بسم الله الرحمن الرحیم

Patterns and AntiPatterns in Software Engineering

الگوها و پادالگوها

در مهندسی نرم‌افزار

نگارش: علی جلال

نسخه ۱.۰

مشاوران نرم‌افزاری اعوان

بهمن ۱۳۸۸

فهرست مطالب

بخش ۱ مقدمه

در این مستند قصد بر آن است تا الگوها و پادالگوها در مهندسی نرم‌افزار شرح داده شود. پیش از هر چیز نیاز داریم بدانیم الگو و پادالگو چیست؟

الگو

الگو راه‌حلی است تکرارشونده برای یک مسئله در حوزه‌ای خاص، که در عمل موفق بوده است. الگو یک سه-تایی متشکل از «مسئله – راه‌حل – حوزه» است. الگوها برای استفاده مجدد در مهندسی نرم‌افزار، از سیستم‌های موجود کشف می‌شوند.

پادالگو

پادالگو راه‌حلی است تکرارشونده برای یک مسئله در حوزه‌ای خاص، که در عمل موفق نبوده است. پادالگوها در مهندسی نرم‌افزار برای تکرار نکردن اشتباهات گذشتگان کشف می‌شوند. اگر یک الگو در حوزه مورد نظر اعمال نشود ممکن است به پادالگو تبدیل شود.

هر الگو برای یک حوزه[۱] خاص کاربرد دارد. منظور از حوزه دو مورد زیر است:

۱- یک سیستم و یا مجموعه سیستم‌ها، مثلاً سیستم‌های داده‌محور مبتنی بر وب، یا سیستم‌های خبره

۲- یک فاز از فرایند ایجاد نرم‌افزار مانند تحلیل، طراحی، برنامه‌نویسی، طراحی معماری و ... . به عنوان مثال الگو برای طراحی وجود دارد و الگو برای معماری نیز وجود دارد و نمی‌توان الگوهای طراحی را برای معماری به کار برد چون حوزه آنها طراحی است و ممکن است در معماری به پادالگو تبدیل شوند.

رخ دادن پادالگوها یا ناشی از عدم آگاهی است و یا در اثر به کار بردن یک الگوی مناسب در محیطی اشتباه.

۱-۱ تاریخچه الگوها

- در سال ۱۹۷۹ مفهوم الگو اولین بار توسط کریستوفر الکساندر در حوزه ساختمان و بناهای شهری (یعنی رشته عمران) ارائه شد.
- در سال ۱۹۸۷ برای اولین بار در مهندسی نرم‌افزار، توسط بِک[۲] و کانینگهام طراحی الگوهای کوچک زبان Smalltalk انجام شد. این الگوها الگوهای سطح زبان برنامه‌سازی[۳] بودند.
- در سال ۱۹۹۰ گروه چهار نفره[۴] ملقب به GoF کار بر روی الگوهای طراحی را شروع کردند.
- در سال ۱۹۹۲ Coad[۵] الگوهای شی‌گرای خود که مربوط به تحلیل و طراحی شی‌گرا بودند ارائه کرد.

---

[۱] Context

[۲] Kent Beck خالق متدولوژی XP و روش تشخیص کلاس CRC

[۳] Idiom، امروزه هر زبان برنامه‌نویسی رایج مثل جاوا و C#، الگوی برنامه‌نویسی خاص خود را دارد.

[۴] Gang of Four که متشکل از Gamma، Helm، Johnson و Vlissides بود.

- در سال ۱۹۹٤ اولین کنفرانس مختص الگوها به نام PLoP[۶] برگزار شد و همچنان تا به امروز در چندین شاخه و انواع الگوها در حال برگزاری است.
- در سال ۱۹۹۵ کتاب «الگوهای طراحی» توسط GoF ارائه شد. این کتاب از پرفروش‌ترین کتاب‌های مهندسی نرم‌افزار است. ضمناً فروش این کتاب از مجموع فروش همه کتاب‌های الگو در مهندسی نرم‌افزار بیشتر است.
- در سال ۲۰۰۰ الگوهای معماری نرم‌افزار توسط Buschmann و همکاران ارائه شد.
- و پس از سال ۲۰۰۰ کشف الگوها با شدت بیشتری دنبال شد... .

پس از کتاب پرفروش الگوهای طراحی GoF، الگوها در حوزه‌های مختلف از جمله تحلیل، طراحی، پیاده‌سازی (الگوهای خاص زبان برنامه‌نویسی)، معماری، فرایند و مهندسی مجدد ایجاد شدند. همراه با الگوها، پادالگوها نیز کشف شدند تا خطاهای رایج در مهندسی نرم‌افزار را گوشزد کنند.

در ادامه در هر بخش، الگوها و پادالگوهای هر حوزه را بررسی می‌کنیم.

۱-۲ چرا الگوها و پادالگوها را مطالعه کنیم؟

در زمان‌های دور[۷] که اولین نرم‌افزارها نوشته می‌شد افراد همه چیز را از صفر[۸] می‌ساختند، چون تجارب نرم‌افزاری وجود نداشت و مجبور بودند ایده‌های جدید برای ایجاد نرم‌افزار تولید کنند. پس از چند سال از قوت گرفتن شرکت‌های نرم‌افزاری در اواخر دهه ۱۹۸۰ کم‌کم مهندسین نرم‌افزار به این فکر افتادند که به جای «اختراع دوباره چرخ[۹]»، از تجارب قبلی خود استفاده کنند.

چرا الگو و پادالگو

همانگونه که در برنامه‌نویسی کتابخانه‌هایی از کلاس‌ها، ماژول‌ها و مؤلفه‌ها ایجاد شدند تا چرخ دوباره اختراع نشود، الگوها نیز از دل تجارب قبلی بیرون کشیده شدند تا همانند «کتابخانه‌ای از تجارب مفید» در دسترس همگان باشند و افراد آنها را نخواهند دوباره آنها را کشف کنند. پادالگوها نیز مانند «کتابخانه‌ای از تجارب غیرمفید» ثبت می‌شوند تا همه بدانند که نباید این تجارب را دوباره تکرار کنند.

انسان تا زمانیکه الگو یاد نگیرد، مبتنی بر قاعده[۱۰] عمل می‌کند. منظور از قاعده، «قوانین ریزدانه‌ای» هستند که فقط به همان مسئله محدود می‌شوند و سایر مسائل را نمی‌پوشانند. در این حالت برای هر مسئله جدید باید قاعده جدید تولید شود. اما الگوها «قوانین درشت‌دانه-ای» هستند که به یک خانواده از مسائل قابل انطباق هستند، یعنی مسائلی که در حوزه مورد نظر رخ می‌دهند. الگوها در حقیقت منطق

---

[۵] Coad به همراه Yourdon این کار را انجام دادند، این دو فرد متدولوژی Coad-Yourdon را نیز ایجاد کرده‌اند.

[۶] Pattern Languages of Programming

[۷] در حدود هفتصد میلیون ثانیه قبل که اولین الگوها در مهندسی نرم‌افزار کشف شدند.

[۸] From Scratch

[۹] Reinvent the Weel که یک پادالگو است.

[۱۰] Rule-Based

درشت‌دانه و عمومی‌ای هستند که برای هر مسئله خاص از آنها نمونه‌سازی[۱۱] می‌شود و نمونه ساخته شده استفاده می‌شود (همانگونه که اشیا از یک کلاس نمونه‌سازی می‌شوند).

امروزه در اکثر دانشگاه‌های مطرح دنیا، الگوهای طراحی را در درس برنامه‌نویسی پیشرفته می‌گویند تا ذهنیت الگو را در برنامه‌نویسی از همان ترم‌های ابتدایی دانشگاه جا بیندازند. در بسیاری از دانشگاه‌های دنیا انواع الگوها در حوزه مهندسی نرم‌افزار تدریس می‌شوند تا دید الگوشناسی به افراد داده شود. پس از اینکه افراد این دید را پیدا کنند در برخورد با چند مسئله خاص در یک حوزه، سعی می‌کنند آن‌ها را عمومی کنند[۱۲] و به یک الگو تبدیل کنند.

در بخش‌های بعدی، انواع الگوها و پادالگوهای فرایند، تحلیل، طراحی، اصلاح[۱۳] کد، معماری و ... شرح داده می‌شوند. در بخش دوم به الگوها و پادالگوهای «فرایند» ایجاد نرم‌افزار می‌پردازیم. توجه داشته باشید که این مستند در حال تکمیل است.

«این مستند یک مستند در حال رشد است و به مرور بخش‌های جدید به آن اضافه خواهد شد»

[۱۱] Instantiation

[۱۲] Generalizing

[۱۳] Refactoring

بخش ۲ الگوها و پادالگوهای فرایند

الگوهای فرایند[۱۴]، تکه‌های قابل استفاده مجدد[۱۵] هستند که از فرایندهای مختلف ایجاد نرم‌افزار بیرون کشیده شده‌اند و برای حل یک مشکل در فرایند متدولوژی، راه‌حل خوبی ارائه می‌دهند.

۲–۱ الگوهای فرایند

در زیر الگوهای به معتبرترین الگوهای فرایند می‌پردازیم.

۲–۱–۱ الگوهای Coplien

اولین بار مفهوم الگوی فرایند توسط Coplien در سال ۱۹۹٤ مطرح شد. Coplien الگوی فرایندی را به عنوان «الگوهای فعالیت در سازمان و پروژه‌های آن» تعریف کرد. این الگوها، الگوهای مدیریتی– سازمانی در سطح ریزدانه هستند و دو مشکل اساسی داشتند: ۱– ریزدانه بودند و از کنار هم قرار دادن آنها فرایند مناسب ایجاد نرم‌افزار ساخته نمی‌شد، ۲– این الگوها بیشتر جنبه مدیریتی– سازمانی دارند و کمتر جنبه متدولوژی و فرایندهای نرم‌افزاری دارند. به همین دلیل به این الگوها نمی‌پردازیم.

۲–۱–۲ الگوهای Ambler

تنها کتاب «الگوهای شی‌گرای فرایند» توسط Ambler نوشته شده است. Ambler الگوی فرایند را به این صورت تعریف کرد: «الگویی که یک دیدگاه و یا یک سری عملیات در عمل ثابت شده، برای ایجاد نرم‌افزار پیشنهاد می‌کند». الگوهای فرایند به سه دسته تقسیم می‌شوند که به ترتیب سطح انتزاع[۱۶] عبارتند از:

۱– **کار**[۱۷]: که جزییات مراحل اجرای یک کار از فرایند را توصیف می‌کند و ریزدانه‌ترین الگوی فرایند است.

۲– **مرحله**[۱۸]: که مراحل لازم برای انجام یک مرحله از فرایند را به تصویر می‌کشد. هر مرحله معمولاً از یک سری کار تشکیل شده است.

۳– **فاز**[۱۹]: که تعامل دو یا چند مرحله به منظور انجام یک فاز را به تصویر می‌کشد و درشت‌دانه‌ترین الگوی فرایند است.

---

[۱۴] Process Patterns

[۱۵] Reusable

[۱۶] Abstraction Level

[۱۷] Task

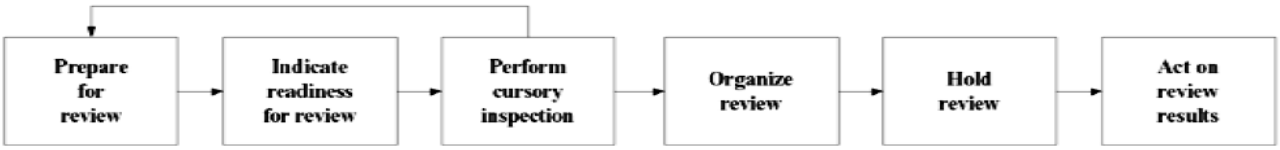
[۱۸] Stage

[۱۹] Phase

«فازها» در طول فرایند به صورت پشت سر هم و متوالی انجام می‌شوند ولی «مراحل» در هر فاز می‌توانند به صورت تکراری انجام شوند. Ambler برای هر یک از ۳ دسته بالا، الگوی‌های زیادی پیشنهاد کرده است که برای هر الگو مراحل دقیق و یک سری راهنمایی برای تجمیع الگوها و ایجاد یک فرایند جامع تولید نرم‌افزار ایجاد کرده است. در زیر یک مثال از هر الگوی فرایند را مشاهده می‌کنیم:

نمونه‌ای از الگوی فرایند کار: بازبینی فنی[۲۰]

در شکل زیر مراحل «کار بازبینی فنی» را مشاهده می‌کنید:



شکل ۱: کار بازبینی فنی (Ambler 1998)

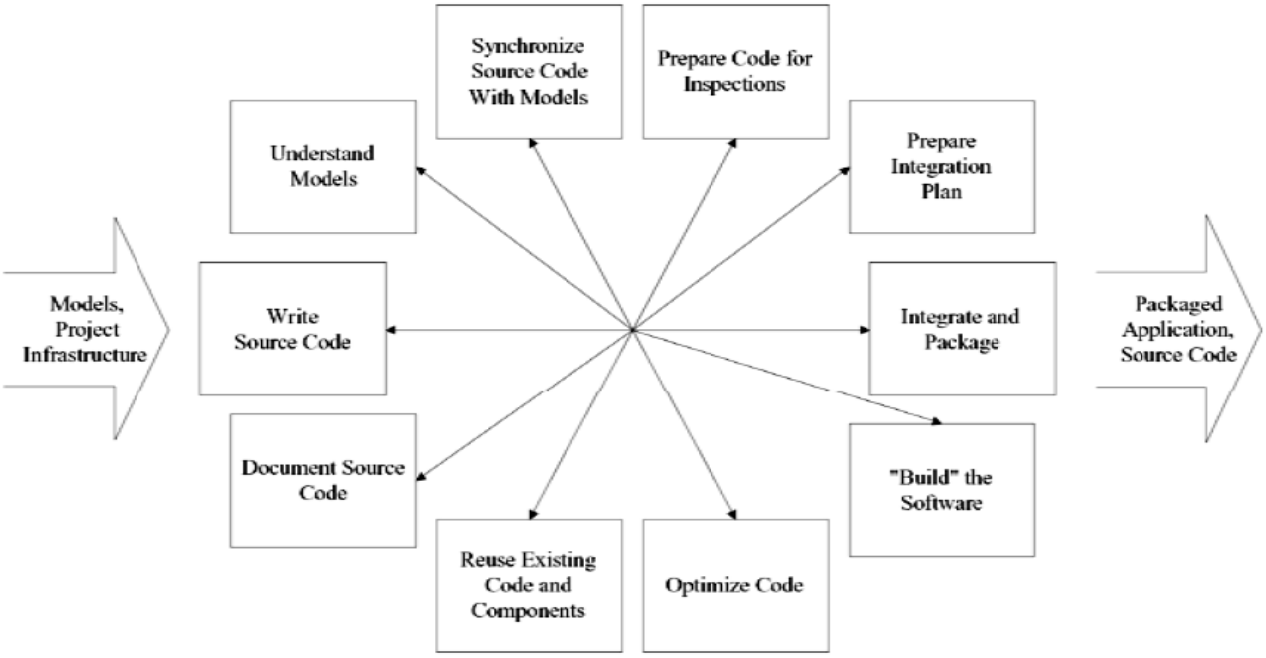
مراحل به این صورت است:

۱- مستنداتی که باید بازبینی شوند آماده می‌شوند.
۲- مدیر بازبینی، آماده بودن مستندات را چک می‌کند.
۳- مدیر یک مرور سریع بر روی مستندات انجام می‌دهد و اگر آماده نبودند به مرحله ۱ برمی‌گردیم.
٤- سه تا پنج نفر برای بازبینی از داخل و خارج سازمان دعوت می‌شوند و امکانات لازم برای بازبینی فراهم می‌شود.
۵- بازبینی انجام می‌شود و اشکالات ثبت می‌شود.
٦- اشکالات به تیم ایجاد داده می‌شوند تا تصحیح شوند و مدیر بازبینی بر کار آنها نظارت می‌کند.

نمونه‌ای از الگوی فرایند مرحله: برنامه‌نویسی[۲۱]

در شکل زیر کارهای مختلفی که در «مرحله برنامه‌نویسی» انجام می‌شوند را مشاهده می‌کنید:

[۲۰] Technical Review
[۲۱] Program



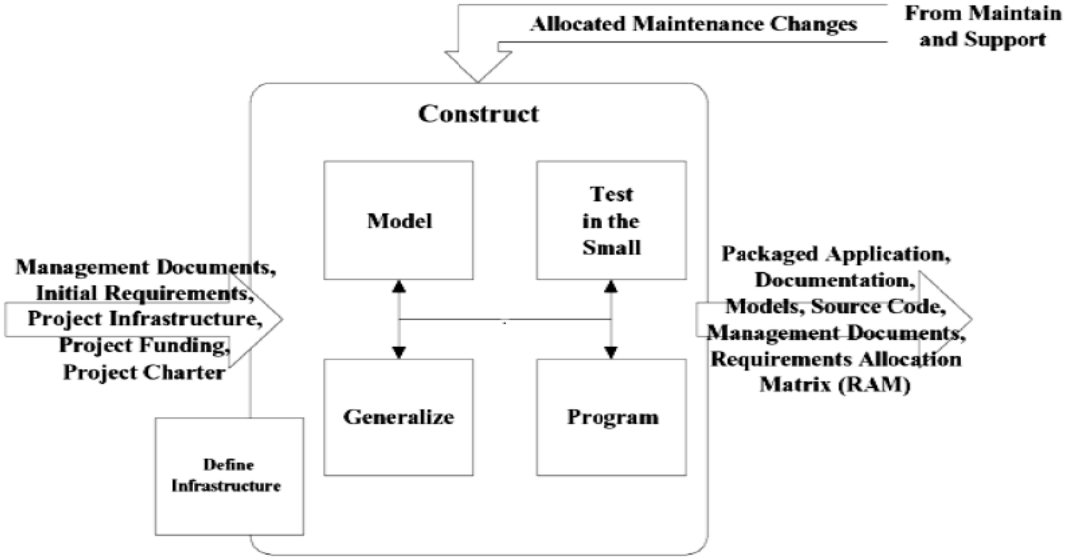
شکل ۲:مرحله برنامه نویسی (Ambler 1998)

در این شکل نشان داده شده است که مدل و معماری و زیرساخت پروژه به عنوان ورودی به این مرحله داده می‌شوند و نرم‌افزار بسته-بندی شده با فرمت مناسب به علاوه متن برنامه‌ها به عنوان خروجی تولید می‌شوند. هر مستطیل در شکل فوق نشان‌دهنده یک کار است که قبلاً تعریف شده است.

نمونه‌ای از الگوی فرایند فاز: ساخت[۲۲]

این فاز، یک فاز ایجاد نرم‌افزار است که در شکل زیر ملاحظه می‌کنید:

[۲۲] Construct

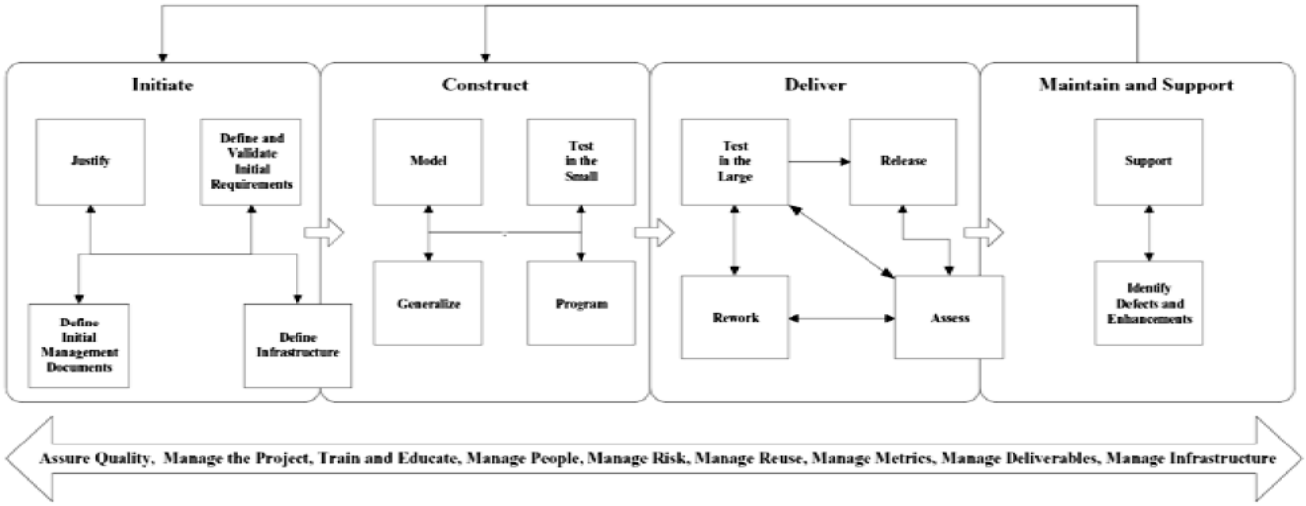


شکل ۳: فاز ساخت (Ambler 1998)

این فاز از چهار مرحله تشکیل شده است که عبارتند از: ۱- Model که در آن مدل‌سازی انجام می‌شود، ۲- Generalize که در آن ساختار و رفتار مشترک چند کلاس بالا کشیده می‌شود و در یک کلاس پدر قرار می‌گیرد، ۳- Program که همانطور که در قسمت قبل مشاهده کردیم در آن برنامه‌نویسی انجام می‌شود، ٤- Test in the Small که در آن تست انفرادی[۲۳] و تست تجمیعی[۲٤] انجام می‌شود.

فرایند Object Oriented Software Process (OOSP)

Ambler با کنار هم قرار دادن فازها، OOSP فرایند کلی ایجاد نرم‌افزار را تعریف کرد. شکل کلی این فرایند به صورت زیر است:



شکل ٤: فرایند OOSP (Ambler 1998)

[۲۳] Unit Test

[۲٤] Integration Test

این فرایند از ٤ فاز تعریف شده است که شکل بالا مشاهده می‌کنید. در فازها فرایند اصلی ایجاد نرم‌افزار انجام می‌شود و فلش زیر شکل، نشان‌دهنده انجام فعالیت‌های چتری[۲۵] در طول پروژه است. ساختار این فازها مشابه فازهای متدولوژی RUP است. بعداً Ambler با ایده گرفتن از OOSP، متدولوژی EUP[۲۶] را ایجاد کرد که گسترشی بر RUP است. ساختار OOSP مشابه فرایند یک متدولوژی سنگین‌وزن و نسل سومی است (رجوع به مستند «متدولوژی اعوان»، بخش متدولوژی‌های شی‌گرا).

- **مزایای الگوهای فرایند و متدولوژی OOSP**

– این متدولوژی کامل و جامع است و چرخه عمر ایجاد نرم‌افزار[۲۷] را می‌پوشاند.

– این متدولوژی تکراری– افزایشی عمل می‌کند (در مراحل می‌توانیم تکرار داشته باشیم).

– مدلسازی ساختاری، رفتاری و وظیفه‌ای سیستم را در همه سطوح انجام می‌دهد.

– بر مبنای مورد کاربرد[۲۸] عمل می‌کند و از UML به عنوان زبان مدلسازی اصلی استفاده می‌کند.

– در آن زمان (سال ۱۹۹۸) که متدولوژی OOSP تعریف شد به بسیاری از مهندسان دید یک متدولوژی جامع را منتقل کرد و روی متدولوژی‌هایی که بعداً طراحی شد تأثیرگذار بود.

- **معایب الگوهای فرایند و متدولوژی OOSP**

– عناصر فرایند OOSP به یکدیگر وابستگی[۲۹] زیادی دارند و نمی‌توان از هر یک از آنها به عنوان یک تکه فعالیت جداگانه استفاده کرد. این خاصیت اگرچه قابلیت فهم و ملموس بودن فرایند را بالا می‌برد ولی باعث می‌شود که نتوان از هر الگوی فرایند (اعم از کار، مرحله و فاز) جداگانه استفاده کرد. این مشکل بسیار جدی است، چون قرار بود این الگوها در مسائل مختلف برای ایجاد یا گسترش متدولوژی‌ها استفاده شوند اما وابستگی بالای آنها مانع از این منظور می‌شود و به نوعی الگو بودن آنها زیر سوال می‌رود.

– فرایند طراحی شده بسیار سنگین است و OOSP مشابه فرایند متدولوژی‌های سنگین‌وزن است.

۲–۱–۳ چارچوب مهندسی متدولوژی OPF

متدولوژی OPEN[۳۰] یک متدولوژی سنگین‌وزن (نسل سومی) بود که از تجمیع چهار متدولوژی در سال ۱۹۹٦ ایجاد شد. پس از اینکه فرایند این متدولوژی به حدی سنگین شد که دیگر به راحتی قابل استفاده نبود، سازندگان این متدولوژی تصمیم گرفتند به جای اینکه

[۲۵] Umbrella Activities مانند مدیریت پروژه، مدیریت ریسک و تضمین کیفیت

[۲۶] Enterprise Unified Process

[۲۷] Software Development Life Cycle (SDLC) که شامل فازهای برنامه‌ریزی (جمع‌آوری نیازمندی‌ها)، تحلیل، طراحی، پیاده‌سازی و نگهداری سیستم است.

[۲۸] Use Case

[۲۹] Coupling

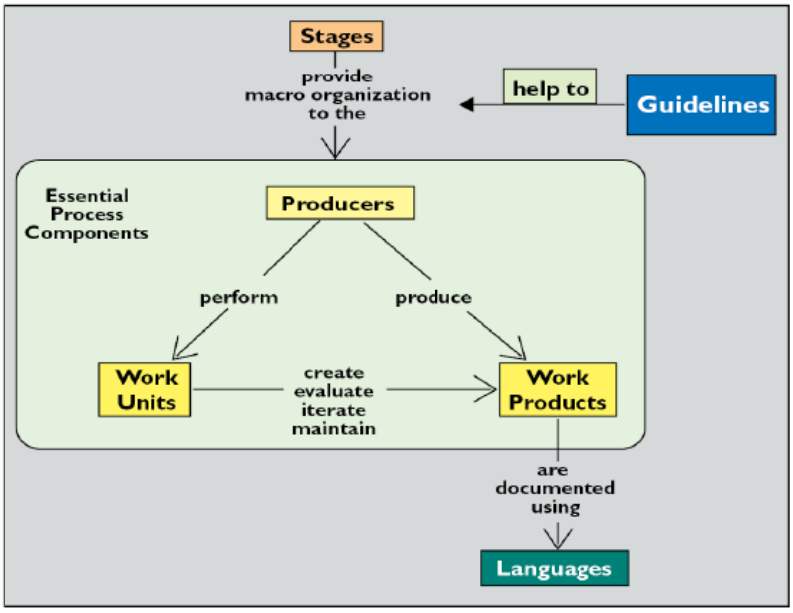
[۳۰] Object-oriented Process, Environment and Notation

یک متدولوژی جامع را ارائه کنند، یک چارچوب ایجاد فرایند به نام OPF[۳۱] و یک کتابخانه از مولفه‌های متدولوژی به نام OPFRO[۳۲] ارائه کنند و هر فرد یا سازمان، متدولوژی خود را با انتخاب مولفه‌ها از کتابخانه و تجمیع آنها طبق چارچوب ایجاد فرایند بسازد.

رابطه چارچوب مهندسی متدولوژی و الگوهای فرایند

هر چارچوب مهندسی متدولوژی (مانند OPF، Rational Method Composer (RMC) و Eclipse Process Framework (EPF)) انواع فعالیت‌ها، نقش‌ها و محصولات را به عنوان تکه‌های متدولوژی در کتابخانه خود دارد که از اتصال مجموعه‌ای از آنها به هم می‌توانیم متدولوژی دلخواه را بسازیم. مجموعه «فعالیت‌های» تعریف شده در کتابخانه‌ی یک چارچوب مهندسی متدولوژی، در اصل همان الگوهای فرایند هستند که از متدولوژی‌های مختلف جمع‌آوری شده‌اند. به عنوان مثال Database Design یک فعالیت در OPF است که یک الگوی فرایند نیز هست و توصیف می‌کند که چگونه پایگاه داده را طراحی کنیم.

در شکل زیر مولفه‌های OPF نشان داده شده‌اند:



For each element (represented by box), OPEN permits the user to select how many and which instances will be used. The OPF documentation provides a comprehensive list of suggestions on the best selections together with guidelines on their best organization.

[Firesmith and Henderson-Sellers 2001]

شکل ۵: مولفه‌های چارچوب مهندسی متدولوژی OPF

همانطور که در شکل بالا مشخص است کتابخانه OPF پنج نوع مولفه را در خود جای داده است:

[۳۱] OPEN Process Framework
[۳۲] OPEN Process Framework Repository Organization

۱- **Work Unit**: که نشان‌دهنده یک واحد کاری است که توسط یک Producer انجام و منجر به ایجاد یا تغییر یک یا چند Work Product می‌شود، مانند مدیریت پیکربندی[۳۳]، مدیریت ریسک، تست و مهندسی نیازمندی‌ها.

۲- **Work Product**: که توسط Producer طی یک Work Unit تولید می‌شود، مانند مدل‌ها، مستندات و نرم‌افزار.

۳- **Producer**: که طی یک Work Unit، یک یا چند Work Product را ایجاد می‌کند یا تغییر می‌دهد. Producer می‌تواند انسان یا ابزار (یک برنامه) باشد. انسانها در قالب نقش‌ها، تیم‌ها و سازمان‌ها مدیریت می‌شوند.

٤- **Stage**: که برای مدیریت Work Unitها استفاده می‌شوند. Work Unitها یک تکه کار از متدولوژی را انجام می‌دهند که برای ایجاد یک متدولوژی کامل نیاز داریم این تکه‌ها را کنار هم بچینیم. برای این کار (یعنی ترکیب و ترتیب دادن به Work Unit ها) از Stage استفاده می‌کنیم. Stage از دو دسته مدت‌دار (مانند فاز یا تکرار) و بدون مدت (مانند Milestone) تشکیل شده است.

۵- **Language**: هر قاعده‌ای (اعم از نحو و معنا[۳٤]) که نحوه ایجاد یک Work Product را توصیف کند. زبان به پنج دسته تقسیم‌بندی می‌شود: ۱- زبان بیان محدودیت‌ها در مدل (مانند OCL[۳۵] که در UML استفاده می‌شود)، ۲- زبان پیاده‌سازی که تشکیل شده است از: زبان‌های برنامه‌نویسی (مثل جاوا)، زبان‌های پایگاه داده (مثل SQL)، زبان‌های تعریف واسط (IDL[۳٦])، زبان‌های پروتکل (مانند TCP/IP) و زبان‌های اسکریپت‌نویسی (مانند Javascript)، ۳- زبان مدل‌سازی، ٤- زبان طبیعی و ۵- زبان توصیف نیازمندی‌ها.

در کنار این پنج دسته مولفه، یک مجموعه **Guideline** برای نحوه قرار دادن این مولفه‌ها در کنار هم ارائه شده‌اند که به کمک آنها می‌توان متدولوژی مناسب‌تری ایجاد کرد.

هر یک از مولفه‌های کتابخانه OPF مانند یک کلاس هستند که ما برای متدولوژی مورد نظرمان، از آنها نمونه‌برداری[۳۷] می‌کنیم و هر نمونه را مطابق با نیاز خود برازش[۳۸] می‌کنیم و آن را در مکان مناسب در متدولوژی قرار می‌دهیم.

همانگونه که در مهندسی نرم‌افزار ابزارهای CASE[۳۹] داریم که به ما در ایجاد نرم‌افزار کمک می‌کنند، در مهندسی متدولوژی نیز ابزارهای CAME[٤۰] داریم که به ما در ایجاد متدولوژی کمک می‌کنند. کتابخانه OPF از نظر محتوا بسیار قوی است اما از نظر ابزار، ابزارهای قوی برای OPF وجود ندارند.

---

[۳۳] Configuration Management

[۳٤] Syntax & Semantic

[۳۵] Object Constraint Language

[۳٦] Interface Definition Language که به کمک آن مولفه‌هایی که یک زبان مشترک ندارند (مثل دو مولفه از زبان C++ و Java) می‌توانند با هم ارتباط برقرار کنند. از استاندارد CORBA و زبان تعریف وب‌سرویس WSDL بر پایه این تکنیک بنا شده‌اند.

[۳۷] Instantiation

[۳۸] Tailoring

[۳۹] Computer-Aided Software Engineering

[٤۰] Computer-Aided Method Engineering

ابزارهای مناسب دیگری برای CAME وجود دارند مانند RMC[۴۱] و EPFC[۴۲]. قویترین ابزار برای مهندسی متدولوژی EPFC است که هر روز کتابخانه آن به‌روز می‌شود. از این کتابخانه فرایند متدولوژی‌های معروف مانند XP نمونه‌سازی شده است و می‌توان به جای ساختن متدولوژی از صفر، نمونه مورد نظر از یکی از متدولوژی‌های دلخواه را انتخاب کنیم و مطابق با نیازمان آن را تغییر دهیم.

۲-۲ پادالگوهای فرایند

پادالگوهای فرایند راه‌حل‌های نامناسب برای مشکلات در فرایند ایجاد نرم‌افزار است. پادالگوهای فرایند اولین بار توسط Brown و همکاران در سال ۱۹۹۸ ارائه شد. پادالگوهای فرایند به سه دسته زیر تقسیم می‌شوند:

۱- **پادالگوهای ایجاد**[۴۳]: پادالگوهایی که افرادی که در فرایند ایجاد نرم‌افزار محصولی تولید می‌کنند (مانند طراحان و برنامه-نویسان) به آنها برمی‌خورند.

۲- **پادالگوهای معماری**: پادالگوهایی که معماران یک سیستم به آن بر می‌خورند.

۳- **پادالگوهای مدیریتی**: پادالگوهایی که در جنبه‌های مدیریتی سازمان یا فرایند متدولوژی ممکن است پیش بیایند.

همه پادالگوهای سه دسته فوق لزوماً پادالگوی فرایند ایجاد نرم‌افزار نیستند و برای ما کاربرد ندارند. در ادامه چند پادالگو از هر دسته که مرتبط با فرایند ایجاد نرم‌افزار نیز هستند را بررسی می‌کنیم.

۲-۲-۱ پادالگوهای ایجاد

پادالگوهایی هستند که در حین فرایند ایجاد نرم‌افزار، اعضای تیم ایجاد ممکن است با آنها مواجه شوند. در زیر پنج نمونه از این پادالگوها که مرتبط با فرایند ایجاد نرم‌افزار هستند را مطالعه می‌کنیم:

۲-۲-۱-۱- جریان مواد مذاب

منظور از «جریان مواد مذاب[۴۴]»، کد مرده[۴۵] و یا طراحی فراموش شده‌ای است که دیگر امکان تغییر آن وجود ندارد و کسی از آن سر در نمی‌آورد (و ممکن است اجرا هم نشود). هنگام ایجاد نرم‌افزار همه‌چیز مانند جریان مواد مذاب زنده و قابل تغییر است. به مرور زمان همانطور که مواد مذاب سرد و سخت می‌شوند، کدها و طراحی‌ها اگر درست مستندسازی و مدیریت نشوند سخت و غیر قابل تغییر و نگهداری می‌شوند (کد مرده اولین نشانه عدم قابلیت نگهداری[۴۶] سیستم است).

---

[۴۱] Rational Method Composer
[۴۲] Eclipse Process Framework Composer
[۴۳] Development
[۴۴] Lava Flow
[۴۵] Dead Code
[۴۶] Maintainability

دلایل

- کدهای تحقیق و توسعه[۴۷] بدون مدیریت و اصلاح در محصول نهایی قرار داده شده‌اند.
- گسترش کنترل نشده‌ی کدی که هنوز تکمیل نشده است، مثلاً متدی که هنوز تکمیل نشده است توسط کلاس‌های دیگر استفاده می‌شود و دیگر تغییر و تکمیل آن متد کار مشکلی است.
- پیاده‌سازی چند روش آزمایشی برای پیاده‌سازی یک متد و حذف نکردن روش‌های آزمایشی نوشته شده.
- «گرگ تنها[۴۸]» کدنویسی و پیاده‌سازی کرده باشد (گرگ‌ها معمولاً دسته‌جمعی حمله می‌کنند و به گرگ خودسر که تنهایی حمله می‌کند گرگ تنها می‌گویند).

گرگ تنها

گرگ تنها فردی است که در تیم به تنهایی طراحی و پیاده‌سازی می‌کند. معمولاً این افراد بسیار باهوش هستند که استانداردهای کدنویسی خاص خود دارند و سعی می‌کنند کدهایشان تا حد امکان بهینه باشد (کد ۱۰۰ خطی را در ۱۰ خط می‌نویسند!). در متدولوژی‌هایی که استانداردهای معین برای کدنویسی تعیین می‌کنند (مانند XP) این افراد را از تیم بیرون می‌کنند، چون فقط خودشان از کد و طراحی خود سر در می‌آورند.

- نداشتن مکانیسم مدیریت پیکربندی و مدیریت فرایند مناسب

مدیریت پیکربندی

مدیریت پیکربندی در مهندسی نرم‌افزار، فرایندی است که طی آن تغییرات دنبال و مدیریت می‌شوند. برای مدیریت پیکربندی محصولاتی که احتمالاً تغییر خواهند کرد شناسایی می‌شوند و مکانیسمی برای نگهداری نسخه‌های مختلف این محصولات، مدیریت تغییرات آنها و نحوه انتشار تغییرات، نظارت و گزارش تغییرات انجام شده ایجاد می‌شود. معمولاً برای مدیریت پیکربندی به یک مخزن محصولات (مانند مخزن کد) نیاز داریم.

- نداشتن و یا نقص معماری برای طراحی و کدنویسی. معماری‌ها اطلاعات مفیدی درباره ساختار کدها و طراحی‌ها در خود دارند که به کمک آنها می‌توان مفهوم طراحی و کدها را بهتر درک کرد.
- داشتن فرایند تکراری. این نکته بسیار مهم است. در اکثر متدولوژی‌ها به خصوص متدولوژی‌های امروزی کارها به صورت تکراری- افزایشی[۴۹] انجام می‌شود. اگر مدت تکرارها و ترخیص‌ها ثابت و قطعی باشد (مثلاً دقیقاً اول هر ماه باید نسخه داده بشود)، افراد اکثراً کدها و طراحی‌ها را ناقص و یا بدون مستندسازی به عنوان محصول ارائه می‌کنند و بعداً نیز اشکالات برطرف نمی‌شود.

---

[۴۷] Research and Revelopment (R&D)

[۴۸] Lone Wolf

[۴۹] Iterative-Incremental

- معماری‌های قدیمی حذف نمی‌شود. چون افراد برای معماری قبلی خود زحمت کشیده‌اند آنها را از نرم‌افزار حذف نمی‌کنند (هرچند معماری قدیمی از چرخه‌ی اجرا خارج می‌شود) و این معماری‌های قدیمی باعث ابهام و پیچیدگی می‌شود.

راه حل

۱- اصلاح[۵۰] و بازرسی[۵۱] کد و طراحی، به منظور بالا بردن کیفیت آنها.
۲- طراحی معماری مناسب برای کدها و طراحی‌های نرم‌افزار.
۳- ایجاد مکانیسم مدیریت پیکربندی برای کنترل تغییرات و نسخه‌های نرم‌افزار.
٤- اگر مشکل این پادالگو در پروژه فعلی وجود دارد می‌توان با مهندسی مجدد سیستم، مشکل را برطرف کرد. برای این کار نیازمندی مربوطه از ابتدا مدل‌سازی می‌شود تا به مرحله طراحی و پیاده‌سازی برسد و بعد با مقایسه طراحی‌ها و پیاده‌سازی‌ها، اشکالات را برطرف می‌کنند.

۲-۲-۱-۲- دیدگاه مبهم

در پادالگوی «دیدگاه مبهم[۵۲]» مدل‌ها بدون یک دیدگاه مشخص تولید می‌شوند. برای مدل‌ها در نرم‌افزار سه دیدگاه متفاوت وجود دارد:

۱- **دیدگاه Business (Problem Domain / Conceptual / Essential)**: که در آن «همه‌ی موجودیت‌های قلمرو مسئله» (مانند موجودیت‌های سازمان مورد نظر) شناسایی و مدل می‌شوند. ممکن است این موجودیت‌ها لزوماً در سیستم مورد نظر ما قرار نگیرند ولی مدل‌سازی آنها باعث پیدا کردن دید نسبت به آنها (به خصوص در مواقعی که تعامل سیستم ما با سایر سیستم‌ها مهم است) می‌شود. ممکن است قبل از انجام پروژه این مدل‌ها توسط متخصصین سازمان مورد نظر تولید شده باشند.
۲- **دیدگاه Analysis (System / Specification)**: این مدل، مدل تحلیل است که در آن «موجودیت‌های داخل سیستم» که در قلمرو مسئله وجود دارند کشف می‌شوند. اگر مدل Business را قبلاً ایجاد کرده باشیم، ایجاد این مدل از روی آن مدل به راحتی امکان‌پذیر است.
۳- **دیدگاه Design (Software / Implementation)**: این مدل، مدل طراحی است که در آن موجودیت‌های داخل سیستم از قلمرو مسئله و قلمرو جواب وجود دارند. این مدل، همه موجودیت‌های داخل سیستم را مدل می‌کند (مفهوم قلمرو مسئله و قلمرو جواب در مستند «متدولوژی اعوان» بخش تحلیل مقدماتی توضیح داده شده است).

فایده جداسازی دیدگاه‌ها این است که اولاً هر دیدگاه کاربرد خود را دارد و مخلوط کردن مدلها از دیدگاه‌های مختلف باعث ابهام در مدل‌سازی دارد. علت دیگر جداسازی مدل‌ها در دیدگاه‌های مختلف این است که دیدگاه‌های با سطح انتزاع بالاتر (یعنی مدلهای Business و Analysis)، مدل‌های مناسب‌تری برای مهندسی مجدد[۵۳] نرم‌افزار در خود دارند چون این مدل‌ها بیشتر مربوط به قلمرو

---

[۵۰] Refactoring
[۵۱] Inspection
[۵۲] Ambiguous Viewpoint
[۵۳] Reengineering

مسئله هستند و آنچه ما در مهندسی مجدد نیاز داریم اطلاعات قلمرو مسئله است نه قلمرو جواب (چون قلمرو جواب فقط خاص سیستم قبلی است که الان دیگر قابل استفاده نیست).

راه‌حل

در فازهای مشخص، مدل‌سازی هر دیدگاه به طور جداگانه انجام شود.

۲-۲-۱-۳- چکش طلایی

«چکش طلایی[۵۴]» به تکنولوژی یا مفهومی گفته می‌شود که به تعداد زیادی از مشکلات نرم‌افزاری (به صورت عقده‌ای!) اعمال می‌شود. شعار معروف این پادالگو این است که «وقتی تنها ابزار چکش است، بقیه چیزها میخ است!».

راه‌حل

دانش تولیدکنندگان نرم‌افزار باید طی دوره‌های آموزش، تمرین و مطالعه کتاب به روز شود (دوره‌های Refreshing انجام شود) تا افراد بتوانند گزینه‌های مختلف را برای حل مسائل در نظر بگیرند و بهترین راه‌حل را انتخاب کنند.

۲-۲-۱-۴- قدم زدن در میدان مین

منظور از «قدم زدن در میدان مین[۵۵]» این است که استفاده از تکنولوژی‌های امروزی مشابه قدم زدن در یک میدان مین است که ممکن است خطاهای گسترده‌ای در حین و بعد از تحویل محصول به مشتری بروز کنند. این پادالگو این اخطار را می‌کند که اگر بدون آگاهی و برنامه در این میدان مین قدم بگذارید قطعاً شکست خواهید خورد و قصد دارد به اهمیت تست در محیط نرم‌افزاری فعلی اشاره کند.

راه‌حل

۱- باید سرمایه‌گذاری مناسب بر روی تست نرم‌افزار انجام شود برای اینکه نرم‌افزارها را تا حد خوبی خطازدایی کنند. در برخی شرکت‌های پیشرفته نرم‌افزاری تعداد افراد تیم تست بیشتر از تیم برنامه‌نویسی است.

۲- از روش «ایجاد مبتنی بر تست[۵۶]» برای ایجاد نرم‌افزار استفاده کنیم و یا حداقل تست را به عنوان یک فاز از فرایند در نظر بگیریم. در روش ایجاد مبتنی بر تست، ابتدا تست یک قطعه برنامه (مانند متد یا کلاس) نوشته می‌شود و بعد کد آن بخش از برنامه نوشته می‌شود.

۳- باید مکانیسم بازبینی کیفیت مانند جلسات دوره‌ای بازبینی محصول و نیز روش‌های بازرسی کد و طراحی اعمال شود.

٤- باید موارد تست[۵۷] به خوبی مدیریت پیکربندی شوند و نتایج اجرای آنها به عنوان گزارش ثبت شود.

---

[۵۴] Golden Hammer

[۵۵] Walking through a Minefield

[۵۶] Test Driven Development (TDD)

[۵۷] Test Cases

۵- تولید موارد تست و اجرا و گزارش‌گیری از آنها باید به صورت اتوماتیک انجام شود.

۲-۲-۱-۵- مدیریت قارچی

در برخی چرخه‌های مدیریتی، یک سیاست آشکار این است تیم ایجاد (از جمله طراح و برنامه‌نویس) را از کاربران سیستم جدا کنند و این افراد با یکدیگر هیچ ارتباطی نداشته باشند. نیازمندی‌ها به صورت دست دوم از طریق واسطها (اعم از معمار، مدیر و تحلیل‌گر نیازمندی‌ها) به تیم ایجاد می‌رسد. شعار این افراد این است که «تیم ایجاد را در تاریکی نگه دار و تفاله اطلاعات را به آنها بده». در تاریکی نگه داشتن تیم ایجاد مثل این است که آنها را مانند قارچ در یک محیط ایزوله نگهداری کنید. منظور از این شعار این است که تیم ایجاد را باید از مشتری دور نگه داشت تا ارتباط آنها با یکدیگر، مسئولیت جدید برای تیم پروژه ایجاد نکند.

مدیریت قارچی فرض می‌کند که نیازمندی‌ها در شروع پروژه به طور کامل توسط تحلیل‌گران پروژه و نیز کاربران فهمیده می‌شود. ضمناً نیازمندی‌ها در طول پروژه ثابت هستند. اما اگر چنین بود روش آبشاری، هرگز به روش تکراری – افزایشی تبدیل نمی‌شد. اما امروزه ثابت شده است که امکان ندارد نیازمندی‌ها از ابتدای پروژه شناسایی و تا انتهای پروژه ثابت باشند.

راه‌حل

استفاده از روش حلزونی[۵۸] (رجوع به مستند «متدولوژی اعوان» بخش روش‌های ایجاد نرم‌افزار) با کمک نمونه‌سازی[۵۹] و بازخورد گرفتن از کاربر. روش حلزونی کمک می‌کند ریسک «عدم شناخت نیازمندیها» با کمک ساختن نمونه‌های اولیه و گرفتن نظر کاربر، کم شود و چون به صورت دوره‌ای محصول را برای کاربر نصب می‌کند کاربر در هر مرحله سیستم را می‌بیند و سیستم طبق نظر کاربر جلو می‌رود (در روش آبشاری کاربران در آخر پروژه سیستم را می‌دیدند که ممکن بود کلاً با نظر آنها مخالف باشد).

۲-۲-۲ پادالگوهای معماری

پادالگوهایی هستند که معماران یک سیستم (اعم از نرم‌افزاری یا غیرنرم‌افزاری) به آن بر می‌خورند. ۵ نمونه از پادالگوهای معماری که مرتبط با فرایند ایجاد نرم‌افزار هستند عبارتند از:

۲-۲-۲-۱- هوای خودت را داشته باش

منظور از «هوای خودت را داشته باش[۶۰]» این است که در فرایندهایی که باید در آنها مستندات تولید می‌شود، برخی افراد مستندات حجیمی تولید می‌کنند بدون اینکه تصمیم مهمی در این مستند گرفته شود. افراد با نگرفتن تصمیمات مهم هوای خودشان را دارند و

---

[۵۸] Spiral

[۵۹] Prototyping

[۶۰] Cover Your Assets

تصمیمی نمی‌گیرند که احیاناً در آینده یقه آنها را بگیرند. این افراد مستندات حجیم می‌سازند و انواع گزینه‌ها[۶۱] را به خوبی تشریح می‌کنند و مزایا و معایب هریک را می‌گویند اما در نهایت هیچ تصمیمی نمی‌گیرند و خواننده دچار ابهام می‌شود.

راه حل

مستندنویسان را مجبور به تولید «**طرح معماری**[۶۲]» بکنید. طرح معماری حدود ۱۰–۱۲ جدول و نمودار است که معماری (دید کلی[۶۳]) عملیاتی و تکنیکی سیستم جاری و سیستم آتی را شرح می‌دهد (یعنی تغییراتی که در سیستم جاری قرار است اعمال شود) و باید بتوان آن را در کمتر از یک ساعت ارائه کرد. طرح معماری یک دید انتزاعی از سیستم بدست می‌دهد که ارتباط بین کاربر و تیم ایجاد را بهبود می‌بخشد.

با کمک طرح معماری فرد مجبور می‌شود از طولانی کردن مستندات خودداری کند و منظورش را در چند شکل بیان کند. در صورتی که فرد نتوان همه منظورش را در ۱۰–۱۲ نمودار بیان کند، باید علاوه بر طرح معماری، مستند دیگری که جزییات را شرح می‌دهد ارائه دهد.

۲–۲–۲–۲– معماری مفروض

منظور از «معماری مفروض[۶۴]» این است که برخی از سازمان‌های نرم‌افزاری از معماری یکسان برای همه پروژه‌ها استفاده می‌کنند و در نتیجه دیگر معماری منطقی و فیزیکی را مدل نمی‌کنند و مستند معماری را برای هر پروژه به طور جداگانه نمی‌نویسند. مشکل این کار این است که از مدیریت ریسک در طول فرایند ایجاد نرم‌افزار به دلیل اطمینان کاذب به موفقیت سیستم‌های قبلی این معماری چشم‌پوشی می‌شود.

راه حل

یک تعریف عمومی و کلی برای معماری در قالب مستند و نمودارهای معماری ایجاد می‌شود که در هر سیستم، مناسب با نیازمندی‌های سیستم و ریسک‌های سیستم برازش می‌شوند.

۲–۲–۲–۳– طراحی در کمیته

«طراحی در کمیته[۶۵]» به طراحی‌هایی گفته می‌شود که چند نفر با ایده‌هایی که از تجربه‌های قبلی دارند دور هم می‌نشینند و برای یک کار جدید ایده‌های قبلی خود را تجمیع می‌کنند. معمولاً نتیجه این طراحی، یک طرح بسیار سنگین است که امکان ندارد با هیچ گروه برنامه‌نویسی آنها را پیاده‌سازی کرد. اگر هم بتوان پیاده‌سازی کرد امکان ندارد بتوانیم کل طراحی را تست کنیم و از صحت پیاده‌سازی مطمئن

---

[۶۱] Alternative

[۶۲] Architecture Blueprint

[۶۳] Big Picture

[۶۴] Architecture by Implication

[۶۵] Design by Commitee

شویم. اجزای این طراحی سنگین به دلیل آنکه نظرات افراد مختلف در آن شرکت داده شده است، چسبندگی[۶۶] بسیار پایینی دارد و مفهوم و تعریف واضحی ندارد.

یک کمیته نمی‌تواند به خوبی طراحی کند چون اولاً فرصت کافی برای طراحی معمولاً در یک جلسه گروهی داده نمی‌شود و ثانیاً هرکس سعی در قالب کردن نظر خود دارد.

راه حل

۱- نقش‌های معماری را در پروژه شناسایی کنیم و به نظرات آنها وزن بیشتری بدهیم.

۲- طراحی را یک نفر یا یک تیم کوچک در مدت زمان کافی انجام دهند و نتیجه به تصویب کمیته برسد. کمیته می‌تواند به خوبی تصمیم‌گیری کند.

۲-۲-۲-۴- اختراع مجدد چرخ

«اختراع مجدد چرخ[۶۷]» به یک مشکل رایج در مهندسی نرم‌افزار اشاره می‌کند که در پروژه‌های مختلف، دانش و تجربه آنها ثبت نمی‌شود و در پروژه‌های بعدی نمی‌توانیم از تجربه‌های پروژه‌های قبلی استفاده کنیم، پس مجبور می‌شویم دوباره و چندباره برای گرفتن تصمیمات، انجام طراحی‌ها و پیاده‌سازی‌ها وقت و منابع صرف کنیم و در نتیجه چرخ‌های اختراع شده در قبل را مجدداً اختراع کنیم.

اکثر افرادی که چرخ را دوباره اختراع می‌کنند شعارشان این است که «مشکل ما یکتاست» و نمونه مشابهی برای آن نیست، اما عملاً در حال حاضر امکان ندارد پروژه‌ای در نرم‌افزار انجام شود که با پروژه‌های قبلی وجه اشتراکی نداشته باشد.

منظور از اختراع نکردن مجدد چرخ این است که نه تنها از تجربیات گذشته تیم خود استفاده کنیم بلکه از تجربیات گذشته سایر تیم‌ها و سازمان‌ها نیز استفاده کنیم. نمود بارز استفاده از تجربیات سایر افراد در مهندسی نرم‌افزار، الگوها و پادالگوها هستند که در هر حوزه‌ای وجود دارند.

راه حل

دانش مدفون شده در سیستم‌های قدیمی و یا سیستم جاری باید استخراج شود تا هزینه، ریسک و زمان تولید محصولات جدید کاهش یابد. امروزه اکثر متدولوژی‌ها پس از پایان پروژه و یا پایان هر تکرار یا فاز پروژه، تجربیات کسب شده از آن مرحله (اعم از مشکلات، راه حل آنها و محیطی که مشکلات در آنها رخ داده‌اند) را ثبت می‌کنند تا در پروژه‌های بعدی و یا تکرارها و فازهای بعدی همان پروژه استفاده کنند. به این کار، فعالیت‌های پس از مرگ[۶۸] گفته می‌شود.

---

[۶۶] Cohesion

[۶۷] Reinvent the Weel

[۶۸] Postmortem Activities

۲-۲-۲-۵- پادشاه قدیمی شهر یورک

«پادشاه قدیمی شهر یورک[۶۹]» نام یک پادالگوست که در آن مدیران با دیدگاه قدیمیِ تساوی‌طلبانه در تصمیم‌گیریها به رأی همه افراد تیم وزن یکسان می‌دهند. افراد تیم پروژه معمولاً به دو دسته تقسیم می‌شوند:

۱- **افراد با دید انتزاعی**[۷۰]: افرادی هستند که می‌توانند انتزاعی از یک موضوع را برای خود تصور کنند. معمولاً این افراد را برای کارهای کشف نیازمندی‌ها، تحلیل، طراحی معماری و کارهای سطح بالای پروژه انتخاب می‌کنند. این افراد به راحتی می‌توانند با مشتری ارتباط برقرار کنند، چون می‌توانند خصوصیات سیستم را برای مشتری شرح دهند و در صورت نیاز سیستم را به اشیا و اتفاقات دنیای واقع تشبیه کنند.

۲- **افراد با دید پیاده‌سازی**[۷۱]: افرادی هستند که نمی‌توانند انتزاعی از یک موضوع یا یک بخش از پروژه را تصور کنند. این افراد را معمولاً برای کارهای طراحی، پیاده‌سازی و تست که به پیاده‌سازی نزدیک است انتخاب می‌کنند. این افراد اگر برای کارهای با انتزاع بالا مثل کشف نیازمندی‌ها انتخاب شوند، نیازمندی‌هایی که این افراد تعریف می‌کنند دارای تعاریف سطح پایین (در سطح کد) و با دید پیاده‌سازی است (در حالیکه نیازمندی‌ها باید از دید کاربر و سطح بالا باشند).

معمولاً در یک تیم تعداد افراد با دید پیاده‌سازی چهار برابر تعداد افراد با دید انتزاعی است. اگر قرار باشد برای تصمیم‌گیری در یک پروژه به همه افراد با دید انتزاعی و با دید پیاده‌سازی وزن یکسان بدهیم، همواره افراد با دید پیاده‌سازی در تصمیم‌گیریها پیروز می‌شوند. درنتیجه قابلیت استفاده و نگهداری سیستم مورد تهدید واقع می‌شود چون معماری مناسبی برای سیستم، مولفه‌ها و کلاس‌ها ایجاد نمی‌شود و این موضوع باعث مرگ زودرس سیستم می‌شود (برای تعریف دقیق نگهداری و مرگ سیستم به مستند «متدولوژی اعوان» مراجعه کنید).

راه حل

برای افراد نقش‌های متفاوتی در تصمیم‌گیری قائل شویم و به نظرات افراد با دید انتزاعی وزن بیشتری بدهیم.

۲-۲-۳ پادالگوهای مدیریتی

پادالگوهایی هستند که در جنبه‌های مدیریتی سازمان یا فرایند متدولوژی ممکن است پیش بیایند را توصیف می‌کنند. در زیر ٤ مورد از این پادالگوها که مرتبط با فرایند ایجاد نرم‌افزار هستند شرح داده می‌شود:

---

[۶۹] The Grand Old Duke of York

[۷۰] Abstractionist

[۷۱] Implementationist

۲-۲-۳-۱- فلج تحلیل

منظور از «فلج تحلیل[۷۲]» این است که سعی می‌شود فاز تحلیل به طور کامل انجام شود و بعد پروژه ادامه یابد. این کار باعث می‌شود تا پروژه در فاز تحلیل قفل و متوقف شود و حجم عظیمی از نیازمندی‌ها و مدل‌ها بدست بیایند که فهم آنها مشکل است.

یک دسته از افرادی که باعث این پادالگو می‌شوند افرادی هستند که به تازگی تحلیل و طراحی سیستم‌ها را یاد گرفته‌اند و سعی دارند همه روش‌هایی که یاد گرفته‌اند در پروژه جاری اعمال کنند تا تسلط خود را نشان دهند. این کار باعث سنگین شدن مدل‌ها و طولانی شدن فاز تحلیل می‌شود.

یکی از نشانه‌های اصلی این پادالگو این است که مستندات تحلیل دیگر برای متخصصان حوزه[۷۳] قابل فهم نیست. مستندات تحلیل باید همواره برای متخصصان حوزه از سازمان مشتری قابل فهم باشند. این مستندات برای مهندسی مجدد سیستم بسیار مفید هستند.

راه حل

استفاده از روش تکراری – افزایشی به منظور به تعویق انداختن تحلیل تفصیلی تا موقعی که واقعاً به آن نیاز داریم.

۲-۲-۳-۲- مرگ در برنامه‌ریزی

«مرگ در برنامه‌ریزی[۷۴]» در دو حالت ممکن است رخ دهد:

۱- اصرار داریم که در ابتدای پروژه برنامه کامل را در بیاوریم و بعد ادامه دهیم،

۲- در ادامه پروژه، به هیچ وجه برنامه را تغییر ندهیم.

حالت اول ذکر شده در بالا بیشتر منظور این پادالگو است. برخی افراد (به خصوص کسانی که به روش آبشاری علاقه خاصی دارند!) سعی می‌کنند در هر فاز کار را به طور کامل انجام دهند و در فاز ابتدایی پروژه نیز سعی می‌کنند برنامه دقیق کارها و مدت زمان آنها را دربیاورند. این کار علاوه بر تعلیق پروژه و طولانی شدن زمان آن، باعث برنامه‌ریزی و زمان‌بندی‌های پیچیده برای پروژه می‌شود که ممکن است برای مدیران رده بالا به راحتی قابل فهم نباشد.

راه حل

۱- به جای سعی در برنامه‌ریزی دقیق برای کل پروژه در اول کار، برنامه‌ریزی تکرار (یا فاز) فعلی را به صورت دقیق در بیاوریم و برنامه تکرارهای (یا فازهای) بعدی را به صورت تقریبی تعریف کنیم.

۲- بر اساس محصول برنامه‌ریزی کنیم نه بر اساس زمان، یعنی به جای آنکه برای فاز زمان تعیین کنیم، بگوییم در انتهای هر فاز چه محصولی تولید می‌شود.

۳- برنامه‌ها را در بازه‌های مشخص (مثلاً هر هفته) با توجه به سرعت تیم ایجاد، بازنگری کنیم.

٤- در میانه فازها و تکرارها Milestone تعریف کنیم تا بر اساس قانون اول پارکینسون همه کارها به انتهای فازها محول نشود.

---

[۷۲] Analysis Paralysis

[۷۳] Domain Experts

[۷٤] Death by Planning

قانون اول پارکینسون

یک کار تا زمانیکه به آن مهلت داده شود طول می‌کشد، یعنی زمان انجام یک کار با توجه به مهلت داده شده به آن کِش می‌آید! برای رفع این مشکل کار را به چند قسمت می‌شکنند و برای آن کار چند Milestone تعریف می‌کنند که در هر کدام یک بخش از کار تحویل داده می‌شود.

۲-۲-۳-۳- سوء مدیریت پروژه

در پادالگوی «سوء مدیریت پروژه[۷۵]» به فرایند مدیریت پروژه توجه نمی‌شود (یا کلاً انجام نمی‌شود و یا به صورت ناقص انجام می‌شود) و پروژه به درستی کنترل و نظارت نمی‌شود و فعالیت‌های برنامه‌ریزی و تضمین کیفیت فرایند ایجاد و محصول و برنامه انجام نمی‌شود. این پادالگو باعث می‌شود که فرایند ایجاد نرم‌افزار بدون جهت (و یا در جهت غلط) انجام شود و مشکلات پروژه به موقع تشخیص داده نشوند.

نظارت و کنترل صحیح پروژه برای انجام صحیح فعالیت‌های ایجاد نرم‌افزار لازم است.

راه حل

۱- فرایند نظارت و کنترل فعالیت‌های پروژه باید به طور دقیق تشریح شوند.

۲- مکانیسم مدیریت ریسک در پروژه به کار گرفته شود.

۳- برنامه، محصول و فرایند ایجاد به طور دوره‌ای مورد بازبینی قرار گیرند تا کیفیت آنها بررسی شود.

۲-۲-۳-۴- ترس از پیروزی

منظور از «ترس از پیروزی[۷۶]» این است که افراد تیم از پایان پروژه یا یک بخش از پروژه می‌ترسند به علت اینکه ممکن است مشکلی در کارهای آنها وجود داشته باشد. افراد با انجام تغییرات و کارهای گوناگون سعی دارند پروژه به پایان نرسد.

راه حل

- مهلت هر تکرار از قبل اعلام می‌شود و سخت‌گیرانه اجرا می‌شود.
- در پایان هر تکرار به اعضای تیم مرخصی داده می‌شود تا از فکر تکرار قبلی بیرون بیایند و برای تکرار بعدی آماده بشوند.
- باید از ابتدا و یا با نزدیک شدن به پایان پروژه، معیارهای اتمام پروژه به دقت تعیین و به همه اعلام شود.

---

[۷۵] Project Mismanagement

[۷۶] Fear of Success

بخش ۳ مراجع

[۱] **الگوها و پادالگوها**: مهمترین منبع این مستند، اسلایدهای درس «الگوها در مهندسی نرم‌افزار» است که توسط دکتر رامسین تدریس می‌شود. برای مشاهده این اسلایدها و منابع مربوطه می‌توانید به صفحه خانگی دکتر رامسین به آدرس http://sharif.edu/~ramsin مراجعه کنید.

[۲] **الگوها**: یکی از قدیمی‌ترین و معتبرترین سایت‌های مرجع برای الگوها سایت http://hillside.net است. این سایت در حال حاضر آخرین الگوها را منتشر می‌کند. در این سایت یک کاتالوگ از چندین دسته الگو در آدرس http://hillside.net/patterns/onlinepatterncatalog.htm موجود است.

[۳] **الگوهای فرایند**:

- **الگوهای Coplien**: "A development process generative pattern language" که در اولین کنفرانس Pattern Languages of Programming (PLoP) در سال ۱۹۹٤ ارائه شد.
- **الگوهای Ambler**: نسخه اولیه کتابش به نام "Process Patterns: Building Large-Scale Systems Using Object Technology" انتشارات دانشگاه کمبریج در سال ۱۹۹۸ منتشر شد و نسخه دوم آن به نام "More Process Patterns: Delivering Large-Scale Systems Using Object Technology" انتشارات دانشگاه کمبریج در سال ۱۹۹۹ منتشر شد.
- **چارچوب مهندسی متدولوژی OPF**: کتابخانه OPF در آدرس http://opfro.org در دسترس است. در این وب‌سایت هر تکه عنصر از متدولوژی توصیف شده است و ارتباطش با بقیه عناصر شرح داده شده است.

[٤] **پادالگوهای فرایند**: "Antipatterns: Refactoring Software, Architectures, and Projects in Crisis" اثر Brown، Malveau، McCormick و Mowbray، انتشارات Wiley در سال ۱۹۹۸. ضمناً وب‌سایت http://www.antipatterns.com/catalog.htm هم مفید است (البته همه پادالگوهای این وب‌سایت فرایندی نیستند).