ی بازخوردهای مناسب است.

لازم به ذکر است که ارزیابی کیفیتِ فرایند تولید، ارتباطی به دیسیپلین تست ندارد. این موضوع نگرانی دیسیپلین محیط[۱۲] و نقشِ مهندس فرایند[۱۳] در این دیسیپلین می‌باشد.

---

[1] - Usability
[2] - Non-Functional Requirement
[3] - Subjective
[4] - Supportability
[5] - Maintainability
[6] - Extensibility
[7] - Flexibility
[8] - Quality Product
[9] - Quality Assurance
[10] - Assessment
[11] - Tester
[12] - Environment Discipline
[13] - Process Engineer

انواعِ تست

انواع مختلفی از تست وجود دارد که هر کدام بر یکی از اهداف و مقصودهای تست، متمرکز بوده و یک خصیصه[۱] یا ویژگی[۲] خاص از نرم‌افزار را تست نموده، یا اینکه سعی در یافتن دسته‌ی خاصی از خطاها یا نقایص دارند. از آن‌جایی‌که تست در سرتاسر چرخه‌ی تولید اجرا می‌شود، نرم‌افزاری که مورد تست و آزمون قرار می‌گیرد، ممکن است که یک تکه کُد، واحدهای یکپارچه‌شده[۳]، یا کلِ سیستم باشد.

برخی از مهم‌ترین انواع تست عبارتند از:

- تستِ محک[۴] : ارزیابی کارایی یک هدفِ تست[۵] در مقابل یک استاندارد یا معیارِ سنجش[۶] موجود
- تستِ پیکربندی[۷] : ارزیابی چگونگی عملکردِ یک هدفِ تست در پیکربندی‌های مختلف (سخت‌افزاری و نرم‌افزاری)
- تستِ وظیفه‌مندی[۸] : ارزیابی چگونگی اجرای مواردِکاربردِ مطلوب توسط وظیفه‌مندی‌های مختلفِ هدفِ تست
- تستِ نصب[۹] : ارزیابی چگونگی نصب هدفِ تست روی پیکربندی‌های مختلف و تحت شرایط مختلف مانند کمبود فضای دیسک
- تستِ انسجام[۱۰] (یکپارچگی) : ارزیابی قابلیتِ اعتماد[۱۱]، استحکام[۱۲]، و مقاومت[۱۳] هدفِ تست در شرایط از کار اُفتادنِ سیستم[۱۴] در زمان اجرا

---

[1] - Characteristics
[2] - Attribute
[3] - Integrated Units
[4] - Benchmark Test
[5] - Target-of-Test
[6] - Measurement
[7] - Configuration Test
[8] - Functional Test
[9] - Installation Test
[10] - Integrity
[11] - Reliability
[12] - Robustness
[13] - Resistance
[14] - System Failure

- تستِ بارگذاری[۱] : ارزیابی قابل قبول بودنِ[۲] کارایی هدفِ تست در شرایطِ متغییر عملیاتی[۳]، مانند تعداد کاربران و تعداد تراکنش‌ها[۴]، در حالی‌که پیکربندی ثابت است.
- تستِ کارایی[۵] : ارزیابی قابلِ قبول بودنِ کارایی هدفِ تست با استفاده از پیکربندی‌های مختلف و در حالی‌که شرایط عملیاتی ثابت می‌ماند.
- تستِ استرس[۶] : ارزیابی کارایی هدفِ تست در شرایط غیرمعمول[۷] یا افراطی[۸] و بحرانی، مانندِ کم‌شدن منابع یا افزایش بیش از حدِ کاربران.

سطوحِ تست

همانگونه که در ابتدای فصل اشاره گردید، تست در تکرارهای مختلف و به منظورهای متفاوتی اجرا می‌شود. به طور کلی در یک پروژه‌ی نرم‌افزاری، چهار سطح از تست قابل تصوّر است. این چهار سطح عبارتند از:

- تست واحد[۹] یا تست اجزاء : تستِ کوچکترین عنصر قابلِ تست از سیستم (معمولاً همزمان با پیاده‌سازی آن عناصر)
- تست یکپارچه‌سازی[۱۰] : تست واحدها، مؤلفه‌ها، یا زیرسیستم‌های یکپارچه‌شده
- تست سیستم[۱۱] : تست نرم‌افزارِ کاربردی[۱۲] و سیستمِ کامل شده (یا تست چندین سیستمِ کاربردی)
- تستِ پذیرش[۱۳] : تستِ نرم‌افزار کاربردی و سیستمِ کامل شده به وسیله‌ی کاربران نهایی[۱۴] یا نمایندگان‌شان، به منظور تعیین آمادگی برای استقرار[۱۵]

---

[1] - Load Test
[2] - Acceptability
[3] - Under Varying Operational Conditions
[4] - Transaction
[5] - Performance Test
[6] - Stress Test
[7] - Abnormal
[8] - Extreme
[9] - Unit Test
[10] - Integration Test
[11] - System Test
[12] - Application
[13] - Acceptance Test
[14] - End User
[15] - Deployment

توجه داشته باشید که سطوح مختلفِ تست، با میزان تأکید و تمرکز متفاوتی به صورت متوالی[۱] و یا موازی[۲]، در طول چرخه‌ی تولید رُخ می‌دهند. یک پیش‌الگوی[۳] ارائه شده در فاز آغازین که هدف آن ارزیابی سودمند بودنِ پروژه می‌باشد، مورد تست پذیرش قرار می‌گیرد. یک پیش‌الگوی معماری که در طولِ فازِ تشریح بوجود آمده، در سطوح مختلف تست‌های یکپارچه‌سازی و نیز تستِ سیستم، ارزیابی می‌گردد.

نقش‌ها[۴] و دستاوردها[۵]

اصلی‌ترین نقش‌هایی که در دیسیپلینِ تست مشارکت دارند، عبارتند از:

- مدیرِ تست[۶] : مسئولیت کلّی موفقیتِ مجموعه فعالیت‌های تست را برعهده دارد. برنامه‌ریزی و مدیریت منابع و نیز حل مُعضلات و موانع انجام فعالیت‌های تست، از جمله وظایف این نقش است.
- تحلیل‌گر تست[۷] : مسئولیت شناسایی و تعریف تست‌های لازم، نظارت بر پیشرفت و نتایج تفصیلی اجرای تست‌ها در هر چرخه‌ی تست[۸] و ارزیابی کیفیت تجربه شده در فعالیت‌های تست را بر عهده دارد. این نقش، مسئولیت بازنمایی و انعکاس مناسب نیازها و خواسته‌های ذینفعانی را که نماینده‌ی مستقیم یا قانونی در پروژه ندارند نیز بر عهده می‌گیرد.
- طراحِ تست[۹] : مسئولیت تعریف رویکردِ تست[۱۰] و اطمینان از صحتِ پیاده‌سازی آن را بر عهده دارد. از دیگر وظایف این نقش، می‌توان به این موارد اشاره نمود: شناسایی تکنیک‌ها، ابزارها، و راهنمایی‌های[۱۱] مناسب برای پیاده‌سازی تست‌های لازم و ارائه‌ی رهنمودهایی برای استفاده‌ی مناسب از منابع مورد نیاز برای انجام فعالیت‌های تست.

---

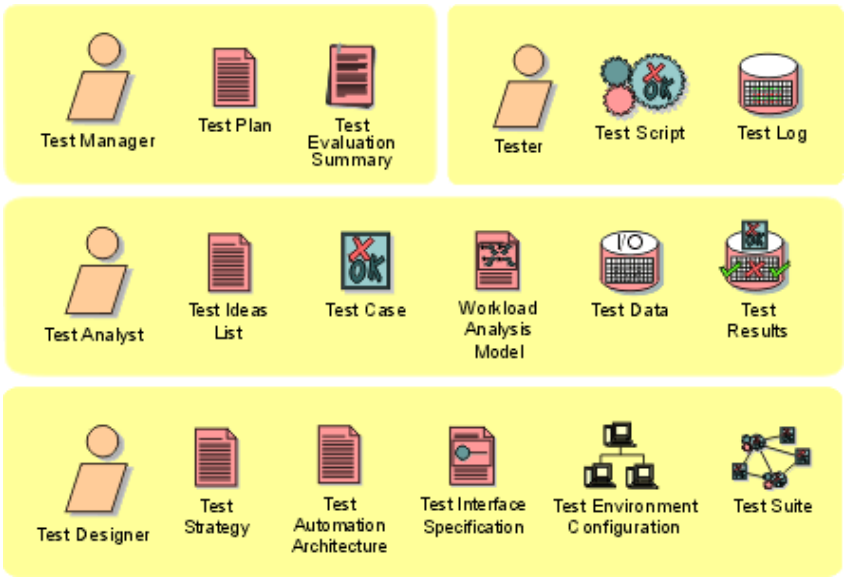
1 - In Sequence
2 - Parallel
3 - Prototype
4 - Roles
5 - Artifacts
6 - Test Manager
7 - Test Analyst
8 - Test Cycle
9 - Test Designer
10 - Test Approach
11 - Guidelines

- انجام‌دهنده‌ی تست[1] : مسئول اجرای تست‌های سیستم می‌باشد. این نقش، وظایف دیگری مانند تنظیم و اجرای تست‌ها، ارزیابی روند اجرای تست[2]، ارزیابی نتایج اجرای تست[3]، و نگهداری درخواست‌های تغییر[4] را انجام می‌دهد.

در شکل ۱۶-۶ ، مجموعه‌ی نقش‌ها و دستاوردهای دیسیپلینِ تست، نشان داده شده است.

شکل ۱۶-۶

نقش‌ها و دستاوردها در دیسیپلین تست

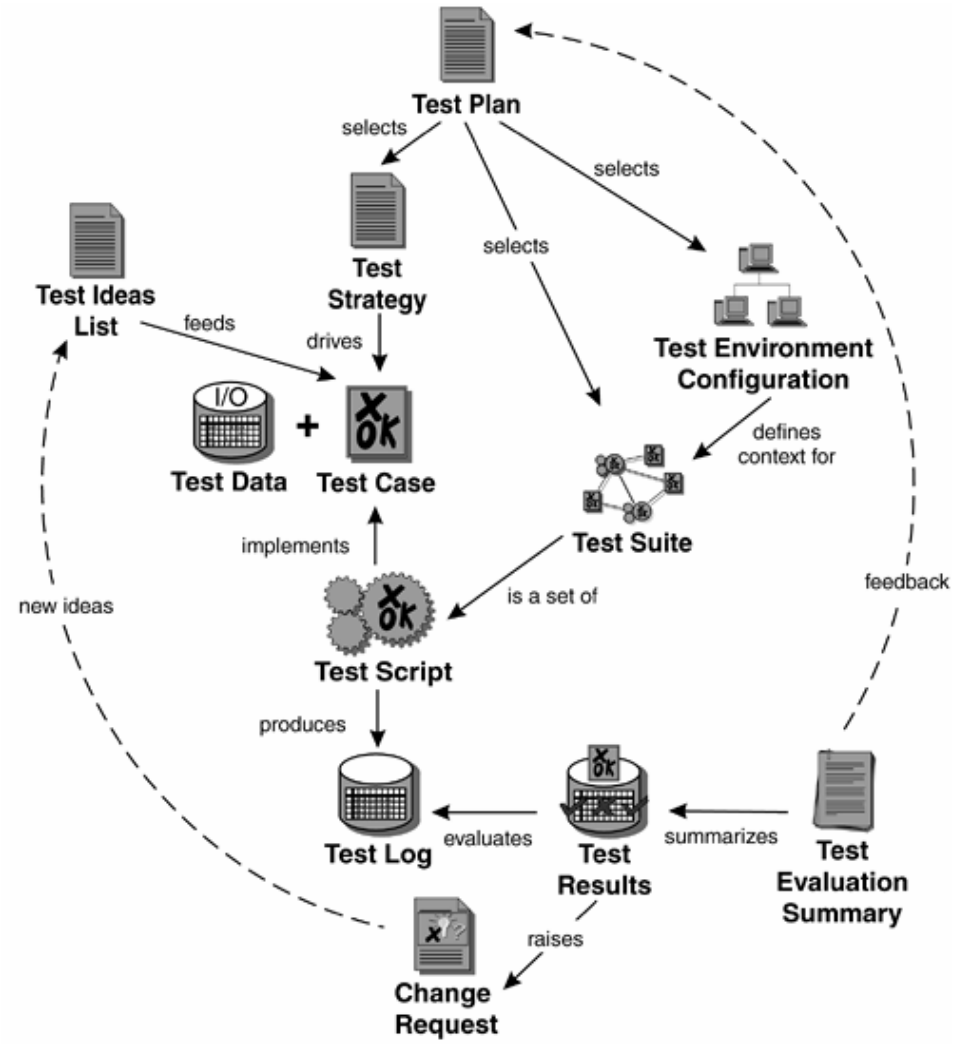


شکل ۱۶-۷ ، دستاوردهای مختلف تست و ارتباطشان را نشان می‌دهد.

[1] - Tester
[2] - Evaluating Test Execution
[3] - Assessing the Results of Testing
[4] - Logging Change Requests

شکل ۱۶-۷

دستاوردها و ارتباط میان آنها در دیسیپلینِ تست



برخی از مهم‌ترین دستاوردهای[1] دیسیپلین تست، عبارتند از:

- برنامه‌ی تست[2] : شامل اطلاعاتی درباره‌ی هدفِ انجام تست در یک بازه‌ی زمانی مشخص از پروژه می‌باشد. استراتژی‌هایی مورد استفاده، منابع ضروری برای پیاده‌سازی و اجرای تست‌ها، و نیز نوعِ پیکربندی‌های خاص برای انجام تست، از جمله مهم‌ترین اطلاعات موجود در برنامه تست است.

[1] - Artifact
[2] - Test Plan

- لیستِ ایده‌های تست[۱] : این لیست که توسط تحلیل‌گرِ تست نگهداری به‌روز رسانی می‌شود، شامل لیستی از ایده‌های مطرح برای تست می‌باشد.
- موردِ تست[۲] : متناظر با یک موردِکاربرد و برای تست آن تعریف می‌شود.
- اسکریپتِ تست[۳] : رویه‌های[۴] دستی و یا خودکاری می‌باشند که تست‌کننده از آن‌ها برای اجرای تست‌ها استفاده می‌نماید.
- مجموعه‌ی تست[۵] : دربرگیرنده‌ی مجموعه‌ای از تست‌ها و اسکریپت‌ها در قالبی شبیه یک بسته[۶] می‌باشد.
- مدلِ تحلیلِ بارِ کاری[۷] : نوع خاصی از موردِ تست می‌باشد که به منظور تستِ کارایی استفاده می‌شود.
- نسخه‌ی ثبت‌شده‌ی تست[۸] : داده‌ی خام جمع‌آوری شده در حین اجرای مجموعه تست‌ها
- نتیجه‌ی تست[۹] : با فیلتر و تحلیل نسخه‌های ثبت‌شده‌ی تست، به دست می‌آید.
- چکیده‌ی نتایج ارزیابی تست[۱۰] : به عنوان بخش از ارزیابی یک تکرارِ پروژه[۱۱] و ارزیابی متناوب وضعیت[۱۲]، تولید می‌شود.

---

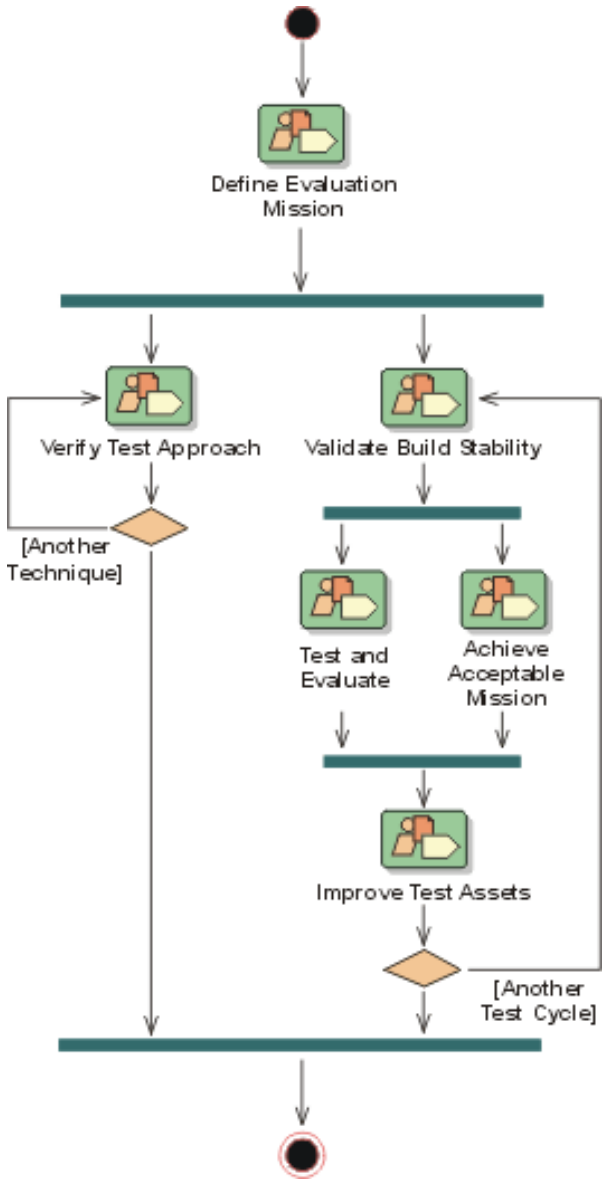
[1] - Test Idea List
[2] - Test Case
[3] - Test Script
[4] - Procedures
[5] - Test Suite
[6] - Package
[7] - Workload Analysis Model
[8] - Test Log
[9] - Test Result
[10] - Test Evaluation Summary
[11] - Iteration Assessment
[12] - Periodic Status Assessment

جریانِ کار[1]

شکل ۱۶–۸ ، نشان‌دهنده‌ی جریانِ کارِ دیسیپلینِ تست[2] می‌باشد.

شکل ۱۶–۸

جریان‌کارِ دیسیپلینِ تست



[1] - Workflow
[2] - Testing Workflow

جریانِ‌کارِ تست، مکانیزمی برای ارائه بازخورد پروژه[1] فراهم می‌کند که بر اساس آن می‌توان کیفیت را سنجید و نقایص و مشکلات را شناسایی نموده و پیش از آنکه خیلی دیر شود و دیگر نتوان روی آنها کار کرد، آنها را از بین ببریم. فعالیت‌های تست از همان ابتدای پروژه، با برنامه‌ریزی تست و انجام ارزیابی (حتی در فاز آغازین) شروع شده و در سرتاسر پروژه ادامه می‌یابد.

چکیده‌ی فصل

این فصل به بررسی دیسیپلینِ تست و ملاحظاتِ مرتبط با آن اختصاص داشت. مهم‌ترین مواردی که در این فصل مورد بررسی قرار گرفت، عبارتند از:

- با انجام تست، ارزیابی کیفیتِ فراورده‌ی در حال تولید، امکان‌پذیر می‌شود.
- تست، یک فرایند تکرارشونده[2] می‌باشد که در تمام فازهای چرخه‌ی تولید فراورده صورت می‌پذیرد. به این ترتیب، از همان ابتدای پروژه، امکان ارائه‌ی بازخوردهای مناسب و به موقع از کیفیتِ فراورده فراهم می‌شود. تست، فعالیتی نیست که تنها در انتهای چرخه‌ی تولید و برای اِثبات یا رد سیستمِ تولید شده، انجام شود، بلکه جزء حیاتی ارائه بازخورد و سنجش کیفیت در طول زمان انجام پروژه (در طول چرخه‌ی تولید[3]) می‌باشد.
- کیفیت، مسئولیت همه‌ی افراد است. کیفیت به وسیله‌ی بخشِ تست یا تضمین کیفیت به وجود نمی‌آید. در یک فرایند تکرارشونده، تست، بازخوردهایی از سنجش کیفیت به سازمانِ تولید کننده ارائه می‌نماید و به این وسیله، سازمان قادر خواهد بود به طور مستمر، کیفیت سیستم را بهبود بخشد.

[1] - Project Feedback Mechanism
[2] - Iterative Process
[3] - Development Cycle (Project Lifecycle)

پرسش‌هایی برای تحقیق و مطالعه‌ی بیشتر

۱. درباره‌ی توسعه مبتنی بر تست[1] و Test First Deign تحقیق نمایید.

۲. چگونگی به دست آوردن مواردِ تست[2] از روی مواردِ کاربرد[3] را بررسی نمایید.

۳. در ارتباط با مفهوم Good Enough Quality و ارتباط آن با فلسفه و دیدگاه تست در آر.یو.پی، تحقیق نمایید.

۴. درباره‌ی انواع ابزارهای تست از منظر کاربردشان در فرایند تولید، تحقیق نمایید.

[1] - Test Driven Development
[2] - Test-Case
[3] - Use-Case

منابع و مراجع

[1]. Ivar Jacobson, Grady Booch, James Rumbaugh, (1999). *The Unified Software Development Process*, Reading, MA: Addison-Wesley

[2]. Marnie L. Hutcheson, (2003). *Software Testing Fundamentals: Methods and Metrics.* Reading, John Wiley & Sons.

[3]. Philippe Kruchten, (2003). *The Rational Unified Process: An Introduction*, Reading, MA: Addison-Wesley.

[4]. Per Kroll, Philippe Kruchten, (2003). *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*, Reading, MA: Addison-Wesley.

[5]. John D. McGregor, David A. Sykes, (2001). *A Practical Guide to Testing Object-Oriented Software*, Reading, MA: Addison-Wesley

[6]. Rational Software Corporation. (2003) "Rational Unified Process 2003.06," available at: http://www.rational.com/

[7]. Software Academy. (2006) "Unified Process Knowledge Base," Available at: http://www.unifiedProcess.info/

[8]. Scott W. Ambler, (2000). *The Unified Process Inception Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[9]. Scott W. Ambler, (2000). *The Unified Process Elaboration Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[10]. Scott W. Ambler, (2000). *The Unified Process Construction Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[11]. Scott W. Ambler and Larry L. Constantine, (2001). *The Unified Process Transition Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[12]. Martin Fowler, et al, (1999). *Refactoring: Improving the Design of Existing Code*, Reading, MA: Addison-Wesley.

فصل هفدهم

دیسیپلینِ اِستقرار

مهم‌ترین موضوعات مورد بررسی در این فصل، عبارتند از:

- اهداف دیسیپلینِ استقرار در آر.یو.پی
- فعالیت‌ها، دستاوردها، و نقش‌های کلیدی
- انواعِ روش‌های استقرارِ فراورده

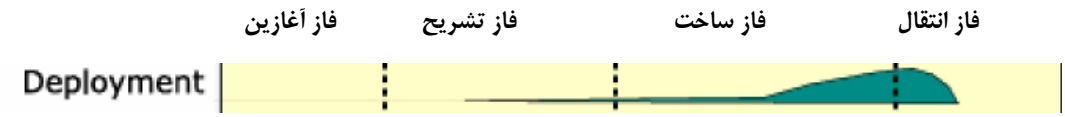
دیسیپلینِ اِستقرار

۱۷

در این فصل، دیسیپلین استقرار و مجموعه فعالیت‌ها، نقش‌ها، و دستاوردهای مرتبط با آن را بررسی خواهیم کرد. هدفِ اصلی مجموعه‌ی فعالیت‌های این دیسیپلین، عبارتست از استقرارِ نرم‌افزار در محیط عملیاتی پیش‌بینی شده و فراهم نمودن دستاوردهای کمکی مورد نیاز برای اطمینان از بکارگیری موفقِ فراورده‌ی نرم‌افزاری توسط تمامی کاربران نهایی آن.

شکل ۱۷-۱

نمایی مقطعی از دیسیپلینِ استقرار در طول فازهای چرخه‌ی تولید



هـدف

بعضی اوقات، افرادِ تیمِ تولیدِ نرم‌افزار با تکمیلِ پیاده‌سازی سیستم، پیروزی و موفقیت خود را اعلام می‌نمایند؛ اغلب فراموش می‌شود که رضایتِ مشتری، تعیین کننده‌ی پایان موفقِ تولید یک فراورده است، نه فقط داشتن یک کامپایل خوب!

هدفِ مجموعه‌ی فعالیت‌های اِستقرار این است که محصولِ نرم‌افزاری تکمیل‌شده را در شرایط عملیـاتی و در اختیار کاربران آن قرار دهد. در واقع، هدفِ اصلی مجموعه‌ی فعالیت‌های این دیسیپلین، تبـدیل سیـستم تولید شده به فراورده‌ای با قابلیتِ به اصطلاح Self-Supportability می‌باشد. بر اساسِ این قابلیت، همـه‌ی کاربرانِ نهایی سیستم اعم از کاربران عادی، مدیران سیستم، و یا مسئولین نگهداری آن، باید بتوانند بدون نیاز به حضور تولیدکنندگان با سیستم به راحتی کار کنند.

دیسیپلینِ استقرار شامل انواع مختلفی از فعالیت‌ها می‌باشد که عبارتند از:

- تستِ نرم‌افزار در محیطی با شرایط عملیاتی نهایی[۱] آن (تست‌های بِتـا[۲])
- بسته‌بندی[۳] نرم‌افزار برای تحویل[۴]
- توزیع[۵] نرم‌افزار
- نصب[۶] و راه‌اندازی نرم‌افزار
- آموزش کاربران نهایی و نیز آموزش نیروهای بازاریابی و فروش
- انتقال و جایگزینی نرم‌افزار موجود و یا تبدیل بانک اطلاعاتی

بر حسب اندازه‌ی پروژه، ماهیت آن، روش تحویل، و نیز زمینه‌ی کسب‌وکـارِ فـراورده[۱]، چگـونگی انجـامِ فعالیت‌های این دیسیپلین در شرکت‌ها و بخش‌های مختلفِ صنعت نرم‌افزار، متفاوت می‌باشد.

---

[1] - Final Operational Environment
[2] - Beta Test
[3] - Packaging
[4] - Delivery
[5] - Distributing
[6] - Installing

انواع روش‌های اِستقرار

رویکرد بیان شده در آر.یو.پی را می‌توان برای طیف وسیعی از پروژه‌های تولید نـرم‌افـزار بکـار گرفـت. بـرای بررسی ملاحظات مختلف مرتبط با اِستقرار فراورده‌ی نرم‌افزاری، سه نوع متداول از روش‌های اِستقرار فراورده را مختصراً بررسی می‌نماییم:

۱- نرم‌افزارهای سفارشی[۲]

۲- بسته‌های نرم‌افزاری[۳]

۳- دریافت و بارگذاری[۴] از طریق اینترنت

تفاوت کلیدی میان این سه نوع روش اِستقرار، در رابطه با میزان درگیر شـدن تولیدکننـدگان فـراورده‌ی نرم‌افزاری در بسته‌بندی، توزیع، و نیز روش یـادگیری چگـونگی اسـتفاده از سیـستم توسـط کـاربران نهـایی می‌باشد.

سیستم‌هایی که به صورت سفارشی ساخته می‌شوند، تا حد زیادی منحصر به فرد بوده و حتّـی در برخـی موارد، دارای سخت‌افزارِ خاص و سفارشی می‌باشند. مشتریانِ چنین سیستم‌هایی را اغلب، بخش‌های دولتـی، صنایع حمل و نقل، مخابرات، بانک‌ها، و مؤسسات بزرگ تجاری تشکیل می‌دهنـد. بـسیاری از سیـستم‌هـای بزرگ سازمانی، نظیر ای.آر.پی[۵] نیز با وجودی‌که به صورت یک یا چند بـسته‌ی آمـاده وجـود دارنـد، معمـولاً مستلزم سفارشی‌سازی برای مشتری می‌باشند.

تفاوت کوچکی میان نحوه‌ی درگیر شدن تیمِ تولیدکننده در نصب و راه‌اندازی نرم‌افزارهـای بـسته‌بنـدی شده و نرم‌افزارهای قابل دریافت از طریق اینترنت وجود دارد. در هر یک از این دو مورد، نصب کننده، معمولاً از کاربران نهایی سیستم می‌باشد. تفاوت در ساز و کارِ تحویل و نیز در چگونگی درگیـر شـدن تولیدکننـده‌ی

---

[1] - Business Context
[2] - Custom-Build
[3] - Shrink-Wrapped
[4] - Download
[5] - ERP

نرم‌افزار در ایجاد بسته‌های مورد نیاز و توزیع فراورده در حالت اول و یا راه‌اندازی وب‌سایت برای حالت دوم، می‌باشد.

دیسیپلینِ اِستقرار در طول فرایند

با توجه به اهداف این دیسیپلین، اوج فعالیت‌های آن در طولِ فاز انتقال[۱] صورت می‌پذیرد. اما در فازهای دیگر فرایند نیز، فعالیت‌هایی از این دیسیپلین اجرا می‌شود.

استقرارِ موفق و در نتیجه موفقیتِ کلّی فعالیت‌های تولید، در قالبِ تمایل مشتری به استفاده از نرم‌افزار جدید، تعریف می‌شود. به عبارت دیگر، در طول فازهای مختلفِ فرایند، مجموعه‌ی فعالیت‌ها، نقش‌ها، و دستاوردهای دیسیپلینِ استقرار با هدف جلبِ رضایت و اطمینان خاطرِ مشتری و استفاده‌کنندگان نهایی سیستم، برنامه‌ریزی می‌گردد. بیشتر فعالیت‌های استقرار برای برآورده نمودن هدفِ اصلی فاز انتقال، یعنی اطمینان از انتقال آرام و تدریجی کاربر به استفاده از نرم‌افزار و دستیابی به قابلیت خودیاری[۲]، انجام می‌شوند.

کاربران در طی فاز انتقال، استفاده از سیستم را تجربه نموده و با چگونگی بکارگیری آن در محیط واقعی و عملیاتی آشنا می‌شوند. نصب و راه‌اندازی‌های آزمایشی[۳] یا تست‌های بِتا در یک‌سری از تکرارهای[۴] این فاز، فرصت مناسبی برای انجام آخرین تنظیمات و بهبودها فراهم می‌نماید.

همانند دیگر فعالیت‌های برنامه‌ریزی در پروژه، برنامه‌ریزی اِستقرار را می‌توان از همان اوایل پروژه آغاز نمود. در این صورت، می‌توان استراتژی‌های مناسبی برای آماده‌سازی مشتری اتخاذ نموده و نیز منابع مورد نیاز برای تحویلِ محصولِ تست شده و سایر دستاوردهای مورد نیاز برای پشتیبانی کاربر نهایی را تخصیص داد. کار روی دستاوردهای خاصی از دیسیپلین استقرار، مانند دست‌نامه‌های کاربران[۵] و دست‌نامه‌های آموزشی را می‌توان بلافاصله بعد از تثبیت معماری، در پایان فاز تشریح که معماری و نیز نیازمندی‌های نرم‌افزار مستحکم می‌شوند، آغاز نمود.

---

[1] - Transition
[2] - Self-supporting
[3] - Trial Installation
[4] - Iteration
[5] - User Manual

نقش‌ها[1] و دستاوردها[2]

نقش‌هایی که معمولاً در مجموعه فعالیت‌های دیسیپلین استقرار درگیر می‌شوند، عبارتند از:

- مدیر استقرار[3] : مسئول برنامه‌ریزی و سازماندهی مجموعه فعالیت‌های اِستقرار می‌باشد. این نقش، مسئولیت برنامه‌ی دریافتِ بازخورد[4] از تست‌های بِتا و اطمینان از بسته‌بندی مناسبِ فراورده را نیز بر عهده دارد.
- مدیر پروژه[5] : واسط اصلی با مشتری است و مسئولیت تأیید عملیات اِستقرار بر اساس نتایج حاصل از ارزیابی تست‌های انجام شده و بازخوردهای دریافت شده و نیز پذیرش مشتری را برعهده دارد.
- نویسنده‌ی فنی[6] : مسئول برنامه‌ریزی و تولید مطالب و اطلاعات پشتیبانی برای کاربر نهایی[7] می‌باشد.
- تولیدکننده‌ی دوره‌ی درسی[8] : مسئول برنامه‌ریزی و تولید مطالب آموزشیِ مرتبط با فراورده می‌باشد.
- هنرمند گرافیست[9] : مسئول تمام کارهای گرافیکی مرتبط با فراورده (مانند کاتالوگ، لوگو، و یا طراحی‌های گرافیکی درون نرم‌افزار) می‌باشد.
- انجام‌دهنده‌ی تست[10] : مسئولیت اجرای تست‌های پذیرش[11] و نیز اطمینان از کافی بودن تست‌ها انجام شده روی محصول را بر عهده دارد.
- پیاده‌ساز[12] (برنامه‌نویس) : اسکریپت‌های نصب و نیز سایر دستاوردهای مربوط به آن را که کاربر نهایی را در نصب و راه‌اندازی محصول یاری می‌دهد، ایجاد می‌نماید.

---

1 - Roles
2 - Artifacts
3 - Deployment Manager
4 - Feedback
5 - Project Manager
6 - Technical Writer
7 - End-user Support Material
8 - Course Developer
9 - Graphic Artist
10 - Tester
11 - Acceptance Test
12 - Implementer

دستاوردهای استقرار[1]

همانطور که پیش از این نیز بیان گردید، اِستقرار بر حسب انواع پروژه‌های مختلف، ممکن است معنا و قالب خاصی به خود بگیرد. در طیفِ وسیعی از پروژه‌ها، اعم از تولیدِ فراورده‌های سفارشی یا فراورده‌هایی که قابل دریافت و بارگذاری از طریق اینترنت می‌باشند، مفهومِ استقرار به گونه‌ی متفاوتی مطرح می‌باشد. در هر یک از این روش‌های اِستقرار، بر حسب مورد، دستاوردهای خاصی از دیسیپلین استقرار شکل می‌گیرد.

کلیدی‌ترین دستاورد، یک نسخه‌ی تحویلی[2] است که در قالب یک یا چند واحدِ استقرار[3] بسته‌بندی شده است. این واحدهای استقرار ممکن است شامل دستاوردهای زیر باشند:

- نرم‌افزار قابل اجرا، در تمامِ قالب‌های مطلوبِ آن
- دستاوردهای نصب و راه‌اندازی: اسکریپت، ابزارها، فایل‌ها، راهنمایی‌ها، و اطلاعات مربوط به لیسانس[4]
- یادداشت‌های مربوط به نسخه‌ی تحویلی[5] توصیف‌کننده‌ی مشخصه‌های نسخه‌ی تحویلی
- برخی مطالب و اطلاعات پشتیبانی مانند دستْنامه‌ی کاربران، دستْنامه‌های نگهداری و پشتیبانی
- مطالب آموزشی[6]

در حالتی که فراورده به صورت یک بسته‌ی آماده[7] باید تحویل شود، دستاوردهای دیگری نیز برای ایجاد یک فراورده‌ی کامل، مورد نیاز است، از جمله:

- لیست محتویات[8] (لیست کاملی از آیتم‌هایی که باید در محصول گنجانده شوند)
- طرحِ هنریِ فراورده[1] (یا اثر هنری مربوط به محصول که بخشی از بسته‌بندی بوده و بیانگر مارک[2] و هویّت فراورده می‌باشد)

---

1 - Deployment Artifacts
2 - Release
3 - Deployment Unit
4 - Licensing Information
5 - Release Notes
6 - Training Materils
7 - Shrink-Wrapped
8 - Bill of Materials

برخی دیگر از دستاوردهای دیسیپلینِ اِستقرار[۳] که ممکن است بر حسب ضرورت ایجاد شوند، ولی لزوماً به مشتری تحویل نمی‌شوند، عبارتند از: نتایجِ تست[۴] (از محل تولید و نیز نتایج حاصل از تست در محل مشتری[۵])، نتایج بازخورد[۶] (از تست‌های بِتا)، و چکیده‌ی ارزیابی تست‌ها[۷].

در شکل ۱۷-۲ ، نقش‌ها و دستاوردهای مهمِ دیسیپلین استقرار نشان داده شده است.

شکل ۱۷-۲

نقش‌ها و دستاوردهای دیسیپلینِ استقرار

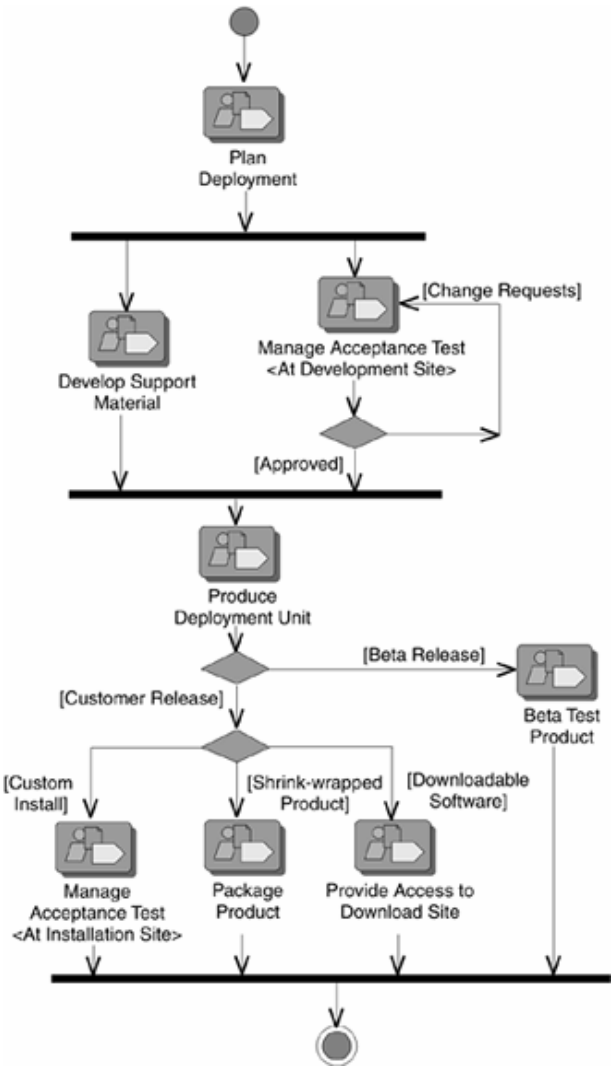
[1] - Product Artwork
[2] - Brand
[3] - Deployment
[4] - Test Results
[5] - On-site Testing
[6] - Feedback Results
[7] - Test Evaluation Summary

جریان‌کار[1]

جریانِ منطقی ارتباطِ میان مجموعه فعالیت‌های دیسیپلینِ استقرار در شکل ۱۷-۳ ، نشان داده شده است. این جریان‌کار که به وسیله‌ی یک دیاگرام فعالیت[2] (از دیاگرام‌های زبانِ مدل‌سازی استاندارد یعنی یو.ام.ال[3]) نشان داده شده است، علاوه بر دسته‌بندی مجموعه فعالیت‌ها، نوعی ترتیب منطقی را نیز میان آن‌ها تعریف می‌نماید.

شکل ۱۷-۳

جریان‌کار در دیسیپلینِ استقرار



[1] - Workflow
[2] - Activity Diagram
[3] - UML: Unified Modeling Language

چکیده‌ی فصل

این فصل به بررسی دیسیپلینِ استقرار و ملاحظات مرتبط با آن اختصاص داشت. مهم‌ترین مواردی که در این فصل مورد بررسی قرار گرفت، عبارتند از:

- دیسیپلین استقرار مرتبط است با تحویلِ همه‌ی دستاوردهای مورد نیاز کاربران نهایی یا مشتریان و نیز دستاوردهایی که در اختیار سازمان‌ها و بخش‌هایی مانند بخش پشتیبانی، بازاریابی، توزیع، و فروش قرار می‌گیرد.
- ماهیت این دیسیپلین وابستگی زیادی به نوعِ فراورده و نیز زمینه‌ی تجاری آن دارد و معمولاً در سازمان‌های مختلف، به شکل متفاوتی پیکربندی می‌شود.
- مجموعه‌ی فعالیت‌های مرتبط با تستِ بِتای نرم‌افزار و ملاحظات مرتبط با نصب و راه‌اندازی، از جمله مهم‌ترین فعالیت‌های این دیسیپلین می‌باشند.

پرسش‌هایی برای تحقیق و مطالعه‌ی بیشتر

۱. درباره‌ی ملاحظات مرتبط با بسته‌بندی فراورده در آر.یو.پی تحقیق نمایید.
۲. چگونگی تولیدِ محتوای آموزشی را در دیسیپلینِ اِستقرار بررسی نمایید.
۳. درباره‌ی تست‌های بِتا و شرایطِ محیطی و ابزارهای آن تحقیق نمایید.
۴. چگونگی اِستقرار فراورده را در محیط مشتری در فازهای مختلف، بررسی نمایید.
۵. مفهوم استقرار در دو رویکردِ آبشاری و تکرارشونده را باهم مقایسه نمایید.
۶. در رابطه با ابزارها و تکنولوژی‌های مختلف مرتبط با فراهم‌آوری ملزومات نصب و راه‌اندازی فراورده، تحقیق نمایید.

منابع و مراجع

[1]. Ivar Jacobson, Grady Booch, James Rumbaugh, (1999). *The Unified Software Development Process*, Reading, MA: Addison-Wesley

[2]. Philippe Kruchten, (2003). *The Rational Unified Process: An Introduction*, Reading, MA: Addison-Wesley.

[3]. Per Kroll, Philippe Kruchten, (2003). *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*, Reading, MA: Addison-Wesley.

[4]. Rational Software Corporation. (2003) "Rational Unified Process 2003.06," available at: http://www.rational.com/

[5]. Software Academy. (2006) "Unified Process Knowledge Base," Available at: http://www.unifiedProcess.info/

[6]. Scott W. Ambler, (2000). *The Unified Process Inception Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[7]. Scott W. Ambler, (2000). *The Unified Process Elaboration Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[8]. Scott W. Ambler, (2000). *The Unified Process Construction Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[9]. Scott W. Ambler and Larry L. Constantine, (2001). *The Unified Process Transition Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

فصل هجدهم

دیسیپلینِ محیط

مهم‌ترین موضوعات مورد بررسی در این فصل، عبارتند از:

- اهداف دیسیپلینِ محیط در آر.یو.پی
- فعالیت‌ها، دستاوردها، و نقش‌های کلیدی در این دیسیپلین
- نقشِ مهندسِ فرایند

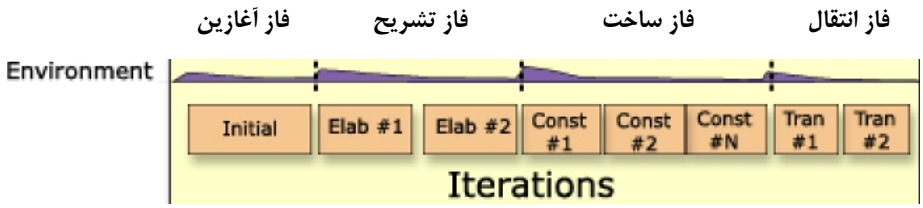
دیسیپلینِ محیط

۱۸

این فصل، به بررسی دیسیپلینِ محیط به عنوان یکی از سه دیسیپلینِ پشتیبان در فرایندِ آر.یو.پی اختصاص یافته است. پس از بررسی اهدافِ این دیسیپلین و مسائل مرتبط با آن، نقش‌ها، دستاوردها، و انواعِ فعالیت‌هایی را که در در قالبِ این دیسیپلین توصیف می‌شود، بررسی خواهیم نمود. شکل ۱۸-۱ ، بیانگر نمایی مقطعی از این دیسیپلین و حجمِ فعالیت‌های آن در طولِ فارهای چهارگانه‌ی چرخه‌ی تولید می‌باشد.

شکل ۱۸-۱

دیسیپلینِ محیط و حجمِ فعالیت‌های مرتبط با آن در طی فازهای مختلف فرایند تولید



هدف

هدفِ دیسیپلینِ محیط، عبارتست از پشتیبانی سازمانِ تولیدکننده‌ی فراورده، هم از جنبه‌ی فرایندِ مناسب برای تولید و هم از نظر ابزارهای لازم. این پشتیبانی‌ها عبارتند از:

- انتخاب[۱] و فراهم‌آوریِ[۲] ابزارهای مناسب و ضروری
- نصب و پیکربندی ابزارهای مورد نیاز
- پیکربندی[۳] فرایندِ تولید
- بهبودِ فرایند[۴]
- ارائه‌ی خدماتِ فنی[۵] برای پشتیبانی از فرایند: فراهم نمودنِ زیرساختِ فناوری‌اطلاعات[۶]، راهبری[۷] ابزارها و حساب‌های شخصی[۸] افراد مختلف تیم، تهیه‌ی پشتیبان[۹]، و موارد مشابه آن.

پیکربندی آر.یو.پی

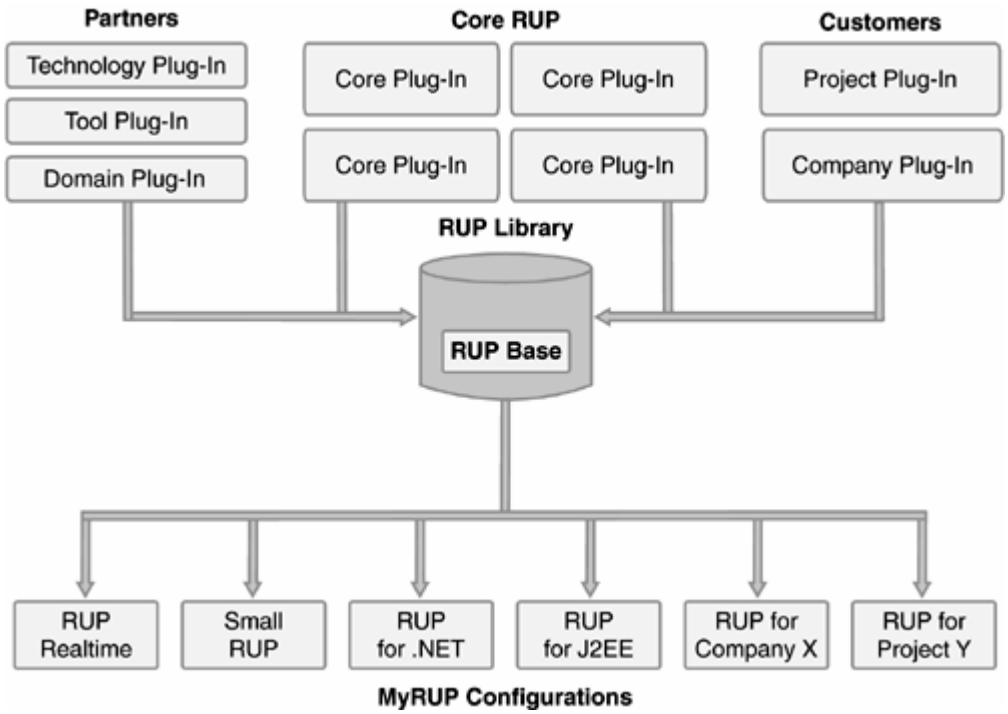
چارچوبِ فرآیندِ آر.یو.پی[۱۰] دربرگیرنده‌ی حجمِ زیادی از راهنمایی‌ها[۱۱]، دستاوردها[۱۲]، فعالیت‌ها، و نقش‌ها[۱۳] می‌باشد. از آن‌جایی که استفاده از تمام این دستاوردها در هیچ پروژه‌ای ممکن نیست، لازم است زیرمجموعه‌ای از آر.یو.پی را در هر پروژه به عنوانِ فرایند و قالبِ پروژه، انتخاب نماییم. این کار به وسیله‌ی انتخاب یا تولیدِ یک پیکربندی از فرایند آر.یو.پی[۱۴]، متناسب با نیازمندی‌ها و الزامات یک پروژه‌ی خاص، انجام می‌شود. بدین منظور، می‌توان از یکسری پیکربندی‌های موجود استفاده نموده و یا اینکه یک پیکربندی خاصی را به طور کلی از ابتدا تولید نمود.

---

[1] - Selection
[2] - Acquisition
[3] - Configuration
[4] - Process Improvement
[5] - Technical Services
[6] - Information Technology (IT) Infrastructure
[7] - Administration
[8] - Accounts
[9] - Backup
[10] - RUP Process Framework
[11] - Guidelines
[12] - Artifacts
[13] - Roles
[14] - RUP Process Configuration

شکل ۱۸-۲

پیکربندی آر.یو.پی برای پروژه‌های مختلف



نمونه‌سازی از آر.یو.پی

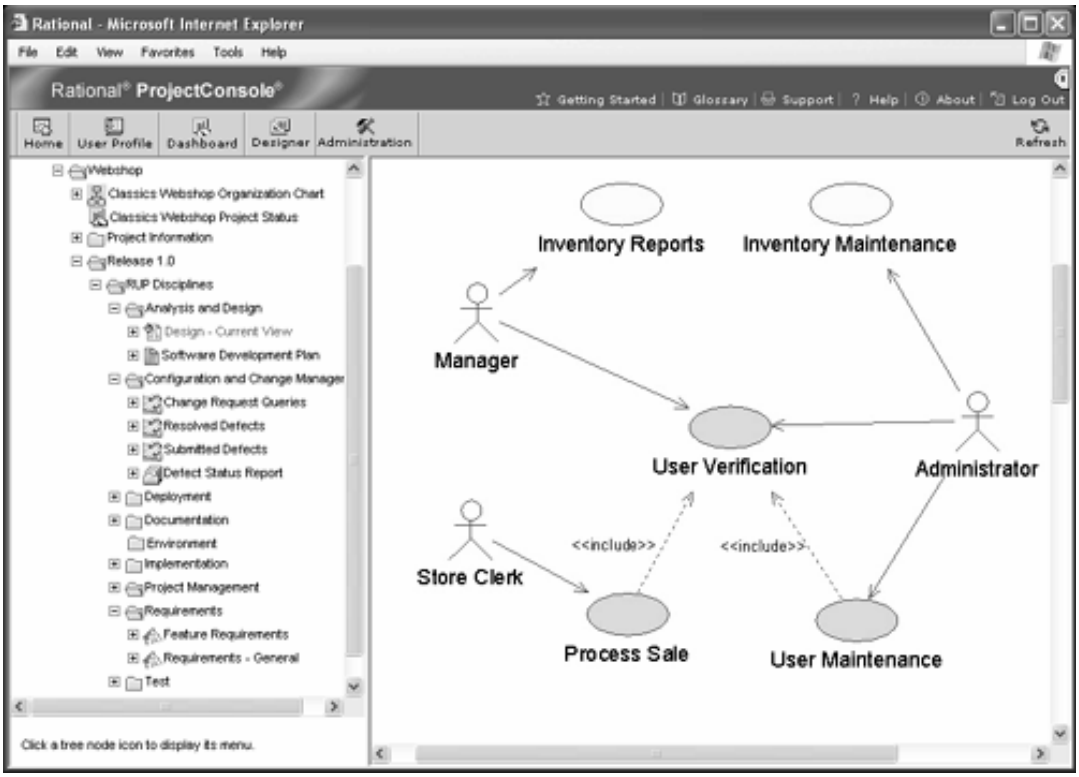
هر تیمی لازم است در رابطه با نوعِ پیکربندی آر.یو.پی و چگونگی استفاده از آن، دستاوردها و ابزارهای مورد نیاز، نقش‌های ضروری و مسائلی مانند آن، تصمیم‌گیری نماید. این ملاحظات در دستاوردی به نامِ قالبِ فرایند تولید[۱] که مهم‌ترین دستاوردِ دیسیپلینِ محیط می‌باشد، بیان می‌شود.

البته، توصیه می‌شود که برای هر پروژه، یک وب‌سایت[۲] در قالب پُرتال[۳] حاوی دستاوردهای مختلفِ مورد استفاده در پروژه، از جمله قالبِ فرایندِ تولید و پیکربندی خاص آر.یو.پی برای آن پروژه، ایجاد شود.

[1] - Development Case
[2] - Project Web Site
[3] - Portal

شکل ۱۸-۳

نمونه‌ای از وب‌سایتِ یک پروژه



سفارشی‌سازی[1] آر.یو.پی

سازمان می‌تواند مجموعه‌ی راهنمایی‌های خاصِ فرایندهای خود و نیز نسخه‌ی بازبینی شده و توسعه یافته‌ی عناصرِ آر.یو.پی را در قالبِ یک نسخه‌ی سفارشی‌شده از پیکربندی آر.یو.پی تهیه نماید. بهبودِ مستمر فرایند و افزودنِ تجارب هر یک از تکرارها، فازها، و چرخه‌های تولید به پیکربندی‌های موجودِ فرایند، از دیگر ملاحظات مورد توجه سازمان می‌باشد.

پیاده‌سازی[2] آر.یو.پی

یکی از مهم‌ترین نقش‌های فعال در دیسیپلینِ محیط، نقشِ مهندسِ فرایند[1] می‌باشد. وظیفه‌ی اصلی یک مهندس فرایند، پیاده‌سازی، استقرارِ آر.یو.پی در یک سازمان است. در بسیاری از موارد، پیاده‌سازی آر.یو.پی

[1] - Customization
[2] - Implementation

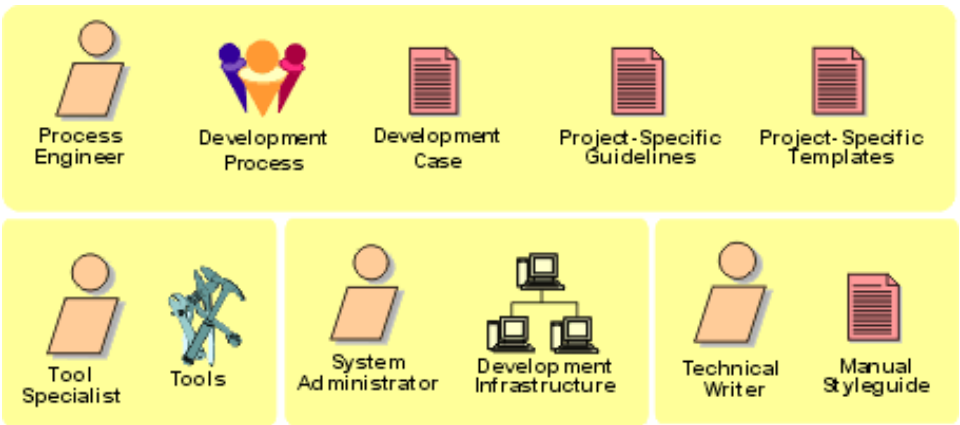
خود یک پروژه است که دارای گام‌های مختلفی مانند ارزیابی وضعیت فعلی سازمان، تعیین اهداف، و برپـایی دوره‌های آموزشی می‌باشد.

نقش‌ها و دستاوردها[۲]

در شکل ۱۸-۴ ، دستاوردها و نقش‌های مختلف مطرح در دیـسیپلین محـیط نـشان داده شـده اسـت. نقـش کلیدی در این دیسیپلین، مهندسِ فرایند[۳] می‌باشد. مهندسِ فرایند، مسئول پیکربندی اولیـه و بهبـودِ مـستمرِ فرایند در طول چرخه‌ی تولید و نیـز در سـازمان مـی‌باشـد. از دیگـر وظـایفِ مهنـدسِ فراینـد، فـراهم‌آوری راهنمایی‌ها و قالب‌های خاصِ پروژه‌ها می‌باشد. شکل ۱۸-۵ ، دستاوردها و فعالیت‌های یک مهندسِ فرایند را نشان می‌دهد.

شکل ۱۸-۴

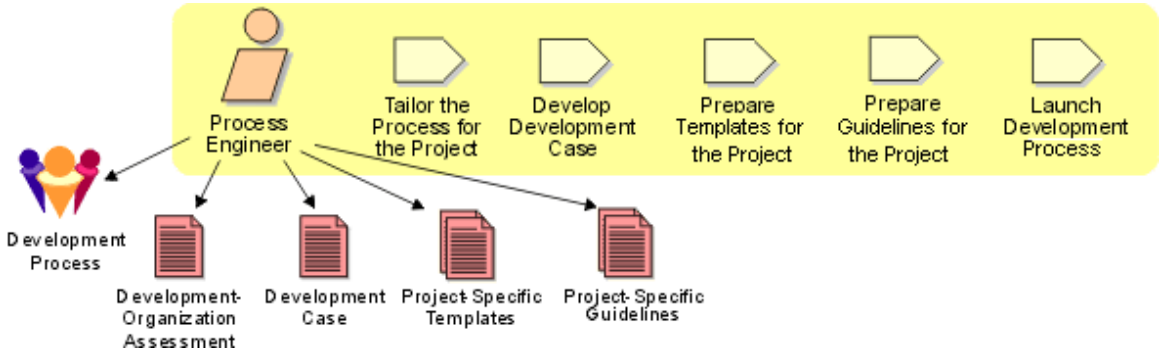
نقش‌ها و دستاوردها در دیسیپلینِ محیط



[1] - Process Engineer
[2] - Roles and Artifacts
[3] - Process Engineer

شکل ۱۸-۵

دستاوردها و فعالیت‌های مرتبط با نقش مهندسِ فرایند



اصلی‌ترین دستاوردِ دیسیپلینِ محیط، قالبِ فرایندِ تولید[1] می‌باشد. این دستاورد، بیانگر فرایندِ سفارشی‌شده[2] برای یک پروژه‌ی خاص می‌باشد. در واقع، قالبِ فرایندِ تولید، برای هر کدام از دیسیپلین‌های فرایند، چگونگی بکارگرفته‌شدن توسط یک پروژه‌ی خاص را بیان می‌نماید. دستاوردهای ضروری از هر یک از دیسیپلین‌ها و نیز چگونگی بکارگیری این دستاوردها، در قالبِ فرایندِ تولید، توصیف می‌گردد. البته، این دستاورد باید تا حدّ امکان به طور خلاصه بیان شده و جزئیات را به پیکربندی خاص مورد استفاده در پروژه، ارجاع دهد.

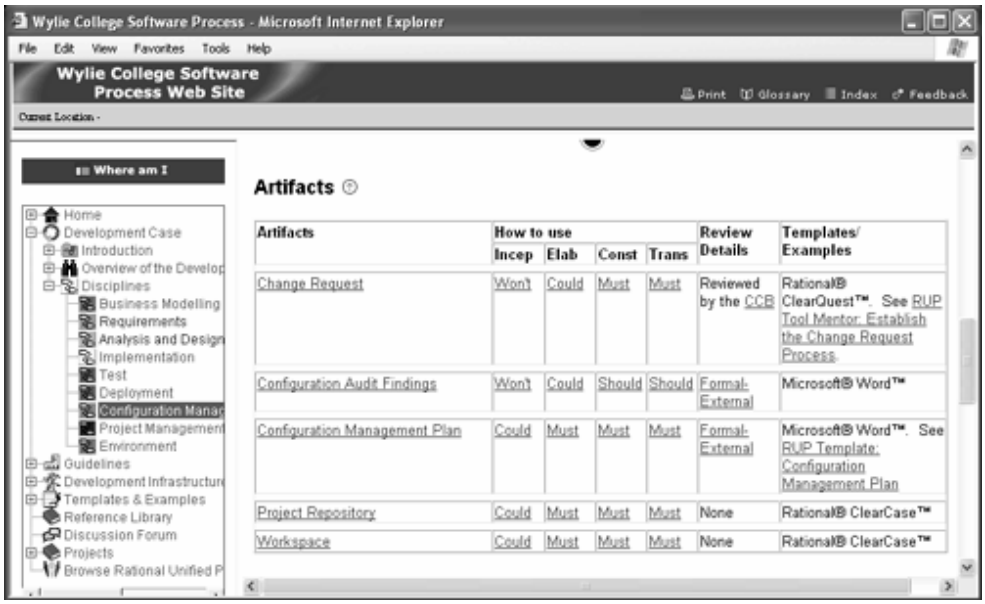
سایر نقش‌های فعّال در دیسیپلینِ محیط عبارتند از:

- متخصص ابزار[3]، مسئولیت انتخاب و تهیه‌ی ابزارها و پشتیبانی از محیط تولید را برعهده دارد. نصب، پیکربندی، و راه‌اندازی ابزارها، از جمله وظایف این نقش می‌باشد. ممکن است در یک پروژه، افراد مختلف در ارتباط با ابزارهای مختلف، این نقش را بر عهده داشته باشند.
- مسئولِ سیستم[4] که مسئول نگهداری محیط سخت‌افزاری و نرم‌افزاری توسعه بوده و فعالیت‌هایی نظیر مدیریت و راهبری حساب‌های شخصی و نیز تهیه‌ی پشتیبان را انجام می‌دهد.

[1] - Development Case
[2] - Tailored Process
[3] - Tool Specialist
[4] - System Administrator

شکل ۱۸-۶

توصیف دستاوردهای مختلفِ یک پروژه و میزان رسمی بودن آنها در دستاوردِ قالبِ فرایندِ تولید



شکل ۱۸-۷

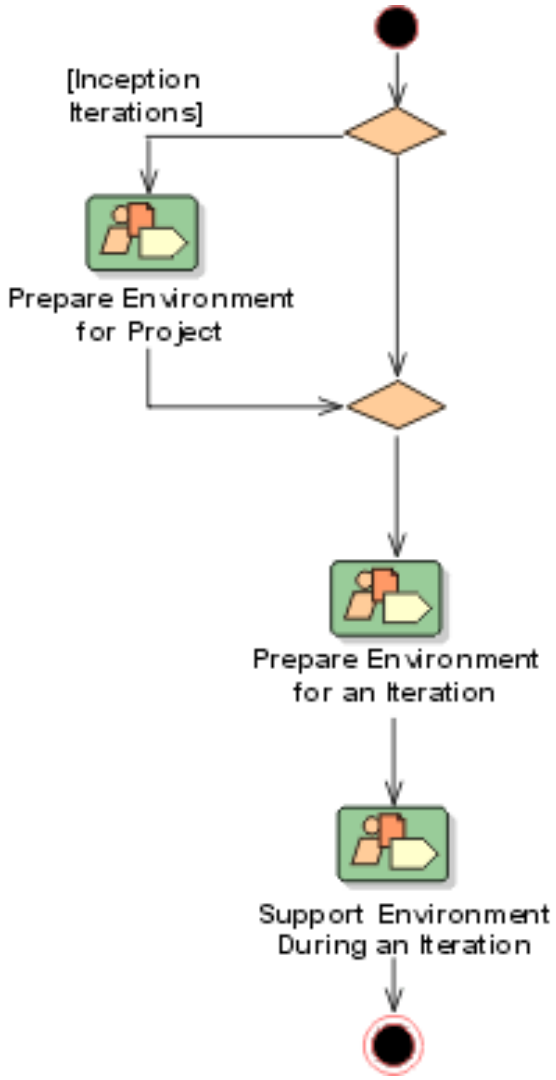
توصیفِ نقش‌ها در نمونه‌ای از یک قالبِ فرایندِ تولید

جریان‌کار[1]

در شکل ۱۸–۸ ، جریان‌کار دیسیپلینِ محیط در قالبِ مجموعه‌های جزئی جریان‌کار[2] نشان داده شده است. در ادامه، توضیح مختصری در رابطه با هر یک از مجموعه‌های جزئی جریان‌کار ارائه می‌گردد.

شکل ۱۸–۸

جریانِ‌کار دیسیپلینِ محیط



[1] - Workflow
[2] - Workflow Details

آماده سازی محیط برای پروژه[1]

اهداف عمده‌ی مجموعه‌ی فعالیت‌های مرتبط با آماده‌سازی محیط برای یک پروژه عبارتند از:

- ارزیابی[2] وضعیت کنونی سازمانِ تولیدکننده.
- ارزیابی وضعیت ابزارها و پشتیبانی آنها در شرایط کنونی سازمان.
- تولیدِ اولین نسخه‌ی پیش‌نویس[3] از قالبِ فرایندِ تولید[4].
- تهیه‌ی لیستی از ابزارهای کاندید برای بکارگیری در محیط تولید.
- تهیه‌ی لیستی از قالب‌ها و الگوهای[5] خاص پروژه برای دستاوردهای کلیدی.

آماده‌سازی محیط برای یک تکرار[6]

اهداف اصلی مجموعه فعالیت‌های مرتبط با آماده‌سازی محیط برای یک تکرار[7] عبارتند از:

- تکمیل و به‌روز رسانی قالبِ فرایندِ تولید[8]
- آماده‌سازی و در صورت ضرورت سفارشی نمودن ابزارها برای بکارگیری در یک تکرار.
- بازبینی و ممیزی[9] پیکربندی و نصب مناسب و صحیح ابزارها.
- تولید مجموعه‌ای از قالب‌ها و الگوهای خاص پروژه برای بکارگیری در یک تکرار.
- آموزش افراد برای یادگیری و بکارگیری ابزارها و قالبِ فرایند.

---

[1] - Prepare Environment for Project
[2] - Assess
[3] - Draft
[4] - Development Case
[5] - Templates
[6] - Prepare Environment for an Iteration
[7] - Iteration
[8] - Development Case
[9] - Verify

با تحلیل هر تکرار و کسب تجربه‌ی بیشتر، راهنمایی‌هـایی بـرای تکـرار بعـدی فـراهم مـی‌شـود. ایـن راهنمایی‌هایی عمدتاً مواردی مانند نکـاتی از مـدل‌سـازی سـازمان[۱]، مـدل‌سـازی مـواردِکـاربرد[۲]، طراحـی[۳]، برنامه‌نویسی[۴]، شیوه‌ی نوشتن مستندات کاربر[۵]، واسط کاربر[۶]، تست[۷]، و ابزارها[۸] را دربرمی‌گیرد.

پشتیبانی محیط برای یک تکرار[۹]

افراد تیم در طول یک تکرار به پشتیبانی نیاز دارند. مجموعه‌ی فعالیت‌های مرتبط با پشتیبانی محیط برای یک تکرار، راهنمایی‌ها و آموزش‌های مورد نیاز افراد و نیز پشتیبانی ابزارها و ملاحظات مربوط به فرایند را فراهم می‌نمایند.

چکیده‌ی فصل

به طور خلاصه، مهم‌ترین نکاتی که در این فصل به آن‌ها اشاره شد، عبارتند از:

- هدف دیسیپلینِ محیط عبارتست از ارائه‌ی خدمات و پشتیبانی کافی به سازمان تولیدکننده‌ی فراورده در مواردی مانند ابزارها[۱۰]، فرآیندها[۱۱]، و روش‌ها[۱۲].
- فراهم‌نمودن یک پیکربندی خاص از آر.یو.پی، متناسب با نیازها، اولویت‌هـا، انـدازه، نـوع، و شـرایط خاص هر پروژه.
- فرایند خاصی که در هر پروژه استفاده می‌شود، در سندی تحت عنوان قالبِ فراینـدِ تولیـد[۱] توصـیف می‌گردد. در واقع یک قالبِ فرایندِ تولید، چگونگی بکارگیری آر.یو.پی را به عنـوان فراینـد تولیـد در یک پروژه‌ی خاص، نشان می‌دهد.

---

[1] - Business Modeling
[2] - Use Case Modeling
[3] - Design
[4] - Programming
[5] - Manual Styleguide
[6] - User Interface
[7] - Test
[8] - Tools
[9] - Support Environment for an Iteration
[10] - Tools
[11] - Processes
[12] - Methods

- بسیاری از فعالیت‌ها و گام‌های آر.یو.پی را می‌توان به کمک ابزارهای مختلف، به صورتِ خودکار[۲] انجام داد. در عین حال که استفاده از ابزار و خودکارسازی[۳] با هدفِ حذف بسیاری از جنبه‌های خسته‌کننده، دستی، و خطاپذیر[۴] و در نتیجه بهبود کارایی فرایند تولید، انجام می‌شود، اما باید توجه داشت که امروزه، تحقّقِ بسیاری از ملاحظات مرتبط با تولیدِ نرم‌افزار، از جمله مواردی نظیر مدیریتِ پیکربندی و اجرای آزمون‌های مختلف، بدون بهره‌گیری از ابزارهای مناسب، امکان‌پذیر نمی‌باشد.

پرسش‌هایی برای تحقیق و مطالعه‌ی بیشتر

۱. درباره‌ی مهارت‌های لازم یک مهندسِ فرایند تحقیق نمایید.
۲. یکی از نمونه‌های قالبِ فرایندِ تولید ارائه شده در آر.یو.پی را بررسی نموده و با توجه به آن، ویژگی‌های پروژه‌ی مربوطه را بیان نمایید.
۳. درباره‌ی جایگاه دیسیپلینِ محیط در تکرارهای مختلف بحث نمایید.
۴. درباره‌ی دیسیپلینِ محیط در فازهای مختلفِ آر.یو.پی تحقیق نمایید.
۵. استراتژی‌های مختلف پیاده‌سازی آر.یو.پی در سازمان را بررسی و تحلیل نمایید.
۶. در رابطه با مفهومِ بهبودِ فرایند و ارتباط آن با دیسیپلینِ محیط تحقیق نمایید.

---

[1] - Development Case
[2] - Automate
[3] - Automation
[4] - Error-prone

منابع و مراجع

[1]. Ivar Jacobson, Grady Booch, James Rumbaugh, (1999). *The Unified Software Development Process*, Reading, MA: Addison-Wesley

[2]. Philippe Kruchten, (2003). *The Rational Unified Process: An Introduction*, Reading, MA: Addison-Wesley.

[3]. Per Kroll, Philippe Kruchten, (2003). *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*, Reading, MA: Addison-Wesley.

[4]. Rational Software Corporation. (2003) "Rational Unified Process 2003.06," available at: http://www.rational.com/

[5]. Software Academy. (2006) "Unified Process Knowledge Base," Available at: http://www.unifiedProcess.info/

[6]. Scott W. Ambler, (2000). *The Unified Process Inception Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[7]. Scott W. Ambler, (2000). *The Unified Process Elaboration Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[8]. Scott W. Ambler, (2000). *The Unified Process Construction Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[9]. Scott W. Ambler and Larry L. Constantine, (2001). *The Unified Process Transition Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[10]. Scott Ambler, (1998). *Process Patterns: Building Large-Scale Systems Using Object Technology*. New York: SIGS Books/Cambridge University Press.

[11]. Walker Royce, "CMM vs. CMMI: From Conventional to Modern Software Development," in The Rational Edge, February 2002.

فصل نوزدهم

دیسیپلینِ مدیریتِ پیکربندی و تغییرات

مهم‌ترین موضوعات مورد بررسی در این فصل، عبارتند از:

- اهداف دیسیپلینِ مدیریتِ پیکربندی و تغییرات در آر.یو.پی
- فعالیت‌ها، دستاوردها، و نقش‌های کلیدی
- مفهومِ مدیریتِ درخواستِ تغییر و ارتباط آن با مفاهیمِ سنجش و پیکربندی

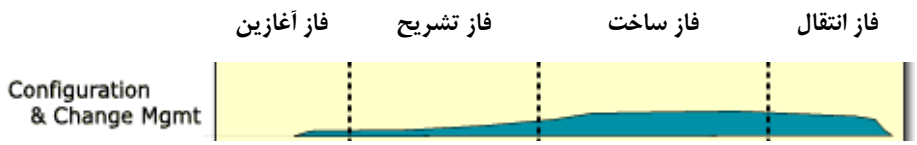
دیسیپلینِ مدیریتِ پیکربندی و تغییرات

۱۹

در این فصل، ابتدا مفاهیمِ مرتبط با مدیریتِ پیکربندی را تشریح نموده و در ادامه، نقش‌ها، فعالیت‌ها، و دستاوردهای دیسیپلینِ مدیریتِ پیکربندی و تغییرات را که یکی از دیسیپلین‌های پشتیبان در آر.یو.پی می‌باشد، توصیف خواهیم نمود. شکل ۱۹-۱، نمایی مقطعی از حجمِ فعالیت‌های مرتبط با دیسیپلینِ مدیریت پیکربندی و تغییرات[1] را در طولِ فازهای مختلف فرایندِ تولید[2] نشان می‌دهد.

شکل ۱۹-۱

نمایی مقطعی از حجم فعالیت‌های دیسیپلینِ مدیریتِ پیکربندی و تغییرات در چرخه‌ی تولید



[1] - Configuration & Change Management
[2] - Development Process

هدف

هدفِ دیسیپلینِ مدیریتِ پیکربندی و تغییرات، پیگیری و حفظ تمامیّتِ[1] دستاوردهای[2] در حال تکامل[3] در یک پروژه می‌باشد. در طی چرخه‌ی تولید، دستاوردهای با ارزش زیادی ایجاد می‌شود. توسعه‌ی این دستاوردها، مستلزم سرمایه‌گذاری و تلاشِ فراوانی است. بنابراین، این دستاوردها، دارایی‌های ارزشمندی محسوب می‌شوند که باید به خوبی نگهداری شده و امکان استفاده‌ی مجدد از آنها فراهم شود. این دستاوردها، دائماً در حال نِمو و تکامل می‌باشند و خصوصاً در رویکرد تکرارشونده، بارها و بارها به‌روز رسانی می‌شوند.

باوجودی‌که در رویکرد آر.یو.پی، مسئولیت هر دستاورد تنها برعهده‌ی یک نفر می‌باشد، اما مسلم است که برای نگهداری دستاوردها و سوابق آنها، همیشه نمی‌توان به ذهن و حافظه‌ی اشخاص متّکی بود. اعضای تیمِ پروژه باید قادر باشند به راحتی دستاوردهای مختلف و محلِ نگهداری آنها را شناسایی[4] نموده، نسخه‌ی مناسبی[5] از یک دستاورد را انتخاب کرده، وضعیت فعلی آن را از روی تاریخچه‌ی[6] مرتبط با آن، تعیین نمایند، و همچنین قادر به مشخص کردن مسئول فعلی آن دستاورد باشند.

بطور همزمان، افرادِ تیم باید قادر به پیگیری روندِ تکاملِ فراورده نیز باشند. واضح است که آنچه موجب تکامل یک فراورده می‌شود، تغییر است. بنابراین، با دریافتِ درخواست‌های تغییر از محل‌های مختلف و اِعمال اثر آنها به گونه‌ای که سازگاری عناصرِ مختلف حفظ شود، می‌توان روندِ تکاملِ فراورده را پیگیری نمود.

در واقع، توجه داشته باشید که برای پشتیبانی از دیسیپلینِ مدیریت پروژه[7]، اطلاعاتی درباره‌ی وضعیت هریک از دستاوردهای کلیدی فراهم شده و معیارهای لازم برای سنجشِ پیشرفت جمع‌آوری می‌شود.

---

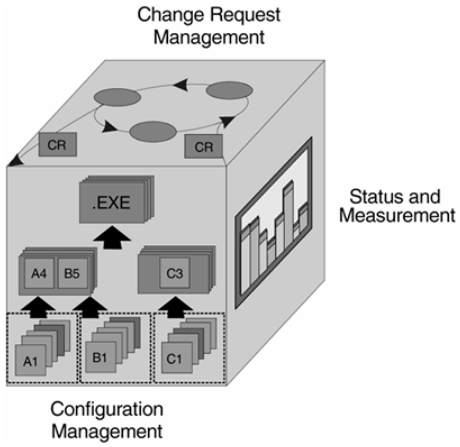
[1] - Maintain Integrity
[2] - Artifacts
[3] - Evolving
[4] - Identify
[5] - Appropriate Version
[6] - History
[7] - Project Management Discipline

مُکعبِ مدیریتِ پیکربندی و تغییرات[1]

مدیریتِ پیکربندی و تغییرات دربرگیرنده‌ی سه وظیفه‌مندی به‌هم وابسته[2] می‌باشد. این سه مقوله‌ی مهم را می‌توان در قالبِ یک مکعب، همانند آنچه شکل ۱۹-۲ نشان می‌دهد، بیان نمود.

شکل ۱۹-۲

مکعبِ مدیریتِ پیکربندی و تغییرات



این مکعب دارای سه وجه[3] است. هر یک از وجه‌های این مکعب، یک جنبه‌ی متفاوت از مسأله را مورد بررسی قرار می‌دهد. این سه وجه که بطور کامل به هم وابسته‌اند، عبارتند از:

- وجهِ مدیریتِ پیکربندی، که به طور عمده درباره‌ی ساختار فراورده می‌باشد.

مدیریتِ پیکربندی[4] با مسائل مختلفی در ارتباط می‌باشد، از جمله: شناسایی دستاوردها[5]، نسخه‌های مختلف[6]، وابستگی میانِ دستاوردها[7]، شناسایی پیکربندی‌های[8] مختلف، و نیز فراهم‌کردن یک محیطِ کاری امن[1] و مناسب برای افراد و تیم پروژه.

---

[1] - The CCM Cube
[2] - Interdependent Function
[3] - Face
[4] - Configuration Management
[5] - Artifact Identification
[6] - Versions
[7] - Dependencies among Artifacts
[8] - Configuration

- وجهِ مدیریتِ درخواست‌های تغییر، که عمدتاً در ارتباط با ساختار فرایند می‌باشد.

مدیریتِ درخواست‌های تغییر[۲] با دریافت و مدیریت تغییرات درخواست شده بـه وسـیله‌ی ذینفعـان[۳] داخـل و خارج از تیم پروژه، سروکار دارد. از دیگر مسائل مرتبط با مدیریت درخواست‌هـای تغییـر، مـی‌تـوان تحلیـل اثرات احتمالی تغییر و ردگیری نتایج حاصل از اِعمال تغییر را ذکر نمود.

- وجهِ مربوط به وضعیت[۴] و سنجش[۵] که مرتبط با کنترل ساختارِ کنترلِ پروژه[۶] می‌باشد.

وضعیت و سنجش با مسائلی مانند استخراج اطلاعات مناسب برای مدیریت پروژه از ابزارهای پشتیبانِ مدیریتِ پیکربندی و مدیریتِ درخواست‌های تغییر در ارتباط می‌باشد. برای یک ارزیابی موفق، داشتن اطلاعات زیر بسیار مفید می‌باشد:

- وضعیت، میزان پیشرفت، گرایش‌ها[۷]، و کیفیت محصول،
- آنچه که انجام شده و آنچه که باقی مانده است،
- برآمد هزینه‌ها[۸]،
- مواردی از مسأله که مستلزم توجه می‌باشد (ریسک‌ها).

در ادامه‌ی این فصل، هر یک از این اَبعاد و وجه‌های مکعبِ مدیریتِ پیکربندی و تغییرات را بـا تفـصیل بیشتری بررسی خواهیم کرد.

---

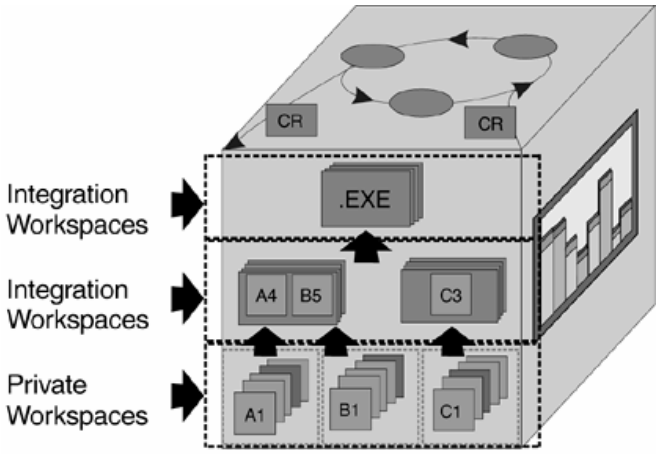
[1] - Secure Workspace
[2] - Change Request Management or CRM
[3] - Stakeholders
[4] - Status
[5] - Measurement
[6] - Project Control Structure
[7] - Trends
[8] - Expenditure

مدیریت پیکربندی[1]

ساختارِ فراورده[2]، مهم‌ترین مسأله‌ی مورد توجه در جنبه‌ی مدیریتِ پیکربندی از دیسیپلینِ مدیریتِ پیکربندی و تغییرات می‌باشد. نسخه‌های مختلفِ دستاوردهای مهمّ پروژه، باید تحتِ کنترل قرار گیرد. با تحوّل و تکامل دستاوردها، نسخه‌های متعددی از آن‌ها به وجود می‌آید و ضروری است مکانیزم مناسبی برای شناسایی یک دستاورد، نسخه‌های مختلف، و نیز تاریخچه‌ی تغییرات آن وجود داشته باشد.

شکل ۱۹-۳

ساختار محصول و محیطِ کار



بسیاری از دستاوردهای مختلفِ پروژه به هم وابسته‌اند. لذا بروزِ یک تغییر، سببِ پدید آمدن اثری موج‌مانند[3] می‌شود. هنگامی که یک دستاورد، دچار تغییر شده و تصحیح می‌گردد، دستاوردهای وابسته به آن نیز باید بازبینی شده و احتمالاً موردِ بازبینی و تصحیح قرار گرفته و یا حتی ممکن است لازم باشد که دوباره از ابتدا تولید شوند.

برای مثال یک کدِ نوشته شده به زبان C++ به یک سری فایل‌های سرآیند[4] وابسته است. این کد همچنین به توصیف کلاسی که پیاده‌سازی آن را برعهده دارد، نیز وابسته است.

---

[1] - Configuration Management
[2] - Product Structure
[3] - Ripple Effect
[4] - Header Files

آخرین نسخه‌ی یک دستاورد، اغلب بهترین نسخه‌ی آن می‌باشد. امّا در برخی از شرایط، موضوع به همین سادگی نیست و لازم است نسخه‌های مختلف از دستاوردها را نگهداری کنیم و به موقع، پایدارترین و بهترین نسخه را که لزوماً آخرین نسخه‌ی آن نیست، انتخاب نماییم. علاوه بر این، در بسیاری از موارد لازم است که چندین نفر به طور موازی روی یک دستاورد خاص کار کرده و یا شکل‌های مختلف یک دستاورد در فراورده‌های مختلفی استفاده شود.

دانش و اطلاعاتِ مرتبط با وابستگی میانِ دستاوردها، تبدیل[۱] و تغییراتی که در میان این وابستگی‌ها اتفاق می‌افتد و نیز ابزارهایی که بر این تبدیل‌ها و انتقال‌ها تاثیرگذار می‌باشند را می‌توان استخراج نموده و به وسیله‌ی آن، بطور خودکار این تبدیل‌ها و تغییرات را روی دستاوردهای به هم وابسته اِعمال نمود. برای مثال، با استفاده از مکانیزم تهیه‌ی Makefile می‌توان به طورِ خودکار، تغییرات و تبدیل‌های لازم را در کدهای تغییر کرده‌ی یک سیستم در حال تولید در زبان C++ ، اِعمال کرد.

یکی دیگر از مفاهیمِ مرتبط با مدیریت پیکربندی، مدیریت ایجادِ یک نسخه‌ی میانی[۲] می‌باشد. ایجادِ یک نسخه‌ی میانی، تنها در صورتی امکان پذیر است که نسخه‌های مختلف از مؤلفه‌های لازم در یک پیکربندی خاص به خوبی تست شده و مخزنی از این نسخه‌ها را در اختیار داشته باشیم. مدیریتِ ساختِ[۳] نسخه‌های میانی، مستلزم فراهم بودن محیطِ مناسب و ابزارهای لازم برای یکپارچه‌سازی می‌باشد.

یک محیطِ کارِ امن[۴] فضایی است که افراد و تیم‌های مختلف را قادر می‌سازد بدون ایجاد اِختلال، به صورت موازی روی مجموعه‌ی مشترکی از دستاوردها کار کنند.

توجه داشته باشید که ساختارِ کلّی فراورده[۵]، در قالب سازماندهی همه‌ی دستاوردها، در منظرِپیاده‌سازی[۶] از معماری بیان می‌شود.

---

[1] - Transformation
[2] - Build
[3] - Build Management
[4] - Secure Workspace
[5] - The Overall Product Structure
[6] - Implementation View

مدیریت درخواست‌های تغییر[1]

مدیریت درخواست‌های تغییر با ساختارِ فرایند[2] در ارتباط می‌باشد. این موضوع در قسمت فوقانی از مکعب شکل ۱۹-۴، نشان داده شده است. یک درخواستِ تغییر[3] عبارتست از پیشنهادی مستندسازی‌شده[4] برای تغییرِ یک یا چند دستاورد. درخواستِ تغییر، ممکن است به دلایل مختلفی و همچنین از منابعِ مختلفی ناشی شده باشد. به عنوان مثال برخی نمونه‌های درخواستِ تغییر، عبارتند از: رفع یک نقص[5]، بهبود کیفیت فراورده (مانند کارایی[6] و قابلیتِ استفاده[7])، افزودن یک نیازمندی[8]، رفع یک ناسازگاری[9].

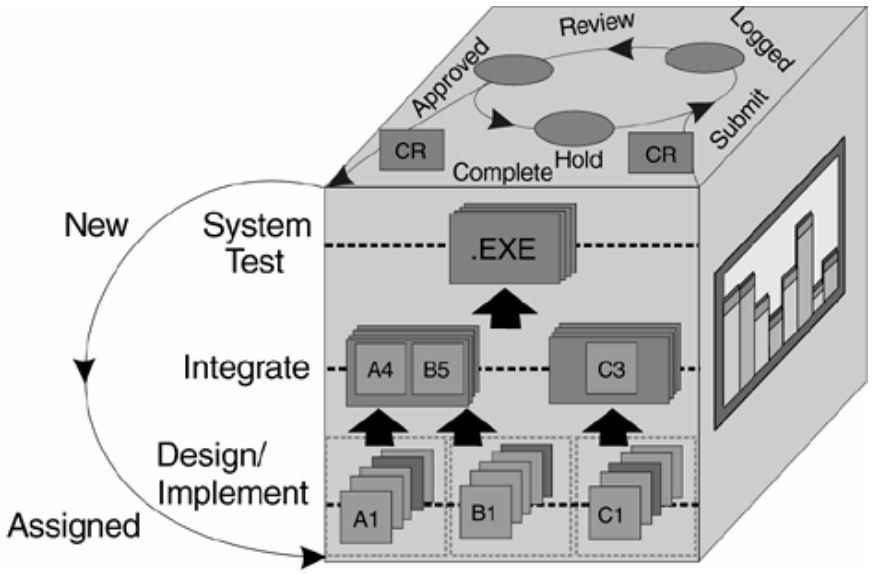
درخواست‌های تغییر دارای یک دوره‌ی عمر[10] می‌باشند که می‌توان آن را در قالبِ یک ماشین حالت[11] با حالت‌هایی مانند جدید، ثبت‌شده[12]، موافقت‌شده[13]، انتساب‌شده[14]، و کامل‌شده[15]، به تصویر کشید. با اِنتقالِ یک درخواستِ تغییر از یک حالت به حالت دیگر، اطلاعاتی درباره‌ی آن، مانند علّتِ تغییر، انگیزه‌ها، دستاوردهایی که تاثیر می‌پذیرند، و تأثیر آن بر طراحی، معماری، هزینه‌ها، و زمان‌بندی، اضافه می‌شود. بدیهی است که امکان پذیرش و تأثیر دادن تمام درخواست‌های تغییر، وجود نخواهد داشت. با توجه به بررسی نتایج ممکن و تأثیرات یک تغییر، نسبت به تأثیر دادن و یا عدم پذیرش آن، نوعی تصمیم‌گیری مدیریتی اتخاذ می‌گردد.

---

1 - Change Request Management
2 - Process Structure
3 - Change Request
4 - Documented Proposal
5 - Defect
6 - Performance
7 - Usability
8 - Requirement
9 - Inconsistency
10 - Life
11 - State Machine
12 - Logged
13 - Approved
14 - Assigned
15 - Complete

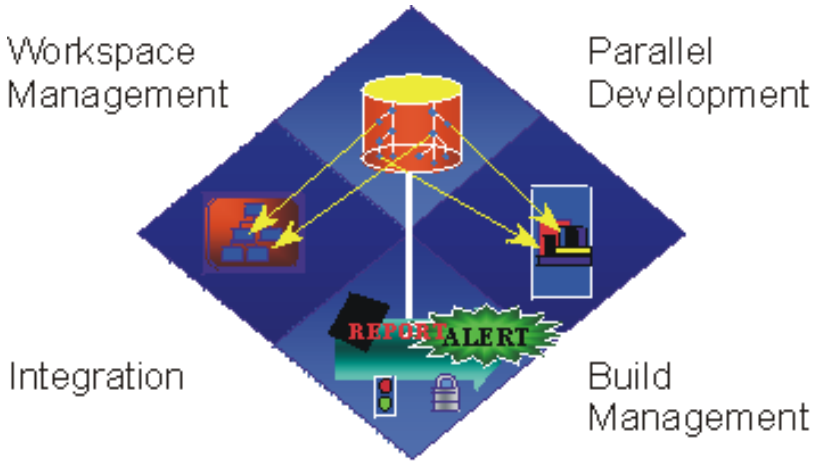
شکل ۱۹-۴

درخواست‌های تغییر



شکل ۱۹-۵

مدیریت تغییرات و عوامل مرتبط با آن

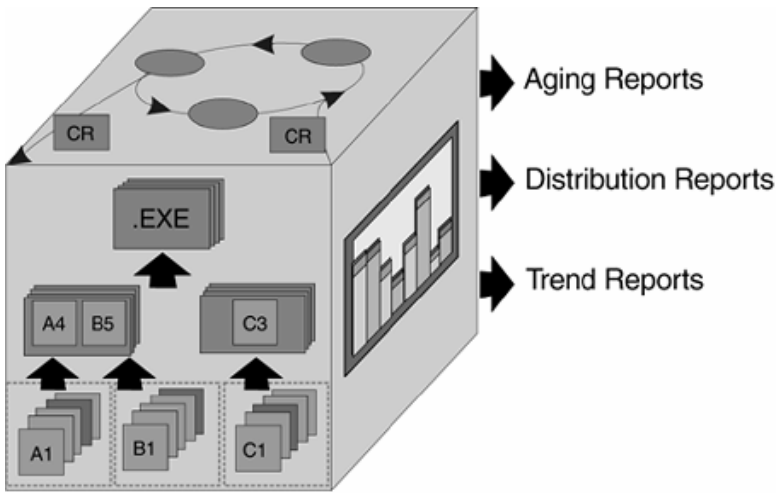


وضعیت و سنجش[1]

جنبه‌ی وضعیت و سنجش در ارتباط با ساختارِ کنترلِ پروژه[2] می‌باشد. این جنبه در سمتِ راست از مکعب شکل ۱۹-۶ ، نشان داده شده است. درخواست‌های تغییر، دارای یک وضعیت و ویژگی‌هایی نظیر دلایل اصلی[3]، ماهیّت (مانند یک نقص یا یک بهبود)، شدّتِ اثر[4]، اولویت[5]، و ناحیه‌ی اثر (مانند لایه‌ها یا زیرسیستم‌ها) می‌باشند. وضعیت‌های مختلف یک درخواستِ تغییر، پایه و مبنای مناسبی برای گزارش‌دهی نسبت به سنجشِ پیشرفت می‌باشد.

شکل ۱۹-۶

وضعیت‌ها و سنجش پیشرفت



بسیاری از درخواست‌های تغییر در یک بانک اطلاعاتی ثبت می‌شوند. خصوصاً در تکرارهای فازِ ساخت و اِنتقال، داشتن یک بانکِ اطلاعاتی از درخواست‌های تغییر و اثر آن‌ها ضروری است. با استفاده از این بانکِ اطلاعاتی، می‌توان اطلاعات مفیدی درباره‌ی پیشرفت پروژه استخراج نمود. از جمله:

- پیشرفت کلی پروژه نسبت به تغییرات
- تعداد تغییرات ثبت شده در هریک از وضعیت‌های مختلف پروژه (فازها و تکرارهای مختلف)

[1] - Status and Measurement
[2] - Project Control Structure
[3] - Root Causes
[4] - Severity
[5] - Priority

- میزانِ عمرِ یک درخواست تغییر، یعنی بازه‌ی زمانی ماندن درخواست‌های تغییر در یک وضعیت خاص

با کمک این بانک اطلاعاتی، گزارش‌های ذیل را نیز می‌توان درباره‌ی درخواست‌های تغییر در اختیار داشت:

- بر اساسِ شدّتِ اثر و یا اولویت،
- با تأثیری که بر یک لایه یا یک زیرسیستم خاص گذاشته‌اند،
- بر اساس تیم،
- بر اساس دلایل و ریشه‌های اصلی.

نقش‌ها[۱] و دستاوردها[۲]

نقش‌ها و دستاوردهای مرتبط با دیسیپلینِ مدیریتِ پیکربندی و تغییرات در شکل ۱۹-۷ ، نشان داده شده است. مهم‌ترین نقش‌های موجود در این دیسیپلین، عبارتند از:

- مدیر پیکربندی[۳] : مسئولیت‌های نصب و راه‌اندازی ساختارِ فراورده[۴] در سیستمِ مدیریتِ پیکربندی[۵]، تعریف و تخصیص فضای کاری[۶] به افراد تیم و نیز برقراری مکانیزم یکپارچه‌سازی و مجتمع‌سازی[۷] را برعهده دارد. در ضمن گزارش‌های مناسبی از وضعیت و سنجش پیشرفت در اختیار مدیر پروژه قرار می‌دهد.
- مدیرِ کنترلِ تغییرات[۸] : مسئولیتِ کنترل فرایند مدیریتِ تغییرات را برعهده دارد. در پروژه‌های بزرگ و پیچیده، این نقش عمدتاً توسط یک کمیته‌ی تخصصی، تحت عنوان بُردِ کنترلِ تغییرات[۹] یا بُردِ

---

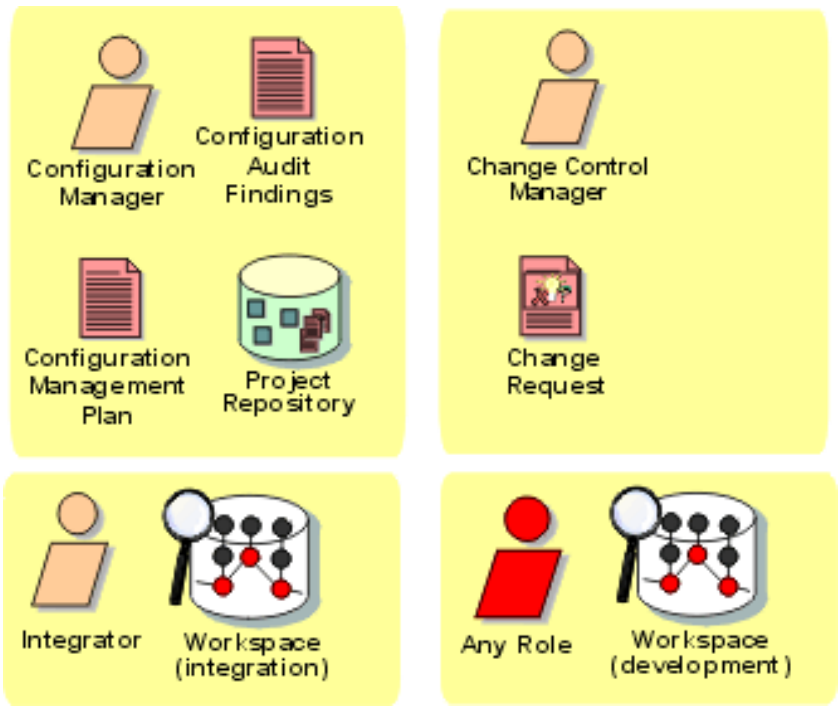
[1] - Roles
[2] - Artifacts
[3] - Configuration Manager
[4] - Product Structure
[5] - CM System
[6] - Workspace
[7] - Integration
[8] - Change Control Manager
[9] - Change Control Board (CCB)

کنترلِ پیکربندی، ایفا می‌شود. در بسیاری از موارد، این نقش را مدیر پروژه یا معمارِ سیستم بر عهده می‌گیرد.

بُرد یا کمیته‌ی کنترلِ تغییرات، متشکل از ذینفعان[1] مدیریتی و فنی پروژه می‌باشد. مدیر پروژه، معمارِ[2] سیستم، مدیرِ پیکربندی[3] و سایر ذینفعان کلیدی از جمله نماینده‌ی مشتری[4]، دراین کمیته عضویت دارند. مهم‌ترین وظایف این کمیته عبارتست از: ارزیابی اثر تغییرات[5]، تعیین اولویت‌ها، و تثبیتِ تغییرات.

شکل ۱۹-۷

نقش‌ها و دستاوردها در دیسیپلین مدیریتِ پیکربندی و تغییرات



برخی دیگر از نقش‌های موجود در فرایند، در فعالیت‌های این دیسیپلین مشارکت دارند. از جمله می‌توان به نقش‌های زیر اشاره نمود:

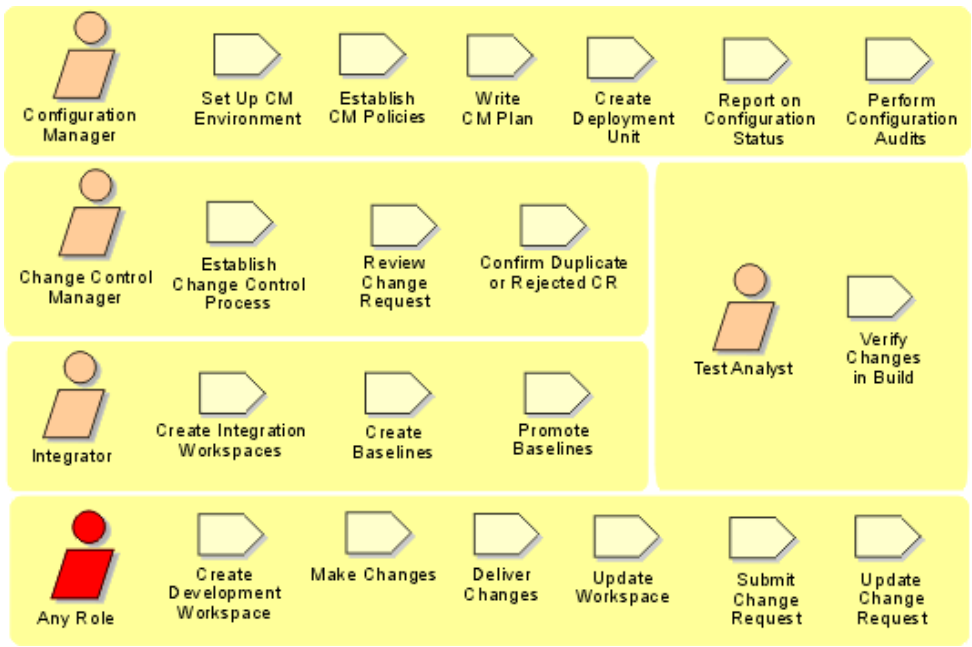
- معمارِ نرم‌افزار که ساختار فراورده را در قالب منظرِ پیاده‌سازی[1] از معماری، تدوین می‌نماید.

[1] - Stakeholder
[2] - Architect
[3] - Configuration Manager
[4] - Customer Representative
[5] - Assess the Impact of Change

- برنامه‌نویسان[۲] که به یک فضای کاریِ اختصاصی دسترسی خواهند داشت و نیز دستاوردهایی را که باید تغییرات‌شان را پیاده‌سازی نمایند، دریافت می‌کنند.
- هر نقشی که به نوعی درخواستِ ارسال یک تغییر را دارد.
- مسئولین یکپارچه‌سازی[۳] که تغییرات را در محیط یکپارچه‌سازی، اِعمال کرده و نسخه‌های میانی[۴] فراورده را می‌سازند.

شکل ۱۹-۸

فعالیت‌های دیسیپلینِ مدیریتِ پیکربندی و تغییرات



مهم‌ترین دستاوردهای[۵] دیسیپلینِ مدیریتِ پیکربندی و تغییرات، عبارتند از: طرحِ مدیریت پیکربندی[۶]، درخواست‌های تغییر[۷]، و یافته‌های بازبینی پیکربندی[۸].

---

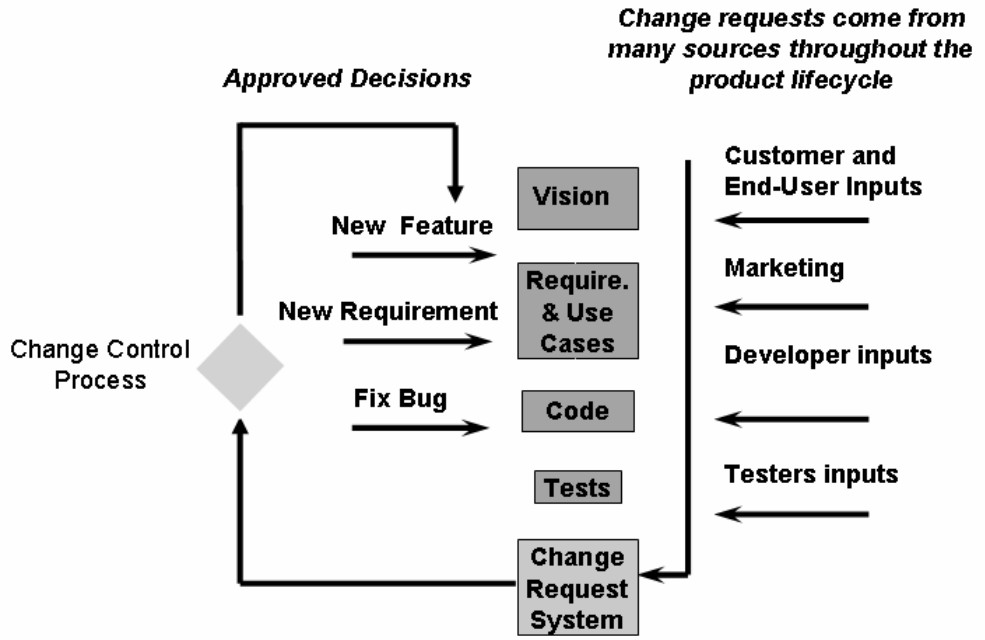
[1] - Implementation View
[2] - Implementers
[3] - Integrators
[4] - Builds
[5] - Artifact
[6] - Configuration Management Plan
[7] - Change Requests
[8] - Configuration Audit Findings

مدیریتِ یکپارچه‌ی تغییرات[1]

آر.یو.پی، مدیریتِ تغییرات را در فرایند تولید نرم‌افزار، با مکانیزمی تحتِ عنوانِ مدیریتِ یکپارچـه‌ی تغییـرات (UCM) معرفی می‌نماید. این مفهوم، با تعریفِ چگونگی مدیریتِ تغییرات در نیازمندی‌ها، مدل‌های طراحـی، مستندسازی، مؤلفه‌ها، مواردِ تست، و کدهای برنامه، تمامِ چرخه‌ی تولید را در بر می‌گیرد.

شکل ۱۹-۹

شِمایی از مفهومِ مدیریتِ یکپارچه‌ی تغییرات (UCM) در آر.یو.پی



ابزارهای پشتیبان

با افزایش اَبعادِ پروژه و تعداد ذینفعان[2]، پیچیدگی مدیریتِ تغییرات به صورت نمایی زیاد مـی‌شـود. بنـابراین، بدون بهره‌گیری از ابزارهای مناسب، مدیریتِ تغییرات در تیم‌های متوسط و بزرگ، کار بسیار دشوار، پر هزینه، و حتی در بسیاری از موارد، غیر ممکن می‌باشد. از میان ابزارهای موجود می‌توان ابزارهایی نظیـرِ Rational ClearCase و Rational ClearQuest را کـه ابزارهـای قدرتمنـدی در مـدیریت پیکربنـدی و تغییـرات می‌باشند، نام برد.

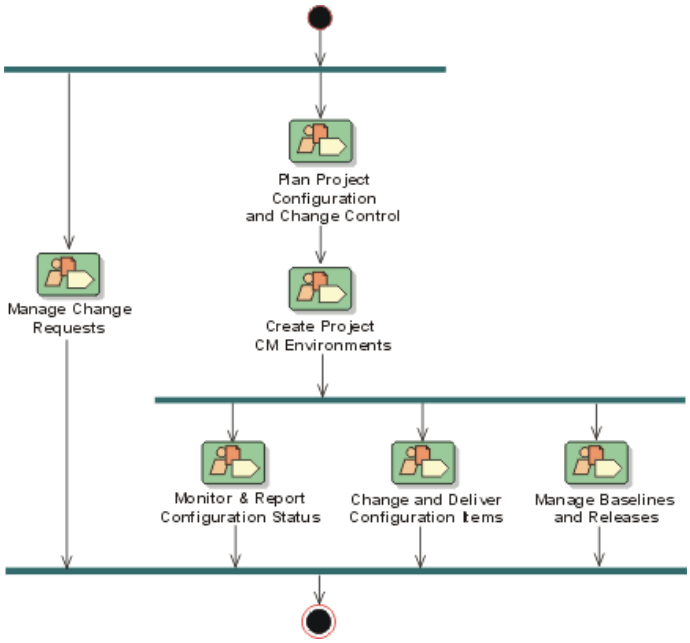
[1] - Unified Change Management
[2] - Stakeholders

جریان‌کار[1]

دیسیپلینِ مدیریتِ پیکربندی و تغییرات متناظر با دو جریان کارِ درهم‌پیچیده می‌باشد که یکی مرتبط با مدیریتِ پیکربندی و ساختارِ فراورده و دیگری مربوط به جنبه‌ی مدیریتِ درخواست‌های تغییر می‌باشد.

شکل ۱۹-۱۰

جریانِ‌کارِ مدیریتِ پیکربندی و تغییرات



چکیده‌ی فصل

در این فصل، یکی دیگر از دیسیپلین‌های پشتیبان را در فرایندِ آر.یو.پی، یعنی دیسیپلینِ مدیریتِ پیکربندی و تغییرات[2]، بررسی نمودیم. مهم‌ترین مباحثی که در این فصل عنوان شد، عبارتند از:

- هدفِ دیسیپلین مدیریتِ پیکربندی و تغییرات عبارتست از حفظ یکپارچگی و تمامیّتِ[3] دستاوردهای[4] پروژه که در پاسخ به درخواست‌های تغییر[5]، دچار تحول[1] می‌شوند.

---

[1] - Workflow
[2] - Configuration and Change Management
[3] - Integrity
[4] - Artifacts
[5] - Change Requests

- مدیریتِ پیکربندی با مسائلی مانند ساختارِ محصول[۲]، شناسایی و تطبیق عناصر[۳]، نسخه‌های مختلف[۴]، پیکربندی معتبر عناصر[۵]، و ملاحظات مرتبط با محیطِ کار[۶]، در ارتباط می‌باشد.
- مدیریتِ درخواست‌های تغییر[۷] عبارتست از فرایندِ کنترل شده‌ی بررسی تغییرات درخواست شده و اِعمال تغییرات روی دستاوردهای مختلف به شکلی که سازگاری[۸] حفظ گردد.
- معیارهای سنجش میزانِ پیشرفت و وضعیت پروژه را می‌توان بر اساس اطلاعات حاصل از مدیریتِ پیکربندی و تغییرات بدست آورد.
- خودکارسازی[۹] و بهره‌گیری از ابزارهای مناسب، نقش مهمی در کارایی فعالیت‌های مرتبط با دیسیپلین مدیریت پیکربندی و تغییرات دارد.

پرسش‌هایی برای تحقیق و مطالعه‌ی بیشتر

۱. در رابطه با مدیریتِ درخواست‌های تغییر و چگونگی تحقّق آن در آر.یو.پی، توضیح دهید.
۲. وظایف کمیته‌ی کنترل تغییرات[۱۰] چیست؟ درباره‌ی ساختار و چگونگی سازماندهی این کمیته توضیح دهید.
۳. درباره‌ی دیسیپلین مدیریتِ پیکربندی و تغییرات در فازهای مختلف آر.یو.پی تحقیق نمایید.
۴. ارتباط میان درخواستِ تغییر[۱۱]، پیکربندی[۱۲]، و سنجشِ پیشرفت[۱۳] را توصیف نمایید.
۵. درباره‌ی معیارهای[۱۴] سنجشِ پیشرفتِ پروژه تحقیق نمایید.

---

[1] - Evolve
[2] - Product Structure
[3] - Identification of Elements
[4] - Versions
[5] - Valid Configuration of Elements
[6] - Workspace
[7] - Change Request Management
[8] - Consistent
[9] - Automation
[10] - Change Control Board
[11] - Change
[12] - Configuration
[13] - Progress Measurement
[14] - Metrics

منابع و مراجع

[1]. Ivar Jacobson, Grady Booch, James Rumbaugh, (1999). *The Unified Software Development Process*, Reading, MA: Addison-Wesley

[2]. Philippe Kruchten, (2003). *The Rational Unified Process: An Introduction*, Reading, MA: Addison-Wesley.

[3]. Per Kroll, Philippe Kruchten, (2003). *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*, Reading, MA: Addison-Wesley.

[4]. Rational Software Corporation. (2003) "Rational Unified Process 2003.06," available at: http://www.rational.com/

[5]. Software Academy. (2006) "Unified Process Knowledge Base," Available at: http://www.unifiedProcess.info/

[6]. Scott W. Ambler, (2000). *The Unified Process Inception Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[7]. Scott W. Ambler, (2000). *The Unified Process Elaboration Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[8]. Scott W. Ambler, (2000). *The Unified Process Construction Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

[9]. Scott W. Ambler and Larry L. Constantine, (2001). *The Unified Process Transition Phase: Best Practices for Completing the Unified Process*, Lawrence, KS: R&D Books.

سخن پایانی

سخن پایانی

به رغمِ بیان آمارهایی تکان‌دهنده‌ی از عدم موفقیتِ پروژه‌ها و طرح‌های تولید فراورده‌های نرم‌افزاری، باید خاطر نشان نمود که در این کتاب هیچ‌گاه بیان نشد که مثلاً در آینده نیز تنها کمتر از ۳۰ درصد پروژه‌های نرم‌افزاری، موفق خواهند شد؛ این آمارها، واقعیت‌هایی بودند از گذشته، گذشته‌ای که اگر در آن کنکاش نماییم، اشتباهات مشترکی میان پروژه‌های ناموفق خواهیم یافت. امروز، دیگر هیچ لزومی ندارد که این اشتباهات گذشته را تکرار کنیم! برای قرار گرفتن در فضای عدم موفقیت، تنها کاری که باید انجام دهید تکرار اشتباهات دیگران است و برای جلوگیری از تکرار خطاها و قرار گرفتن در فضای موفقیت، تنها کارِ شما، کنار نهادن روش‌های اشتباه گذشته و یادگیری راهکارها و تجارب موفق می‌باشد.

موفقیت در عرصه‌ی امروزی تولید به طور کلی و تولید فراورده‌های نرم‌افزاری به طور خاص، مانند یک بازی شطرنج است. در این بازی، شانس، نقش چندانی ندارد! آنچه می‌تواند ضامن موفقیت در بازی شطرنج باشد، داشتن استراتژی مناسب است؛ پیروزی، بیش از هر چیز در دانش و توانمندی نهفته است، نه در بخت و اقبال! چنانچه به قوانین بازی آگاه بوده و از اشتباهات دوری جویید، احتمال موفقیت و پیروزی شما بسیار زیاد است. این یعنی، بهره‌گیری از راهکارهای موفق و دانشِ مهندسی.

در این کتاب سعی شد در حد توان و به طور مختصر، راهکارهای موفق و چگونگی تحقق آنها مورد بحث و بررسی قرار گیرد. در واقع، آر.یو.پی چیزی نیست جز گنجینه‌ی ارزشمندی از دانش و تجارب موفق در پروژه‌های نرم‌افزاری؛ گنجینه‌ی ارزشمند و قابل تحسینی که با وجود ماهیت و قالب تجاری آن، اکنون در اختیار ماست، پس باید آنرا به درستی بشناسیم و حداکثر بهره‌برداری را از آن داشته باشیم.

این کتاب به بررسی و شناختِ چیستیِ[1] آر.یو.پی اختصاص داشت. مفاهیم مهمی نظیر فازهای چرخه‌ی تولید و اهداف متناظر با آنها، نگاه خاص فرایندهای امروزی به مدیریت ریسک، نقش معماری، و توجه به مشتری و نیازهای واقعی‌اش، از مهم‌ترین نکات مطرح‌شده در این کتاب بود.

[1] - What

در اینجا لازم است بار دیگر فلسفه و روح آر.یو.پی را که برخی آن را روح و حال و هوای حاکم بر سازمان‌ها و تیم‌های موفق مهندسی و تولید نرم‌افزار نامیده‌اند، یادآور شویم:

- تولید نرم‌افزار کاری است تیمی، که در آن تنها یک تیم مشارکت دارد.
- فراورده‌های نرم‌افزاری را با استفاده از مؤلفه‌ها بنا نمایید.
- کیفیت را در بطنِ تمام فعالیت‌ها بگنجانید.
- همواره بر داشتن یک سیستم نرم‌افزاری با قابلیت اجرا شدن، تأکید داشته باشید.
- غلبه بر ریسک‌های عمده و حیاتی را در صدر فعالیت‌های خود قرار دهید؛ در غیر اینصورت، این ریسک‌ها بر شما غلبه خواهند نمود.
- تلاش نمایید که هر چه سریع‌تر، پایه‌ها و زیربنای سیستم را که معماری آن می‌باشد، تثبیت نموده و آن را مبنای بنا کردن بقیه‌ی سیستم، قرار دهید.
- تمام فعالیت‌های شما باید ارزشی برای مشتری در بر داشته باشد.
- تغییر، تنها مفهوم همیشگی و ثابت است. بنابراین، از همان ابتدای پروژه، استراتژی و تاکتیک مناسبی برای مدیریت آن، اتخاذ نمایید.

بی‌گمان، معرفی تمام جزئیات فرایندی مانند آر.یو.پی نه در یک کتاب که حتی در چندین کتاب نیز ممکن نمی‌باشد. در این کتاب سعی گردید به طور مختصر و در عین حال با نگرشی جامع، ابعاد و ویژگی‌های این به اصطلاح چارچوب فرایندهای تولید سیستم، تشریح گردد. امید است این کتاب بتواند ذهنیت مناسبی از آر.یو.پی که گنجینه‌ی دانش حاصل از بررسی و کنکاش پروژه‌های مختلف در طول بیش از بیست سال در سرتاسر دنیا می‌باشد، فراهم نماید. مسلماً این نوشته عاری از نقص و اشتباه نیست؛ لذا از خواننده‌ی محترم صمیمانه تقاضا می‌گردد با در میان نهادن نظرات و دیدگاه‌های سازنده‌ی خود، ما را در بهبود و تکمیل این کتاب، یاری نماید.

در اینجا و به عنوان آخرین سخن، نکته‌ی مهمی را در قالب یک داستان واقعی، یادآور می‌شویم. آقای پروفسور اینشتن در یکی از ترم‌هایی که در دانشگاه کرسی استادی داشتند، سؤالاتی را در پایان ترم در اختیار

دانشجویانشان قرار دادند. چند نفر از دانشجویان که سؤالات سال‌های قبل استاد را مرور نموده بودند، متوجه شدند که سؤالات تکراری است! در این حال، یکی از دانشجویان از پروفسور پرسید: «استاد، به نظر می‌رسد اشتباهی رخ داده باشد چرا که سؤالات تکراری هستند!» فکر می‌کنید که استاد در پاسخ این دانشجو چه گفتند؟ جالب است بدانید که پاسخ ایشان چنین بود: « **سؤالاتِ امروز ما همان‌هایی است که دیروز هم با آنها مواجه بودیم، امّا امروز، جواب‌ها متفاوت هستند!** ». آری، امروز بسیاری از سؤالات ما در دنیای نرم‌افزار مشابه بیست یا سی سالِ پیش هستند با این تفاوت که امروز باید پاسخ‌هایی متفاوتی برای آنها داشته باشیم. بنابراین، امروز باید به پاسخ‌های امروزی توجه کنیم و پاسخ‌های گذشته را کنار بگذاریم.

پیوست‌ها

واژه‌های اختصاری

فرهنگِ واژه‌های تخصصی

فرهنگِ واژه‌های اختصاری

API	Application Programming Interface
BOM	Bill of Materials
CASE	Computer Aided Software Engineering
CBD	Component-Based Development
CCM	Change and Configuration Management
CD	Compact Disk
CM	Configuration Management
CMM	Capability Maturity Model
CMMI	Capability Maturity Model Integrated
CMP	Core Message Platform
CMU	Carnegie Mellon University
COCOMO	COnstructive COst MOdel
COTS	Commercial Off-The-Shelf
CR	Change Request
CRM	Change Request Management
DC	Development Cycle
ERP	Enterprise Resource Planning
EUP	Enterprise Unified Process
EVM	Earned Value Management
GA	Goal Achieved
GEQ	Good Enough Quality
IBM	International Business Machines
IDC	International Data Corporation

IOC	Initial Operational
ISO	International Standards Organization
IT	Information Technology
J2EE	Java 2 Platform Enterprise Edition
LCA	Lifecycle Architecture
LCO	Lifecycle Objectives
MDA	Model-Driven Architecture
MDD	Model-Driven Development
MTBF	Mean Time between Failures
OMG	Object Management Group
PMBOK	Project Management Book of Knowledge
PMI	Project Management Institute
PR	Product Release
RFI	Request for Information
RFP	Request for Proposal
RFQ	Request for Quote
ROI	Return on Investment
RPW	Rational Process Workbench
RUP	Rational Unified Process
SAD	Software Architecture Document
SDP	Software Development Plan
SEBOK	Software Engineering Book of Knowledge
SLA	Service-Level Agreement
SOA	Service Oriented Architecture
SPEM	Software Process Engineering Metamodel

SPI	Software Process Improvement
TCO	Total Cost of Ownership
UCM	Unified Change Management
UI	User Interface
UML	Unified Modeling Language
UP	Unified Process
USDP	Unified Software Development Process
XP	eXtreme Programming

فرهنگِ واژه‌های تخصصی

در این واژه‌نامه، واژه‌های مرتبط با آر.یو.پی و نیز واژه‌هایی را که از منظر آر.یو.پی معنای خاصی دارند، معرفی می‌نماییم. مراجع اصلی که در تهیه‌ی این واژه‌نامه‌ی تخصصی استفاده شده‌اند، عبارتند از:

[1]. Rational Software Corporation. (2003) "Rational Unified Process 2003.06," available at: http://www.rational.com/

[2]. Per Kroll, Philippe Kruchten, (). *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*, Reading, MA: Addison-Wesley.

[3]. Philippe Kruchten, (2003). *The Rational Unified Process: An Introduction*, Reading, MA: Addison-Wesley.

[4]. Ivar Jacobson, Grady Booch, James Rumbaugh, (1999). *The Unified Software Development Process*, Reading, MA: Addison-Wesley

واژه‌نامه‌ی انگلیسی به فارسی

Activity **(فعالیت)**

واحدی است از کار که یک نقش خاص آن را انجام می‌دهد. هر فعالیت ممکن است به چند گام (Step) شکسته شود. فعالیت‌های مختلف در آر.یو.پی یا تولیدکننده‌ی دستاوردها هستند یا تصحیح کننده‌ی آنها.

Actor **(اَکتور یا کُنشگر)**

شخص یا چیزی خارج از سیستم (یا سازمان) که با سیستم (یا سازمان) در تعامل می‌باشد.

Architectural pattern (الگوی معماری)

توصیفی از یک راه حلِ نمونه و موفق برای حلِ مسائل و مشکلات جاری در طراحی که منعکس‌کننده‌ی تجارب اثبات شده‌ی طراحی معماری می‌باشد.

Architectural view **(منظرِ معماری)**

منظری از معماری سیستم از یک چشم‌اندازِ مشخص؛ تمرکز عمده بر ساختار، پیمانه‌ای بودن، مؤلفه‌های کلیدی، و جریان‌های کنترل اصلی.

Artifact **(دستاورد)**

قطعه‌ای از اطلاعات که به وسیله‌ی یک فرایند تولید، تصحیح، یا به روز رسانی می‌شود. این قطعه‌ی اطلاعاتی معمولاً یک مسئولیت را ایجاب نموده (نقشی که مسئولیت آن را بر عهده می‌گیرد) و گونه‌های مختلف آن باید کنترل گردد. یک دستاورد می‌تواند یک مدل، عنصری از یک مدل، یا یک سند باشد.

Baseline (مبنا، خطِ مبنا)

یک نسخه‌ی بازبینی‌شده و پذیرفته‌شده از دستاوردها است که مبنا و مرجعی مورد توافق را برای تکامل یا توسعه فراهم می‌نماید و تغییرِ آن تنها در یک رویه‌ی رسمی مانند کنترل تغییرات و پیکربندی، اِمکان‌پذیر می‌باشد.

Build (نسخه‌ی میانی)

یک گونه‌ی عملیاتی از سیستم یا قسمتی از آن است که در آن زیرمجموعه‌ای از قابلیت‌ها که باید در فراورده‌ی نهایی وجود داشته باشد، نشان داده می‌شود.

Change control board or CCB (کمیته‌ی کنترل تغییرات)

نقش کمیته‌ی کنترل تغییرات عبارتست از فراهم نمودن یک ساز و کارِ کنترلِ مرکزی به منظور اطمینان از اینکه تمام درخواست‌های تغییر به درستی مورد توجه قرار گرفته، تصویب، و هماهنگ شده‌اند.

Change management (مدیریت تغییرات)

فعالیتِ کنترل و ردگیری تغییرات اِعمالی به دستاوردها

Change request or CR (درخواستِ تغییر)

درخواستی برای تغییر یک دستاورد یا فرایند. چیزهایی که در یک درخواست تغییر، مستند می‌شوند عمدتاً شامل اطلاعاتی پیرامون منشأ و شدّت اثرِ مشکل، راه حل پیشنهادی، و هزینه‌ی آن می‌باشد.

Class (کلاس یا رَده)

توصیفی از مجموعه‌ای از اشیاء که دارای مسئولیت‌ها، روابط، عملکرد، و ویژگی‌های مشابهی می‌باشند.

Component (مؤلفه)

یک جزء با اهمیت، تقریباً مستقل، و قابل جایگزین از سیستم که یک وظیفه‌مندی و عملکردِ مشخص را در یک معماریِ به خوبی تعریف شده، انجام می‌دهد. یک مؤلفه در تطابق با مجوعه‌ای از واسطها (interfaces) بوده و تحقق فیزیکی آنها را فراهم می‌نماید.

Component-based development or CBD (توسعه یا تولید مبتنی بر مؤلفه)

ایجاد و استقرارِ سیستم‌های نرم‌افزاری با گردِ هم آوردنِ مؤلفه‌های نرم‌افزاری و نیز تولید و استحصالِ این مؤلفه‌ها.

Configuration (پیکربندی)

(۱) در معنای کلی: چیدمانی از یک سیستم یا شبکه که به وسیله‌ی طبیعت، تعداد، و ویژگی‌های اصلی واحدهای عملیاتی آن تعریف می‌گردد؛ این مفهوم هم درباره‌ی نرم‌افزار و هم درباره‌ی سخت‌افزار مطرح است. (۲) نیازمندی‌ها، طراحی، و پیاده‌سازی‌هایی که تعریف کننده‌ی یک گونه از سیستم یا یک مؤلفه از سیستم می‌باشد.

Configuration management or CM (مدیریتِ پیکربندی)

یک جریانِ کار پشتیبانی‌کننده از فرایند که هدف آن عبارتست از: شناسایی، تعریف، و مبنا قرار دادنِ اَقلام و دستاوردها؛ کنترلِ تصحیحات و نسخه‌های منتشره‌ی این دستاوردها؛ گزارش‌دهی و ثبتِ وضعیتِ اَقلام و تصحیحات درخواست‌شده؛ اطمینان از کامل بودن، سازگاری، و صحّتِ اقلام؛ کنترلِ ذخیره‌سازی، اداره‌نمودن، و تحویل این اَقلام. (بر اساس تعریفِ سازمانِ ایزو)

Construction (ساخت)

فاز سوم از فرایند یکپارچه‌ی رَشنال (آر.یو.پی)، که در آن نرم‌افزار از یک معماری قابل اجرای مبنا قرار داده شده (که نتیجه‌ی فاز دوم فرایند می‌باشد) به نقطه‌ای می‌رسد که آماده‌ی انتقال به محیط کاربران نهایی می‌گردد. بخش عمده‌ای از مرحله‌ی فرآوری محصول در این فاز انجام می‌شود.

Cycle (چرخه)

یک گذارِ کامل از چهار فازِ آر.یو.پی (یعنی فازهای آغازین، تشریح، ساخت، و انتقال) که به یک نسخه‌ی منتشره از فرآورده منجر می‌گردد.

Defect (نقص، عیب)

یک نابهنجاری در فرآورده؛ مانند نقص‌ها و کمبودهایی که عمدتاً در مراحل اولیه پیدا می‌شوند.

Deliverable (قابلِ تحویل)

یک دستاورد خروجی از فرایند که دارای ارزش یا اطلاعاتی برای یک مشتری است.

Design model (مدلِ طراحی)

مدلی از اشیاء که بیانگر تحقق مواردِ کاربرد می‌باشد؛ این مدل به عنوان مدلی تجریدی از مدلِ پیاده‌سازی و مجموعه‌ی کدهای سیستم مطرح می‌باشد.

Development case (قالبِ فرایندِ تولید)

مشخصات فرایندی که توسط سازمانِ تولیدکننده به عنوان فرایندِ تولید، استفاده می‌شود. این قالب به وسیله‌ی سفارشی‌سازی و پیکربندی آر.یو.پی بدست آمده و در تطابق با نیازهای پروژه می‌باشد.

Discipline (**دیسیپلین**)

گروه‌بندی منطقی نقش‌ها، فعالیت‌ها، دستاوردها و دیگر عناصر راهنمای فرایند در توصیف یک فرایند.

Domain (**دامنه**)

مجموعه‌ی دانش یا فعالیت‌هایی که مشخصه‌ی خانواده‌ای از سیستم‌های به هم مرتبط می‌باشد. مثلاً، دامنه‌ی سیستم‌های یکپارچه‌ی سازمانی.

Elaboration (**تشریح**)

دومین فاز از فرایند تولید که در طی آن، چشم‌انداز فراورده‌ی نهایی، نیازمندی‌ها، و معماری آن تعریف و تثبیت می‌گردد.

Environment (**محیط**)

یکی از دیسیپلین‌های پشتیبان در فرایندِ تولید که هدف آن تعریف و مدیریت محیطِ تولید می‌باشد؛ تشریح فرایند، پیکربندی محیط، و ابزارهای تولید.

Evolution (**تکامل**)

حیاتِ نرم‌افزار پس از چرخه‌ی تولید آن؛ هر چرخه‌ی بعدی که طی آن فراورده تکامل می‌یابد.

Extended help (**راهنمای گسترش‌یافته**)

یکی از ویژگی‌ها و اِمکانات آر.یو.پی که از طریقِ آن، ابزارهای تولید برحسب موضوعات مختلف با مجموعه‌ی محتویات آر.یو.پی مرتبط می‌شوند.

Framework (چارچوب)

یک ریزْمعماری[۱۸۹۷] که یک الگویِ ناکامل را برای سیستم‌های کاربردی در یک دامنه‌ی خاص، فراهم می‌نماید. چارچوب را می‌توان مانند یک میز پر از غذاهای مختلف تصور نمود.

Implementation (پیاده‌سازی)

دیسیپلینی از فرایند تولید که هدف آن پیاده‌سازی و تستِ واحدِ[۱۸۹۸] هر یک از کلاس‌ها می‌باشد.

Implementation model (مدلِ پیاده‌سازی)

مدلی که بیانگر مجموعه‌ای از زیرسیستم‌های پیاده‌سازی و مؤلفه‌های[۱۸۹۹] موجود در آنها می‌باشد.

Implementation view (منظرِ پیاده‌سازی)

یکی از منظرهای معماری[۱۹۰۰] که بیانگر سازماندهی عناصرِ استاتیکِ نرم‌افزار (کد، داده، و دستاوردهای مرتبط با آنها) در قالب بسته‌بندی[۱۹۰۱]، لایه‌گذاری[۱۹۰۲]، و مدیریت پیکربندی[۱۹۰۳] (مالکیت، استراتژی ارائه‌ی نسخه‌های تحویلی و مانندِ آن) می‌باشد. در آر.یو.پی این منظر، منظری است از مدلِ پیاده‌سازی.

Inception (آغازین)

اولین فاز از فرایند تولید که در طی آن اولین حرکت‌ها برای تولید فراورده‌ی نرم‌افزاری شروع شده و به نقطه‌ای می‌رسد که ریسک‌های عمده‌ی سازمانی (به صرفه‌بودن یا نبودن، ماهیت و ابعادِ پروژه و مانند آن) رفع می‌گردد.

1897 - Micro-architecture
1898 - Unit Test
1899 - Component
1900 - Architectural View
1901 - Packaging
1902 - Layering
1903 - Configuration Management

Increment (**خُرده، مقدارِ دِلتا**)

تفاوتِ کوچک (دِلتا) میان دو نسخه‌ی ارائه شده در طی دو تکرارِ متوالی.

Integration (**یکپارچه‌سازی، تلفیق**)

یکی از فعالیت‌های مهم در تولید نرم‌افزار که به وسیله‌ی آن، مؤلفه‌های نرم‌افزاری در قالب یک موجودیتِ بزرگترِ قابل اجرا، ترکیب می‌شوند.

Iteration (**تکرار**)

یک توالی مشخص از فعالیت‌ها که دارای برنامه و معیارهای ارزیابی[۱۹۰۴] مشخصی بوده و به یک نسخه‌ی قابلِ تحویل[۱۹۰۵] (در داخل سازمان یا به خارج از سازمان) منجر می‌گردد.

Layer (**لایه**)

یکی از راه‌های گروه‌بندی[۱۹۰۶] بسته‌های یک مدل در یک سطحِ یکسان از نظرِ تجرید[۱۹۰۷]

Logical view (**منظر منطقی**)

یکی از منظرهای معماری که در آن کلاس‌های اصلیِ طراحی سیستم توصیف می‌گردد. از جمله کلاس‌های مرتبط با کسب و کار و نیز کلاس‌هایی که مکانیزم‌های ساختاری و رفتاری کلیدی (مانایی[۱۹۰۸]، ارتباطات[۱۹۰۹]، تحملِ خطا[۱۹۱۰]، و واسطِ کاربر). در آر.یو.پی، این منظرِ، منظری از مدلِ طراحی می‌باشد.

1904 - Evaluation Criteria
1905 - Release
1906 - Grouping
1907 - Abstration
1908 - Persistency
1909 - Communications
1910 - Fault Tolerance

Management (مدیریت)

یکی از دیسیپلین‌های پشتیبان در فرایند تولید که هدفِ آن برنامه‌ریزی و مدیریت پروژه می‌باشد.

Measure (اندازه، مقیاس)

یک ویژگیِ عددی[1911] از یک موجودیت[1912]

Method (مِتُد، روش)

(۱) یک روش سیستماتیک و منظم برای انجامِ چیزی؛ برنامه‌های تشریحی و به طور منطقی منظم شده یا رویه‌هایی که به منظور انجام یک وظیفه و دستیابی به یک هدف، دنبال می‌شود. (۲) در یو.اِم.اِل: پیاده‌سازی یک کارکرد؛ الگوریتم یا رویه‌ای که بر نتیجه‌ی یک کارکرد تأثیر می‌گذارد.

Metrics (معیار)

از نسخه‌ی ۲۰۰۰ آر.یو.پی به بعد، این مفهوم منسوخ شده و جای خود را به اصطلاحِ measurement داده است.

Milestone (گامِ اصلی)

نقطه‌ای که در آن یک تکرار یا یک فاز رسماً به انتها می‌رسد؛ در تطابق با نقطه‌ی ارائه‌ی یک نسخه

Model (مدل)

یک تجرید از سیستم که دارای معنا و مفهوم مشخص و معینی است. در آر.یو.پی، مدل عبارتست از: توصیفی کاملِ سیستم از یک جنبه یا دیدگاه خاص که با حذف جزئیات غیر ضروری، درک سیستم را از آن جنبه آسان می‌سازد.

[1911] - Numeric Attribute
[1912] - Entity

MyRUP

مرورگر آر.یو.پی که با کمک آن می‌توانید فرایند را مشاهده کرده، در آن به جستجو بپردازید، به طور گرافیکی فرایند را پیمایش نمایید، و منظرهای شخصی از یک پیکربندی فرایند، برای خود ایجاد کنید.

Node (گره)

یک شیء فیزیکی در زمان اجرا که بیانگر یک منبعِ رایانش[۱۹۱۳] و عمدتاً دارای حداقل یک حافظه و ظرفیت پردازشی. مؤلفه‌های نرم‌افزاری روی این گره‌ها مستقر می‌شوند.

Phase (فاز)

بازه‌ی زمانی میان دو گامِ اصلی در فرایند که در طی آن به یکسری اهدافِ مشخص دست یافته، دستاوردهایی کامل شده، و تصمیماتی برای رفتن و یا نرفتن به فاز بعد اتخاذ می‌گردد.

Prototype (پیش‌الگو)

Quality (کیفیت)

Release (نسخه‌ی تحویلی)

Report (گزارش)

Requirement (نیازمندی)

توصیفی از یک وضع[۱۹۱۴] یا قابلیتِ سیستم؛ یا مستقیماً از نیازهای کاربر مشتق می‌شود یا اینکه در قرارداد، استاندارد، توصیف‌ها، یا اسناد رسمی دیگری ذکر شده است.

Requirements discipline (دیسیپلین نیازمندی‌ها)

[1913] - Computational Resource
[1914] - Condition

Risk **(ریسک، مخاطره)**

یک نگرانی یا ملاحظه‌ی در حال وقوع یا مانع شونده که دارای یک احتمال قابل توجه در تأثیرگذاری بر موفقیت در دستیابی به اهداف می‌باشد. به بیان دیگر، ریسک عبارت است از هر نگرانی یا شرایطی که مانع موفقیت گردد.

Role **(نقش)**

Scenario **(سِناریو)**

یک نمونه‌ی توصیفی (عینی) از یک موردِ کاربرد یا زیر مجموعه‌ای از یک موردِ کاربرد.

Sequence diagram **(دیاگرامِ توالی)**

دیاگرامی از زبانِ مدل‌سازی استاندارد یو.اِم.اِل که در آن تعاملِ میان اشیاء بر اساسِ ترتیب زمانی مدل می‌شود.

Software architecture **(معماری نرم‌افزار)**

Stakeholder **(ذینفع)**

Step **(گام)**

Test **(تست، آزمون)**

Tool mentor **(راهنمایِ ابزار)**

توصیفی که فراهم کننده‌ی راهنمایی‌ها و توصیه‌های عملی در رابطه با چگونگی انجامِ یک فعالیتِ خاصِ فرایند به وسیله‌ی یک ابزارِ نرم‌افزاری خاص می‌باشد.

Transition **(انتقال)**

آخرین فاز از فازهای چهارگانه‌ی چرخه‌ی تولید در آر.یو.پی که منجر به نسخه‌ی تحویلی نهایی از فراورده می‌گردد.

Unified Modeling Language **(زبان مدل‌سازی یکپارچه)**

زبان مدل‌سازی استاندارد در دنیای مهندسی نرم‌افزار

Use-Case **(موردِ کاربرد)**

Use-case model **(مدل مواردِ کاربرد)**

مدلی که بیانگرِ چیستی سیستم و محیط آن می‌باشد.

Use-case realization **(تحققِ مواردِ کاربرد)**

توصیفِ چگونگیِ تحققِ یک موردِ کاربرد در قالبِ همکاری میان اشیاء در مدلِ طراحی

Use-case view **(منظرِ مواردِ کاربرد)**

منظری از معماری که بیانگرِ مواردِ کاربردِ کلیدی سیستم می‌باشد.

Version **(نسخه، گونه)**

View **(منظر)**

توصیفی ساده‌شده از یک مدل که از یک جنبه یا نقطه‌نظرِ خاص دیده می‌شود و در آن اطلاعاتِ غیرِ مرتب با آن جنبه، حذف می‌گردد.

Vision (چشم‌انداز)

در آر.یو.پی، چشم‌انداز عبارتست از دیدگاه و منظرِ کاربر یا مشتری از محصولی که باید تولید شود و به صورت خلاصه و در سطح نیازهای کلیدی ذینفعان و ویژگی‌های عمده‌ی سیستم، توصیف می‌گردد.

Worker (انجام‌دهنده‌ی کار)

این مفهوم از نسخه‌ی سال ۲۰۰۰ آر.یو.پی منسوخ شده و اصطلاحِ نقش (Role) به جای آن بکار می‌رود.

Workflow (جریان کار)

جریانِ کار، عبارتست از توالی فعالیت‌هایی که در یک سازمان انجام می‌شود و در نتیجه‌ی آن، ارزشِ قابلِ مشاهده‌ای برای یک اَکتور یا سازمان به دست می‌آید. در آر.یو.پی، مفهوم جریان‌های اصلیِ کار که ظرفی برای توصیف فرایند بود، جای خود را به مفهوم دیسیپلین داده است.

وب‌سایتِ دانشِ فرایند تولید

www.unifiedProcess.info

آکادمی نرم‌افزار

آدرس: تهران، خ آزادی، بلوار شهید اکبری،

خ شهید قاسمی، کوی تیموری، پلاک ۳

تلفن: ۶۶۰۰۸۶۰۴-۰۲۱

Software Academy

Sharif University of Technology

Web: *www.software-academy.com*

Email: info@software-academy.com