دانشگاه صنعتی شریف

دانشکده‌ی مهندسی کامپیوتر

تمرین دوم متدلوژی‌های ایجاد نرم‌افزار

نگارنده

محمدرضا لطفی‌نمین ـ ۹۶۲۱۰۱۷۸

استاد درس

دکتر رامسین

نیمسال اول ۱۳۹۷ـ۱۳۹۸

فهرست مطالب

فصل ۱

بررسی متدولوژی USDP

۱.۱ بخش فرآیند

۱.۱.۱ تعریف متدولوژی

این متدولوژی به شکل فرآیند محور و چرخشی‌تکراری تعریف شده است و فازهای درشت دانه و فعالیت‌های تکرار شونده در هر فاز دقیق توصیف شده‌اند. بخشی از چرخه‌ی نرم‌افزار که پوشش داده شده است دقیق تعریف شده ولی در بخش بعد می‌بینیم که پوشش کاملی ندارد و مطلوب نیست زیرا اگر هم بخشی را عمدا پوشش نمی‌دهد باید مواردی که متدولوژی ندارد صراحتاً ذکر شود و روش کامل کردن متدولوژی حداقل در حد کلیات مطرح می‌شد که مطرح نشده است (شبیه کاری که FDD انجام داد).

متدولوژی به شکل فرآیندمحور تعریف می‌شود و نقش‌های درگیر تا حد خوبی تعریف شده‌اند و محصولات با دقت کامل تعریف شده‌اند.

زبان مدل‌سازی متدولوژی به شکل صریح UML است و توصیف کامل آن توسط ObjectManagement-Group به شکل دقیق تعریف شده است. (البته syntax دقیق است ولی semantic خیلی قوی نیست و تفسیرش بر عهده کاربرد خاصش است).

محصولات این متدولوژی به خوبی تعریف شده است و هم در قالب محصولات ریزدانه تولید شده در Dis-cipline ها است و برای فازها هم لیست محصولات به شکل کلی‌تر همراه با خصائص کیفی که باید داشته باشند (مشابه DefinitionOfDone در اسکرام) آورده شده است.

هرجا تکنیکی مفید بوده و به آن اشاره شده است به قدر کافی دقیق و بدون ابهام تعریف شده است. مثلا اشاره شده است که از دو نوع پروتوتایپ Proof of Concept و Technical استفاده می‌شود و دقیقا مشخص شده است که پروتوتایپ دور ریختنی است و قرار نیست همان را تکمیل کنیم و به نرم‌افزار اصلی تبدیل شود.

متدولوژی به فعالیت‌های چتری توجه دارد اما discipline مربوط به آن‌ها که در RUP صراحتاً وجود داشت حذف شده است و کمی کمرنگ‌تر است ولی به هرحال برنامه‌ریزی و مدیریت ریسک و کنترل کیفیت در متدولوژی

بسیار جدی گرفته می‌شود و فقط Discipline هایش حذف شده است.
در کل نقص در تعریف دیده می‌شود اما ناسازگاری و یا نادقیق بودن مطالبی که عنوان شده است دیده نمی‌شود.

۲.۱.۱ پوشش چرخه‌ی عمومی تولید نرم‌افزار

این متدولوژی فعالیت استخراج نیازمندی را در قالب discipline نیازمندی‌ها در نظر دارد. در فاز inception تحلیل امکان سنجی انجام می‌دهد و تا انتها در قالب discipline تحلیل به مدل‌سازی قلمرو مسئله می‌پردازد. هم‌چنین برای طراحی معماری در فاز Elaboration معماری قابل اجرا به عنوان خروجی اصلی در نظر گرفته شده است. طراحی جزئی هم در discipline طراحی در فازهای بعد از Inception وجود دارد و پیاده‌سازی و تست را هم در قالب دو discipline در خود جای داده است. ضمناً می‌توان گفت که برای استقرار هم فاز Transition هست. پس تنها بخشی که پوشش داده نشده است فاز Maintenance است که نه به فعالیت‌های پشتیبانی و نه فعالیت‌های نگهداری و در نهایت مرگ نرم‌افزار نپرداخته است.

۳.۱.۱ پشتیبانی از فعالیت‌های چتری

مدیریت ریسک

یکی از مهم ترین اهداف در فاز Inception از این متدولوژی تحلیل امکان سنجی اولیه قبل از شروع واقعی پروژه است که زمان کمی می‌برد و ریسک را به شدت کاهش می‌دهد. همچنین این متدولوژی در تمامی مراحل ریسک را مورد توجه دارد و در واقع نوعی risk-based جلو می‌رود. همچنین تاکید ویژه دارد که تا پایان Elaboration باید تمام ریسک‌ها از بین رفته باشند و حتی usecase های ریسکی باید پیاده سازی شوند. استفاده از دو نوع Proof of concept prototype و Technical prototype هم در این متدولوژی توصیه شده که ریسک را کاهش می‌دهد. ورود سریع به قلمرو جواب (که می‌تواند از اواخر Inception و شروع استخراج معماری باشد) هم ریسک را کاهش می‌دهد. علاوه بر همه این نکات متدولوژی به شکل Iterative & Incremental اجرا می‌شود که خود باعث کاهش ریسک است. همچنین محوریت usecase در تمامی مراحل Continous Verification را تضمین می‌کند که ریسک را کاهش می‌دهد. البته Continous Validation در حدی که در اجایل‌ها به آن تاکید شده نیست ولی به هرحال چون چرخشی و تکراری است می‌توان آن را هم در قالب مرحله‌ی تست متدولوژی در نظر گرفت. متدولوژی از روش‌های فرمال مدل‌سازی می‌تواند استفاده کند که در سیستم‌های بحرانی می‌تواند ریسک را کاهش دهد. ضمناً هر چند به اندازه agile ها در آن ContinousIntegration نداریم ولی به هر حال در پایان هر تکرار یک قطعه نرم‌افزار جدید باید با قبلی‌ها ترکیب شود پس حدی از CI را هم دارد که ریسک را مدیریت می‌کند. وجود فعالیت جداگانه در قالب Test هم می‌تواند در کاهش ریسک موثر باشد. همچنین محصولات خروجی هر فاز به دقت در قالب یک چک‌لیست با وجود خصیصه‌های کیفیشان تعریف شده است که می‌تواند برای کنترل کیفیت استفاده شود که ریسک اینکه محصولی تولید نشود یا کیفیتش کم باشد کاهش می‌یابد. تبیین صریح توجیه اقتصادی در ابتدای پروژه و بازبینی مکرر آن تا انتهای پروژه هم مصداقی از مدیریت ریسک‌های مالی است.

مدیریت پروژه

مدیریت پروژه در این متدولوژی نقش جدی دارد و در پایان هرفاز باید یک برنامه‌ریزی دقیق برای چند تکرار اول فاز بعد و یک برنامه‌ریزی کلی‌تر برای تکرارهای دورتر باشد. همچنین بر بازبینی این برنامه‌ریزی در طول اجرا تاکید جدی دارد. اما متاسفانه discipline مربوط به فعالیت مدیریت پروژه را حذف کرده است. البته یک جنبه از مدیریت پروژه مدیریت منابع انسانی و روابط تیم‌ها و افراد است که این متدولوژی به قدر کافی به آن توجه نکرده است.

تضمین کیفیت

همانطور که گفتیم محوریت usecase باعث ایجاد Continous Verification می‌شود که کیفیت را بهبود می‌دهد و به صورت خودکار رهگیری مستقیم به نیازمندی ایجاد می‌کند که آن هم کیفیت را بهبود می‌دهد. همچنین تعریف دقیق معیارهای کیفیت در مرحله‌ی Elaboration که به آن تاکید شده است می‌تواند کیفیت محصولات ساخته شده در ادامه آن فاز را افزایش دهد. ضمناً اهمیت تست در این متدولوژی خیلی جدی است و آن هم می‌تواند در حفظ کیفیت موثر باشد. همچنین UML پشتیبانی خوبی از شبه فرمالیزم در قالب OCL دارد و اگر در پروژه‌ای از آن استفاده شود می‌تواند کیفیت را افزایش دهد. ضمناً اگر چه صراحتاً توصیه نشده است ولی به دلیل چرخشی تکراری بودن متدولوژی امکان انجام مکرر بازبینی بر محصولات دارد که به شدت کیفیت را افزایش خواهد داد.

۴.۱.۱ بی‌درزی و همواری گذار

اکثر مدل‌ها به جز usecase در این متدولوژی شی گرا هستند. همچنین در تمامی مراحل محوریت ساخت مدل‌ها usecase است و یک محوریت واحد وجود دارد. این دو باعث می‌شوند وضع بی درزی خوبی در این متدولوژی حاکم باشد. ضمناً در مورد usecase تبدیل آن به Sequence Diagram از طریق رسم نمودار Activity و شی‌گرا کردن آن با اضافه کردن SwimLane هموار می‌شود و درز پوشانده می‌شود. ضمناً بیشتر مدل‌ها پس از ایجاد اولیه تا مرحله‌ی انتهایی همراه هستند و خیلی مدل جدید معرفی نمی‌شود (نمودار کلاس و نمودار Sequence نمودارهای اصلی هستند که از ابتدا تا طراحی جزئی همراه هستند. به علاوه نموداری مثل Component که جدید تولید می‌شود هم از صفر تولید نمی‌شود و تحت تاثیر نمودار package است و ناهمواری جدی ایجاد نمی‌کند.) و در کنار چرخشی‌تکراری بودن متدولوژی وضع همواری را بسیار خوب می‌کند. همچنین استخراج ویژگی‌های مهم قلمرو جواب و اضافه کردن جزئیات مربوط به قلمرو جواب فراموش نشده است و ازین نظر هم درزی ایجاد نشده است.

۵.۱.۱ میزان توجه به نیازمندی‌های وظیفه‌ای و غیروظیفه‌ای

این متدولوژی نیازمندی‌ها را در اولین زمان ممکن یعنی Inception شروع به ثبت کردن و استخراج می‌کند و در آنجا حدود ۸۰ درصد نیازمندی‌های مهم از نظر مشتری (که البته در کل حدود ۲۰ درصد نیازمندی‌ها هستند) استخراج می‌شود و در ادامه تا انتهای فاز Construction ادامه می‌یابد. ضمناً مدل خالص نیازمندی‌ها در این متدولوژی پیش‌بینی شده است و نیازمندی‌ها در قالب usecase هستند که از نظر مشتری هستند و اتمیک هستند و در ادامه

باید توسط SequenceDiagram ها تحقق یابند. با توجه به اینکه تا قبل از فاز Transition امکان انجام فعالیت نیازمندی وجود دارد بنابر این تغییر نیازمندی‌ها هم در این متدولوژی امکان پذیر است و نکته‌ی مثبتی است. در نهایت این متدولوژی نیازمندی‌رانه [۱] هم هست زیرا تمام محصولات در تمام مراحل (حتی پیاده‌سازی) بر مبنای usecase های استخراج شده ساخته می‌شوند. ضمناً نیازمندی‌های غیروظیفه‌ای هم در این متدولوژی مهم هستند و ثبت می‌شوند زیرا از عوامل اصلی شکل دهنده‌ی معماری هستند که در این متدولوژی بسیار جدی گرفته می‌شود.

۶.۱.۱ قابلیت تست، میزان ملموس‌بودن و قابلیت رهگیری به نیازمندی‌ها

قابلیت تست

قابلیت تست محصولات به دلیل تعدد محصولات مورد ضربه واقع می‌شود. البته با توجه به مشخص بودن وابستگی محصولات و انجام Continous Verification مکرر به دلیل محوریت usecase این مشکل تا حدی کنترل می‌شود. یک نکته‌ای که ممکن است تست پذیری را آسیب بزند تکرار برخی جنبه‌های سیستم در نمودار ها است. مثلا ActivityDiagram پس از usecase کشیده می‌شود تا از روی آن SequenceDiagram کشیده شود و این کار با هدف هموار کردن تبدیل است ولی به هر حال بخشی از سیستم در این دو نمودار تکرار شده که به تست‌پذیری آسیب می‌زند. یک نکته دیگر که به تست‌پذیری کمک می‌کند ساده بودن خود مدل‌هاست و با مکانیزم‌های مدیریت پیچیدگی در UML که در بخش مدل‌سازی توضیح داده‌ایم هیچکدام از مدل‌ها شلوغ نمی‌شود و برای تمام مدل‌های ساختاری و رفتاری و وظیفه‌ای امکان ایجاد سلسله مراتب هست تا از پیچیدگی جلوگیری شود و این ساده شدن به آسان‌تر شدن تست هم کمک می‌کند.

میزان ملموس‌بودن

از نظر ملموس بودن محصولات محصولات قلمرو مسئله سعی می‌شود برای کاربر قابل فهم باشد به خصوص use-case که از دید کاربر نوشته می‌شود. محصولات دیگر که در مراحل ابتدای تولید می‌شود مثل سند چشم‌انداز و سند توجیه اقتصادی هم کاملا برای مشتری ملموس است و در کنارش یک واژه‌نامه تهیه میشود که سند خیلی خیلی مهمی است و زبان مشترک ما و مشتری و کاربران تا انتهای پروژه می‌شود. نمودارهای کلاس و SequenceDiagram و در کنارش PackageDiagram هم چون جزئیات و جنبه‌های پیاده‌سازی کامپیوتری ندارد برای مشتری قابل فهم است و در صورت لزوم می‌توان از آن‌ها برای ارتباط موثر با مشتری استفاده کرد. محصولات برای پیاده‌سازان هم ملموس است زیرا در مرحله‌ی طراحی تمام کلاس‌ها و جزئیات لازم برای پیاده‌سازی در قلمرو جواب استخراج می‌شود و نمودار دور ریختنی نداریم که در تولید کد به هیچ استفاده‌ای نیاید. (PackageDiagram مستقیم در تولید کد استفاده نمی‌شود ولی تاثیر جدی بر ساخت ComponentDiagram که مستقیماً به محیط پیاده‌سازی برمیگردد دارد.).

[۱] Requirements Driven

قابلیت رهگیری به نیازمندی‌ها

رهگیری به نیازمندی‌ها در سطح مطلوب است و به دلیل نیازمندی‌رانه بودن در قالب تحقق usecase ها محقق می‌شود.

۷.۱.۱ میزان مشارکت فعال کاربر

مشارکت کاربر در این متدولوژی تا حد خوبی هست. جدی‌ترین آن در قالب استخراج نیازمندی‌هاست که تا انتهای فاز Construction انجام می‌شود. همچنین تکنیک‌های فیدبک گرفتن از کاربر در قالب Proof of Concept Prototyping استفاده شده است که مشارکت فعال محسوب می‌شود. همچنین امکان مشارکت کاربر در جلسات برنامه‌ریزی و بازبینی هست که البته به اندازه‌ی متدولوژی‌های اجایل قوت ندارد ولی وجود دارد. همچنین اینکه دوبار قرار داد بسته می‌شود هم نمودی از مشارکت فعال کاربر است زیرا در هنگام قرارداد دوم سختی‌ها وریسک‌ها و حجم سیستم و دلایل اینکه هزینه پروژه این میزان است توجیه می‌شود.

۸.۱.۱ قابلیت اجرا و سادگی و موثر بودن اجرا

قابلیت اجرای این متدولوژی زیر سوال است زیرا فرآیند پیچیده است و از چهار فاز که هر کدام تعداد زیادی محصول به عنوان milestone دارند تشکیل شده است. خود هر فاز هم همان ۵ discipline را دارد که محصولات در ضمن آن تهیه می‌شوند و در کل رویکرد IterativeIncremental هنوز هم برای برخی قابل درک نیست و اجرایش دشوار است. ضمناً ابزاری که برای RUP هست برای این متدولوژی نیست و customize کردنش هم بسیار دشوار است. بنابر این قابلیت اجرای این متدولوژی برای پروژه‌ی کوچکی که مثلاً می‌خواهد دوماهه تمام شود بی معناست و قابل اعمال نیست. البته در پروژه‌های خیلی بزرگ یک یا چند ساله با وجود پیچیدگی قابل اعمال است زیرا فعالیت‌های مدیریتی و معماری و ریسک در آن بسیار جدی است.

از نظر سادگی اجرا مجدداً همان پیچیدگی و غیرقابل Customizable بودن آسان آن سادگی اجرا لطمه می‌خورد. در کنار آن نکات مثبتی هست که می‌تواند باعث سادگی اجرا شود. اولاً usecase محور بودن و همچنین معماری محور بودن باعث ایجاد تمرکز می‌شود که اجرا را موثر می‌کند. مدیریت پروژه در آن بسیار جدی است که باعث موثر شدن اجرا می‌شود. به تکنیک‌های خطا خیز هم تکیه نکرده است که موثر بودن اجرایش را لطمه بزند. به دلیل سنگین بودن مدل‌ها و پیچیدگی فرآیند برای اجرای آن حتماً به ابزار نیاز داریم که در تمرین قبل در همین مورد برای متدولوژی‌های مختلف گفتیم که وابستگی به ابزار هم خوب است و هم بد. خوبی‌اش این است که بسیاری از کارها به شکل automated و روش‌مند قابل انجام می‌شوند ولی به هر حال تهیه ابزار به زمان و هزینه نیاز دارد و ازین جهت منفی است.

۹.۱.۱ مدیریت پیچیدگی فرآیند

فرآیند متدولوژی به ۴ فاز شکسته شده است و در داخل هر فاز جداگانه آن ۵ discipline اجرا می‌شوند که می‌توان نوعی layering محسوبش کرد. همچنین همینکه ۴ فاز جدا داریم و ۵ discipline جدا خودش نوعی Partition-ing محسوب می‌شود و باعث شده کارها ریزدانه‌تر و ترتیب آن‌ها قابل درک‌تر باشد. ضمناً در هر فاز محصولات تولیدی و کیفیت آن‌ها (در واقع Definition of done آن‌ها) جداگانه تعریف می‌شود که باعث فهم راحت‌تر خروجی فازها می‌شود. همچنین تمرکز usecase و معماری در تمام طول پروژه می‌تواند به افراد کمک کند در عین بزرگ بودن فرآیند پیچیدگی آن را کمتر حس کنند.

۱۰.۱.۱ توسعه‌پذیری ، مقیاس‌پذیری، قابلیت پیکربندی و انعطاف‌پذیری

در این متدولوژی چیزی تحت عنوان توسعه و نقاطی که در آن بتوان متدولوژی را توسعه داد و به آن اضافه کرد وجود ندارد پس توسعه‌پذیری اصلاً نداریم.

از نظر مقیاس‌پذیری بحرانی، می‌توان با اطمینان گفت این متدولوژی پروژه‌های تا سطح Essential Money یعنی سطح ۳ را پشتیبانی می‌کند. زیرا در قالب OCL امکان اعمال شبه‌فرمالیزم هست و فعالیت‌های سنجش و مدیریت ریسک از ابتدا تا انتها خیلی جدی مطرح هست و تست هم خیلی جدی گرفته می‌شود و به کیفیت هم توجه ویژه است و تا پایان Elaboration خصیصه‌های کیفی برای محصولات تعریف می‌شود. در مورد سیستم‌های Life Critical ولی اعمال متدولوژی زیر سوال است زیرا فرمالیزم صد در صد ریاضی در UML نداریم و بهتر است برای این سیستم‌ها از روش‌های کاملاً فرمال استفاده کرد. از نظر مقیاس‌پذیری سایز پروژه و تعداد افراد درگیر ولی وضعیت خوبی داریم زیرا فعالیت مدیریت پروژه و زمان‌بندی توجه شده است علاوه بر آن مدل‌سازی به خصوص مدل‌سازی معماری در آن جدی گرفته می‌شود که در تیم‌های بزرگ می‌تواند وسیله همگام شدن و ارتباط گرفتن افراد در تیم‌های مختلف شود.

امکان پیکربندی متدولوژی در ابتدای پروژه وجود دارد ولی ابزاری برای آن ارائه نشده است که با توجه به بزرگ بودن متدولوژی عملاً اجرایش بسیار سخت و زمان‌بر است و فقط برای پروژه‌ی خیلی طولانی (مثلا در حد چندسال) مطرح است وگرنه نمی‌ارزد دوماه صرف پیکربندی‌اش کنیم.

با وجود تکراری‌چرخشی بودن متدولوژی که فرصت خوبی برای اصلاح و تغییر فرآیند در حین اجرا ایجاد کرده است متدولوژی پیش‌بینی برای این کار نکرده است و فعالیتی برای تغییر فرآیند در حین اجرا ندارد.

۱۱.۱.۱ حیطه‌ی کاربرد

حوزه‌ی کاربرد خاصی برای متدولوژی مطرح نشده است و به عنوان متدولوژی عمومی شی‌گرا (در واقع هدف هم این بود که متدولوژی باشد برای تمام دوران‌ها! که موفق نشدند) مطرح شده است. ولی در عمل برای پروژه‌های که تعداد نفرات بالا است و ریسک نسبی وجود دارد خوب است چون مدل‌سازی و فعالیت‌های مدیریت پروژه در آن جدی است ولی برای پروژه کوچک کوتاه مدت بدون ریسک این فعالیت‌ها فقط سربار اضافی هستند و سرعت را

کم می‌کنند ولی در پروژه بزرگ برای کنترل پیشرفت و کیفیت و هماهنگی افراد لازم هستند. البته این سنگین بودن متدولوژی بیشتر برای افزایش تعداد افراد مفید است و در واقع اگر پروژه کوچک باشد ولی بحرانی هم باشد همچنان استفاده از متدولوژی معقول نیست.

۲.۱ بخش مدل‌سازی

۱.۲.۱ پشتیبانی از مدل‌سازی سازگار، دقیق و بدون ابهام شی‌گرا

متدولوژی بر مبنای مدل‌سازی UML مطرح می‌شود و همه مدل‌های بصری آن با این زبان توصیف می‌شوند. البته سند متنی هم دارد مثل سند Business Case و یا Vision و Project Glossary و اسناد مرتبط با سنجش و مدیریت ریسک.

در این متدولوژی هر سه نوع مدل‌سازی دیده می‌شود. مدل‌سازی وظیفه‌ای در قالب usecase انجام می‌شود. مدل‌سازی ساختاری هم بسیار متنوع هست. نمودار ClassDiagram و همچنین ObjectDiagram و Package-Diagram و ComponentDiagram و DeploymentDiagram همگی ساختاری هستند. مدل‌سازی رفتاری هم در قالب SequenceDiagram و CommunicationDiagram و InteractionOverviewDiagram و TimingDiagram و همچنین StateTransitionDiagram قابل انجام است.

گذار مدل‌ها از سطح منطقی به فیزیکی و پیاده‌سازی هم به خوبی دیده می‌شود. نمودار Class که ساختاری است در ابتدا کلاس‌های قلمرو مسئله است و در ادامه با تدقیق همان کلاس‌های قلمرو جواب اضافه می‌شود که فیزیکی می‌شود و همین حرف در مورد ObjectDiagram هم می‌تواند باشد. نمودار ساختاری PackageDiagram در قلمرو منطقی مطرح می‌شود و در حالت فیزیکی به ComponentDiagram تبدیل می‌شود. نمودارهای رفتاری Sequence و Communication هم برای کلاس‌های قلمرو مسئله و در ادامه قلمرو جواب کشیده می‌شوند پس مدل‌سازی رفتاری هم از منطقی به فیزیکی به خوبی صورت می‌گیرد. البته مدل‌سازی وظیفه‌ای فقط همان usecase است که همیشه در سطح منطقی می‌ماند و مدل‌سازی فیزیکی وظیفه‌ای نداریم.

از نظر سطوح مختلف درشت دانگی در سطح سازمان مدلی دیده نمی‌شود. مدل‌سازی رفتاری سطح سیستم را می‌توان در قالب StateTransitionDiagram در صورت لزوم انجام داد. مدل‌سازی وظیفه‌ای سیستم خیلی جدی دیده نمی‌شود ولی از آنجا که در usecase مرز سیستم مشخص است می‌توان آن را نمودار سیستمی دانست. مدل‌سازی ساختاری سیستم هم در متدولوژی هست اگر نمودار کلاس اولیه را که شامل کلاس‌های خارج سیستمی هم هست و باید حذف شود در نظر بگیریم می‌توان تا حدی سیستم حسابش کرد و مهم‌تر از آن نمودار Deploy-mentDiagram می‌تواند در حد کل سیستم باشد (البته می‌تواند در حد زیرسیستم‌ها هم باشد پس در بخش بعد به آن مجدد اشاره نمی‌کنیم). مدل‌سازی وظیفه‌ای زیرسیستم‌ها هم در قالب همان نمودار usecase هست زیرا در نمودار usecase امکان قرار دادن موارد کاربرد در زیرسیستم‌ها وجود دارد. برای رفتار یک تک سیستم می‌توان در صورت نیاز StateTransitionDiagram کشید و رفتار بین سیستم‌ها هم با SequenceDiagram های تو در

تو در قالب InteractionOverviewDiagram قالب مدل‌سازی است. مدل سازی ساختاری زیرسیستم‌ها هم در قالب PackageDiagram و ComponentDiagram محقق شده است. مدل‌سازی وظیفه‌ای در سطح بین اشیا و داخل اشیا در حد همان مشخص کردن متدها در نمودار کلاس است و مدل دیگری نمی‌بینیم. مدل‌سازی رفتاری بین اشیا در قالب SequenceDiagram و یا CommunicationDiagram محقق می‌شود. مدل‌سازی ساختاری بین شی هم در قالب ClassDiagram و ObjectDiagram انجام می‌شود. مدل‌سازی ساختار درون اشیا هم در قالب همان بخش Attribute های کلاس‌ها در نمودار ClassDiagram قابل انجام است. مدل‌سازی رفتاری هم می‌تواند در قالب StateTransitionDiagram باشد که رفتار وابسته به حالت یک تک کلاس را مدل کند و هم می‌توان بدنه‌ی متدها را با ActivityDiagram نشان داد. مدل‌سازی وظیفه‌ای خاصی در سطح داخل کلاس دیده نمی‌شود.

پشتیبانی متدولوژی از فرمالیزم با اضافه کردن OCL که جزئی از UML است ممکن شده است. ضمناً یک نکته جالب دیگر نمودار InteractionOverviewDiagram است که با آن ترتیب مجاز اجرای usecase ها را می‌توان نشان داد و چیزی شبیه ترتیب اجرای عملیات‌ها در Fusion است و نوعی فرمالیزم است و دقت را افزایش می‌دهد.

۲.۲.۱ وجود راهکارهای رفع ناسازگاری و مدیریت پیچیدگی مدل‌ها

تعداد مدل‌های این متدولوژی زیاد است و حفظ سازگاری بین آن‌ها تلاش زیادی می‌خواهد. در زبان مدل‌سازی صراحتاً رابطه‌ی مدل‌ها مشخص نیست (بر خلاف آنچه مثلا در Fusion دیدیم) و کمکی به رفع ناسازگاری وجود ندارد. نکات مثبتی که به حفظ سازگاری کمک می‌کنند یکی تکراری چرخشی بودن متدولوژی است که می‌تواند کمک کند تغییرات برخی نمودارها که در تکرار قبل یادمان رفته انتشار دهیم در این تکرار جدید اعمال کنیم. یک نکته‌ی مثبت دیگر محوریت usecase است و تاحدی امکان چک مکرر مدل‌ها را فراهم می‌کند اما به هرحال همچنان ممکن است یک SequenceDiagram یک usecase را محقق کند ولی افراد یادشان رفته باشد تعدادی از متدهای اضافه شده در SequenceDiagram را در ClassDiagram اضافه کنند. یک نکته‌ی مثبت دیگر وجود مرحله‌ی تست در discipline ۵ است و اگر در مرحله‌ی تست تمام مدل‌ها (و نه فقط SequenceDiagram) مورد سنجش قرار گیرند امکان رفع ناسازگاری‌ها هست ولی به هرحال این تست هم هزینه دارد و در کل خود زبان مدل‌سازی جلوی عدم ناسازگاری را نگرفته است و مجبوریم با هزینه در فرآیند این ناسازگاری را جبران کنیم.

اینکه هرجنبه‌ی سیستم نمودار جدا دارد خود باعث رفع پیچیدگی است و می‌توان جنبه‌های مختلف را در مدل‌های مختلف دید. همچنین با نمودارهای ComponentDiagram و PackageDiagram و Interac-tionOverviewDiagram و هم‌چنین جدا کردن usecase ها در قالب زیرسیستم‌ها امکان دسته‌بندی اجزای مدل‌ها که نوعی Layering است فراهم شده است تا قابلیت فهم مدل‌های flat زیاد شود و بتوان اجزای آن‌ها را در یک ساختار درشت‌دانه دسته‌بندی کرد.

فصل ۲

مقایسه‌ی متدولوژی‌های USDP و EUP

۱.۲ بخش فرآیند

۱.۱.۲ تعریف متدولوژی

متدولوژی از RUP ارث می‌برد و فرآیندمحور و چرخشی‌تکراری تعریف شده است و دو فاز و تعدادی discipline برای فعالیت‌های مربوط به پشتیبانی و بالانگه‌داشتن نرم‌افزار و فعالیت‌های سازمانی اضافه شده‌اند و این فعالیت‌های اضافه شده تشریح شده‌اند. البته همچنان مانند USDP چون پوشش از چرخه‌ی نرم‌افزار کامل نیست و توضیح نداده چگونه متدولوژی را توسعه دهیم تا پوشش کامل شود نمی‌توان تعریف فرآیند را کامل دانست ولی بخش‌هایی که تعریف شده‌اند دقیق و سازگار هستند.

تعریف متدولوژی به شکل فرآیند محور است و در کنارش نقش‌های درگیر و محصولات تولید شده هم در حد قابل قبولی تعریف شده‌اند.

زبان اصلی مدل‌سازی که از RUP به ارث برده شده است UML است که تعریف دقیق دارد ولی متدولوژی خود را به UML محدود نکرده است و برای مدل‌سازی سازمانی DFD را پیش‌نهاد کرده است که پیش‌نهاد خوبی است و مشکلی ندارد ولی برای مدل کردن نیازمندی‌ها اجبار Usecase را ندارد و هر سندی می‌تواند استفاده شود حتی سند متنی که این یک ضعف است و این بخش از مدل‌سازی دقیق تعریف نشده است. در کنار این زبان مدل‌سازی محصولاتی که باید در فازها و فعالیت‌ها ایجاد و یا تکمیل شوند توضیح داده شده است.

تکنیک‌ها و قواعد هم اگر استفاده شده‌اند در تعریف متدولوژی گنجانده شده‌اند.

فعالیت‌های چتری هم در تعریف متدولوژی هست و هم برای آن‌ها Milestone وجود دارد (که مثلا ریسک تا پایان Elaboration از بین رفته باشد) و هم Discipline های مختص به آن برای انجام مکرر تعریف شده است.

به جز نقصی که در پوشش چرخه‌ی نرم‌افزار هست تعریف متدولوژی دقیق و سازگار و کامل است.

شباهت‌ها هر دو متدولوژی فرایند محور تعریف شده‌اند و نقش‌های درگیر در هر فرایند و محصولات خروجی فرآیندها را به خوبی مشخص کرده‌اند. هر دو متدولوژی عمده‌ی مدل‌سازی را با UML انجام می‌دهند.

پوشش خوبی از فعالیت‌های چتری در هر دو متدولوژی هست و تکنیک‌های مورد استفاده هم به دقت کافی در هر دو تعریف شده‌اند.

تفاوت‌ها فرآیند تعریف شده در EUP از نظر درشت دانگی بخش بیشتری از چرخه‌ی ایجاد نرم‌افزار را پوشش می‌دهد ولی در ازایش بخش‌هایی که مشترک هستند در USDP با دقت بیشتری توصیف شده‌اند. در EUP زبان مدل‌سازی و محصولات محدود به زبان UML نیستند که نکته‌ی خوبی است ولی در مورد نیازمندی اشتباه صورت گرفته است که اجازه‌ی هرسندی را داده است و در زمینه مستند کردن نیازمندی USDP بهتراست ولی در بقیه محصولات هر دو متدولوژی قابل قبول هستند و با دقت کافی محصولات تعریف شده است. فعالیت‌های چتری در هر دو متدولوژی هستند ولی در EUP به دلیل وجود ۳ Discipline مجزای مربوط به آن‌ها می‌توان گفت جدی‌تر گرفته شده‌اند. یک نکته‌ی مثبت دیگر در مورد تعریف محصولات USDP وجود چک لیست درشت دانه برای پایان هرفاز است که در RUP وجود ندارد.

سایر نکات ـ

۲.۱.۲ پوشش چرخه‌ی عمومی تولید نرم‌افزار

هر آنچه در مورد USDP گفتیم در EUP هم پوشش داده می‌شود. البته چون EUP بر مبنای RUP است برخی جزئیات آن کمتر است و کلی‌گو تر است. ضمناً فعالیت تحلیل در RUP و در نتیجه EUP اختیاری است و خب بخشی از مدل‌سازی فضای مسئله ناقص می‌شود اما نمی‌توان گفت مدل‌سازی قلمرو مسئله به کلی انجام نمی‌شود چون هنوز مدل‌های سازمانی و همچنین مدل‌های قلمرو مسئله که در فعالیت BusinessModeling استخراج می‌شوند وجود دارند. علاوه بر آن این متدولوژی به بخش‌هایی از فاز Maintenance هم توجه دارد. در فاز Production در اصل به Operation & Support و بالا نگه داشتن سیستم می‌پردازد و فاز Retirement را هم دارد که به مرگ نرم‌افزار و جایگزینی آن با سیستم جدید می پردازد. هم چنین یک discipline برای Operation & Support اضافه کرده است. اما همچنان بخش اصلی Maintenance به معنی اضافه کردن قابلیت به نرم‌افزار، فیکس باگ و . را ندارد و پوشش کامل کامل نیست.

شباهت‌ها هر دو متدولوژی به مدل‌سازی قلمرو مسئله و استخراج نیازمندی‌ها و تحلیل امکان سنجی و طراحی معماری و طراحی جزئی و پیاده‌سازی و تست می‌پردازند.

تفاوت‌ها در EUP که از RUP ارث می‌برد فعالیت چتری مخصوص استقرار هست پس می‌توان گفت استقرار در EUP بهتر پوشش داده شده است. همچنین در EUP به فعالیت‌های بالانگه داشتن نرم‌افزار و پشتیبانی از آن و همچنین مرگ پروژه توجه شده است که در USDP وجود ندارد. در USDP فعالیت تحلیل اجباری است و در EUP اختیاری پس برخی مدل‌سازی‌های قلمرو مسئله (همانطور که گفتیم

مدل‌سازی قلمرو مسئله در EUP صفر نیست) مثل تولید packagediagram در USDP اجباری هستند ولی در EUP اختیاری هستند.

سایر نکات به صورت کلی بخواهیم بگوییم EUP پوشش بهتری دارد زیرا بخش‌هایی از فاز نگهداری را در خود جای داده که USDP ندارد اما USDP جزئیات بیشتری دارد و دقیق‌تر تعریف شده است.

۳.۱.۲ پشتیبانی از فعالیت‌های چتری

مدیریت ریسک

مواردی که در مورد USDP گفتیم اینجا هم صادق است و به کاهش ریسک کمک می‌کند و در این بخش فقط تفاوت‌ها را ذکر می‌کنیم. علاوه بر آنچه در USDP دیدیم فعالیت Operation and Support این متدولوژی می‌تواند به کاهش ریسک کمک کند زیرا می‌تواند باعث شود برخی مشکلات مربوط به اجرا در محیط کاربر زودتر معلوم شود. همچنین در فاز Retirement ریسک از کار افتادن سیستم‌های دیگر با تغییر و جایگزینی نرم‌افزار مرده با استراتژی‌های گفته شده کاهش می‌یابد. اینکه فعالیت تحلیل در این متدولوژی اختیاری است ممکن است کمی ریسک را افزایش دهد چون برخی مدل‌های قلمرو مسئله تولید نمی‌شوند و تولید ناگهانی مدل طراحی آن‌ها می‌تواند شامل ریسک شود. همچنین اینکه سند نیازمندی متنی هم در آن قابل استفاده است ریسک اشتباه فهمیده شدن نیازمندی‌ها را دارد.

شباهت‌ها هر دو متدولوژی از تحلیل امکان‌سنجی اولیه برای جلوگیری از ریسک در آغاز استفاده می‌کنند. هر دو متدولوژی از برنامه‌ریزی مبتنی بر ریسک استفاده می‌کنند که به کنترل ریسک کمک می‌کند. هر دو متدولوژی از پروتوتایپ برای کاهش ریسک استفاده می‌کنند. هر دو متدولوژی چرخشی‌تکراری هستند و حدی از CI را دارند که به کنترل ریسک کمک می‌کند. هر دو متدولوژی فعالیت تست و روش‌های فرمال را دارند که به مدیریت ریسک کمک می‌کند.

تفاوت‌ها در EUP به دلیل توجه جدی‌تر به پشتیبانی و بالانگه‌داشتن نرم‌افزار ریسک‌های مربوط به این دو فعالیت بهتر مدیریت می‌شوند. همچنین ریسک جانشینی سیستم بعد از مرگش با سیستم بعدی از EUP بهتر مدیریت شده. از طرف دیگر به فعالیت تحلیل اختیاری نگاه کرده و اجازه‌ی وجود سند متنی برای نیازمندی‌ها را داده است که مطابق آنچه در بالا در توضیحش گفتیم می‌تواند ریسک ایجاد کند.

سایر نکات ـ

مدیریت پروژه

این متدولوژی خوبی‌های USDP را دارد و علاوه بر آن چون بر اساس RUP است صراحتاً یک discipline برای مدیریت پروژه دارد. همچنین در مجموعه ۷ فعالیت سازمانی خود یک فعالیت مخصوص برای مدیریت ارتباطات افراد با سازمان دارد که نکته‌ی مثبتی است.

شباهت‌ها هر دو متدولوژی سه فعالیت plan و schedule و monitor را در خود دارند.

تفاوت‌ها هر دو متدولوژی فعالیت مدیریت پروژه دارند ولی می‌توان گفت در کل EUP اهمیت جدی‌تری دارد. زیرا اولا discipline جداگانه برای آن در نظر گرفته است. دوماً دید سازمانی دارد و در دید سازمانی روابط افراد و تیم‌ها را در نظر دارد که برای مدیریت پروژه بسیار مهم هستند. یک تفاوت دیگر هم در نحوه نگرش USDP و EUP که از RUP ارث برده به مدیریت پروژه هست که در USDP تاکید شده در انتهای Elaboration باید پلن دقیق چند تکرار اول Construction دقیق باشد ولی در RUP صرفا میگوید یک پلن درشت دانه برای کل فاز در بیاور که رویکرد USDP منطقی‌تر است.

سایر نکات متدولوژی EUP برتری کامل دارد.

تضمین کیفیت

این متدولوژی تمام نکات مثبت USDP را دارد البته در مورد usecase چون در این متدولوژی اختیاری شده و می‌توان از آن استفاده نکرد بنابر این استفاده نکردن از آن می‌تواند رهگیری مستقیم به نیازمندی‌ها را سخت کند و ان نکته‌ی مثبت از بین می‌رود. اما فعالیت‌های سازمانی این متدولوژی می‌تواند بر بهبود کیفیت تاثیر بگذارد. به عنوان مثال در فعالیت Strategic Reuse می‌توان از ماژول‌های با کیفیت نرم‌افزاری که در سایر پروژه‌های سازمان استفاده و تست شده‌اند استفاده کرد. همچنین در Enterprise Architecture علاوه بر معماری نرم‌افزار سازمان استانداردها و رهنمودهای کیفی سازمان هم تبیین می‌شوند که باید در تمام پروژه‌ها رعایت شوند و کیفیت را افزایش می‌دهد.

شباهت‌ها وجود تست می‌تواند در تضمین کیفیت موثر باشد. وجود فرمالیزمی که در UML هست به کیفیت کمک می‌کند. با چرخشی‌تکراری بودن امکان کنترل کیفیت و بازبینی مکرر محصولات ایجاد شده است.

تفاوت‌ها در USDP یک چک لیست برای محصولات هر فاز و خصصیه‌های کیفی لازم برای آن‌ها هست که در EUP نیست و می‌تواند به تضمین کیفیت محصولات کمک کند. همچنین در EUP استفاده اختیاری از usecase نکته‌ای منفی است چون رهگیری به نیازمندی‌ها را سخت می‌کند که باعث کاهش کیفیت می‌شود. نکته‌ی مثبت که در EUP هست ولی در USDP نه عدم توجه به سازمان است و استفاده از زیرساخت‌های سازمان در قالب معماری سازمانی و یا ماژول‌های نرم‌افزاری سازمان می‌تواند کیفیت را افزایش دهد.

سایر نکات ـ

۴.۱.۲ بی‌درزی و همواری گذار

این متدولوژی نسبت به USDP ول‌انگارانه‌تر مدل سازی می‌کند و می‌تواند بی‌درزی‌اش را آسیب بزند. زیرا هیچ اجباری برای استفاده از usecase ندارد و آن محوریت را از دست داده. توصیه به استفاده از DFD برای مدل‌سازی سازمانی هم به بی درزی آسیب می‌زند و ارتباط بین DFD و مدل‌های شی‌گرا مشخص نیست. البته همچنان Iterative Incremental بودن می‌تواند این درز را بپوشاند و از تبدیل این درز به ناهمواری جلوگیری کند.

شباهت‌ها استفاده از UML باعث بی‌درزی می‌شود. تکمیل مدل‌های ساختاری و رفتاری از قلمرو مسئله به جواب در هر دو از ایجاد درز جلوگیری می‌کند. چرخشی‌تکراری بودن متدولوژی‌ها به همواری گذار در آن‌ها کمک می‌کند.

تفاوت‌ها عدم استفاده از usecase آن مفهوم مشترکی که به واسطه محوریت usecase ایجاد شده و باعث بی‌درزی می‌شود در EUP باعث می‌شود وضع بی‌درزی در آن بدتر باشد. همچنین در EUP می‌توان از DFD استفاده کرد که درست است برای مدل‌سازی وظیفه‌ای سازمانی خیلی مناسب است ولی به هرحال شی‌گرا نیست و در نقطه‌ای مجبور هستیم دورش بیندازیم و ادامه پیدا نمی‌کند که باعث درز است. البته یک نکته در EUP هست و در آن زمینه خاص از USDP هموارتر است. آن هم هنگام مرگ پروژه است و USDP نسخه‌ای برایش نپیچیده ولی در EUP فاز Retirement را داریم و امکان گذر هموار به ساخت سیستم جدید پس از مرگ سیستم فعلی را فراهم می‌کند و این در USDP وجود ندارد. با تعریف فعالیت‌های Operation & Support می‌توان گفت گذار به فاز Transition هم در EUP هموارتر از USDP انجام می‌شود.

سایر نکات ـ

۵.۱.۲ میزان توجه به نیازمندی‌های وظیفه‌ای و غیروظیفه‌ای

وضعیت تا حد زیادی شبیه به USDP است ولی توصیه‌ی جدی به استفاده از usecase ندارد و usecase محوریتش را از دست داده و لزوماً نیازمندی‌رانه نیست و صرفاً بر مبنای نیازمندی هاست.

شباهت‌ها هر دو متدولوژی به استخراج و مدل‌سازی نیازمندی‌های وظیفه‌ای و غیروظیفه‌ای می‌پردازند و بر اساس آن‌ها سایر مدل‌ها را می‌سازند. امکان تغییر وتکمیل تدریجی نیازمندی‌ها در هر دو متدولوژی دیده شده است.

تفاوت‌ها در EUP چون ممکن است سند متنی برای نیازمندی‌ها استفاده شود دیگر Usecase Driven نیست و در نتیجه نیازمندی‌رانه نیست و می‌توان صرفاً گفت مبتنی بر نیازمندی هاست.

سایر نکات در این معیار USDP بهتر عمل کرده است.

۶.۱.۲ قابلیت تست، میزان ملموس‌بودن و قابلیت رهگیری به نیازمندی‌ها

قابلیت تست

این متدولوژی از نظر تست نکات خوبی که برای USDP را گفتیم را دارد ولی بعضی جنبه‌هایش بدتر است. اولا مدل‌ها بیشتر شده که باز هم تست‌پذیری را سخت تر می‌کند. ضمناً محوریت usecase که باعث verification مکرر محصولات می‌شد از بین رفته است.

شباهت‌ها هر دو متدولوژی به دلیل تعدد مدل‌ها دچار مشکل در تست‌پذیری مدل‌ها هستند. هر دو متدولوژی مکانیزم‌های خوب مدیریت پیچیدگی مدل‌ها هستند و با وجود تعدد مدل خود مدل‌ها ساده می‌شوند که تست‌پذیریشان را افزایش می‌دهد.

تفاوت‌ها نیازمندی‌رانه بودن USDP که در EUP الزامی نیست محوریت مشترک ایجاد می‌کند که کمی فهم مدل‌ها و تست‌پذیری آن‌ها را آسان‌تر می‌کند.

سایر نکات ـ

میزان ملموس‌بودن

از نظر ملموس بودن برای کاربر وضع مشابه USDP است و نمودار DFD سازمانی هم برای افراد قلمرو مسئله مفید و قابل فهم است. در عین حال معرفی کردن DFD ممکن است برای برنامه‌نویسان خوشایند نباشد و مدل مفیدی در پیاده‌سازی شی‌گرای سیستم نیست و در عمل نموداری است که دور انداخته می‌شود و در تولید کد نقش چندانی ندارد پس برای برنامه نویسان مدل مطلوبی نیست.

شباهت‌ها نمودارهای قلمرو مسئله برای مشتری و ذی‌نفعان درگیر در مورد قلمرو مسئله ساده و قابل فهم هستند. بخش‌های UML هر دو متدولوژی برای برنامه‌نویسان ملموس و مفید هستند.

تفاوت‌ها در EUP از DFD استفاده شده است که برای مدل‌سازی سازمانی مفیدتر است و به راحتی برای مشتری قابل فهم است که نکته‌ی مثبتی است. از طرف دیگر این DFD به وضوح کافی در فرآیند برای تولید کد مفید نیست و دور انداخته می‌شود پس از نظر برنامه نویسان نمی‌تواند ملموس باشد.

سایر نکات ـ

قابلیت رهگیری به نیازمندی‌ها

در این متدولوژی محوریت usecase از بین رفته و الزامی به Usecase Driven بودن نیست و از هر چیزی حتی سند متنی می‌توان توصیف نیازمندی را تشکیل داد. این کار باعث می‌شود رهگیری به نیازمندی در ذات متدولوژی نباشد و مجبور به استفاده از تکنیک‌های مثل ماتریس رهگیری شویم.

شباهت‌ها ندارند.

تفاوت‌ها در USDP رهگیری به نیازمندی‌ها به دلیل ساخت تمام مدل‌ها در تمام مراحل بر مبنای تحقق usecase انجام می‌شود ولی اگر در EUP مثلا سند متنی برای نیازمندی استفاده شود دیگر معلوم نیست کدام بخش از کدام مدل کدام خط یا جمله از سند نیازمندی را محقق کرده (مگر اینکه این تحقق دستی جایی ثبت شود.). پس USDP وضع بسیار بهتری دارد.

سایر نکات ـ

۷.۱.۲ میزان مشارکت فعال کاربر

این متدولوژی هم مشابه USDP است. در کنار آن در نظر گرفتن یک discipline جداگانه برای Ops و Support می‌تواند نکته‌ی مثبت اضافی باشد زیرا در این فعالیت می‌توان از کاربر در مورد سیستم در حال اجرا و یا اسناد آموزش بازخورد گرفت.

شباهت‌ها در هر دو متدولوژی در قالب استخراج نیازمندی‌ها به شکل مکرر با مشتری ارتباط داریم. همچنین از تکنیک ساخت پروتوتایپ برای بازخورد گرفتن از کاربر به شکل فعال استفاده شده است. با توجه به چرخشی تکراری بودن امکان برگزاری جلسات با ذی‌نفعان هست ولی خب به شکلی که در اجایل‌ها توصیه شده وجود ندارد.

تفاوت‌ها فعالیت مربوط به Ops & Support در EUP می‌تواند تا حدی مشارکت بیشتر کاربر را منجر شود.

سایر نکات

۸.۱.۲ قابلیت اجرا و سادگی و موثر بودن اجرا

این متدولوژی از RUP برگرفته شده و علاوه بر چهار فاز RUP دو فاز جدید به آن اضافه کرده است. علاوه بر discipline های هفت گانه‌ی RUP دو discipline جدید هم دارد که مورد مدیریت سازمانی آن خودش ۷ فعالیت جدا است! پس در کل به شدت فرآیند پیچیده است و به قابلیت اجرا لطمه می‌زند و تنها در پروژه بسیار بزرگ که ارتباط با سازمان در آن اهمیت دارد می‌تواند اجرایش منطقی باشد و در پروژه‌ی کوچک اصلاً قابل اعمال نیست. البته ابزار RMC که در اصل برای RUP است اینجا قابل اعمال است چون بر اساس RUP شکل گرفته اما متدولوژی کاملا بر مبنای RUP نیست و آن ابزار هم به طور کامل نمی‌تواند کمک کند. به دلیل پیچیدگی که وجود دارد، سادگی اجرا هم لطمه می‌خورد. اما در کنار آن نکات مثبتی هست که امکان اجرای ساده و موثر را بهبود می‌دهد. فعالیت مدیریت پروژه در آن جدی است و هم در milestone های فازها هست و هم discipline جدا دارد. مانند RUP تاکید بر معماری دارد و نوعی دیدمشترک و متمرکز ایجاد می‌کند. یک نکته منفی دیگر برای سادگی اجرا این است که استفاده از usecase در ان اجباری نیست و حتی می‌توان از سند متنی استفاده کرد که در واقع یک تکنیک

خطا خیز استفاده کرده است. همچنین برای customize کردن ابزار ویژه می‌خواهد که مشابه استدلال‌های قبل برای این معیار ابزار داشتن هم نکته‌ی منفی است هم نکته‌ی مثبت.

شباهت‌ها هر دو متدولوژی به دلیل زیاد بودن واحدهای فرآیند و تولید محصولات زیاد برای پروژه‌های کوچک قابلیت اجرا ندارند. هر دو متدولوژی به همین دلیل در سادگی اجرا هم دچار لطمه می‌شوند. هر دو متدولوژی دید متمرکز با معماری ایجاد کرده‌اند و به مدیریت پروژه اهمیت داده‌اند که اجرای موثر را خوب می‌کند. هر دو متدولوژی به دلیل تعدد محصولات برای مدل‌سازی به ابزار نیاز دارند.

تفاوت‌ها متدولوژی EUP فازها و discipline های بیشتری دارد و بنابر این قابلیت اجرای آن فقط برای پروژه‌هایی در ابعاد سازمان مطرح است ولی USDP کم‌حجم‌تر است و در مقیاس‌هایی کمتر از مقیاس EUP قابل اجراست. در USDP علاوه بر معماری محوریت و تمرکز بر usecase هم هست که می‌تواند باعث موثر شدن اجرا شود ولی در EUP این مورد از بین رفته است و USDP یک ویژگی مثبت دارد. از طرف دیگر مدیریت پروژه در EUP صراحتاً به عنوان discipline مطرح است ولی در USDP discipline ندارد پس ازین نظر EUP بهتر عمل کرده است. ضمناً توجه به فعالیت‌های سازمانی در EUP در جایی که ارتباط پروژه با سازمان مطرح باشد می‌تواند باعث شود این متدولوژی در عمل راحت‌تر از USDP اجرا شود زیرا به طور مثال در USDP ارتباط افراد با سازمان در نظر گرفته نشده است که می‌تواند مشکل زا شود. یک نکته‌ی منفی هم برای EUP هست و آن بسنده کردن به سند متنی نیازمندی است و usecase اجباری نیست و می‌دانیم سند متنی ابهام دارد پس از تکنیک خطاخیز استفاده کرده که به موثر بودن اجرا لطمه می‌زند ولی USDP این مشکل را ندارد. تفاوت آخر هم این است که دست کم برای خاص‌سازی بخش‌هایی از متدولوژی EUP با ابزار RMC ابزار هست ولی USDP چنین ابزاری ندارد.

سایر نکات در کل اگر یک کلام بخواهیم بگوییم کدام قابلیت اجرای ساده‌تری دارد USDP به دلیل ساده‌تر و کم‌حجم‌تر بودن در درصد بیشتری از پروژه‌ها به شکل موثر اجرا می‌شود ولی به هر حال در جایی که در سطح Enterprise هستیم حتی USDP هم جواب نمی‌دهد و در آن‌جا EUP بهتر اجرا می‌شود.

۹.۱.۲ مدیریت پیچیدگی فرآیند

در EUP هم مشابه USDP شش فاز درشت دانه تعریف شده و ۷ discipline تعریف شده در RUP به علاوه discipline های Ops & Support و فعالیت‌های سازمانی به آن اضافه شده است. ضمناً خود فعالیت سازمانی به ۷ discipline جدا شکسته شده است. پس با لایه‌بندی از طریق فازها و تکه‌تکه کردن فعالیت‌ها سعی شده است فهم و اجرای متدولوژی ساده‌تر شود. محوریت معماری در متدولوژی هم می‌تواند به افراد کمک کند در حین اجرای فرآیند با وجود بزرگ بودن و تعداد زیاد فعالیت‌ها تمرکز داشته باشند و این پیچیدگی کمتر حس شود.

شباهت‌ها هر دو با مشخص‌کردن فازهای درشت دانه و قرار دادن فعالیت‌های ریزدانه‌تر درون این فازها سعی کرده‌اند فهم و اجرای فرآیند متدولوژی را آسان کنند و پیچیدگی‌اش را مدیریت کنند. همچنین دید معماری در هر دو متدولوژی می‌تواند به افراد جهت بدهد و با وجود مراحل زیاد متدولوژی در هر دو متدولوژی این دید مشترک باعث پوشاندن پیچیدگی شود.

تفاوت‌ها در USDP محصولات تولیدی در انتهای هرفاز با دقت بیشتری تعریف شده و خاصیت کیفی که محصولات باید داشته باشند هم تعریف شده است که می‌تواند به فهم بهتر فازها کمک کند ولی در EUP این نکته نیست.

سایر نکات هر دو متدولوژی مدیریت پیچیدگی فرآیند دارند و خیلی نمی‌توان گفت کدام بهتر است ولی با توجه به سادگی نسبی USDP و در عین حال جزئیات بیشترش و کم‌حجم‌تر بودن می‌توان گفت مدیریت پیچیدگی فرآیندش بهتر صورت گرفته است.

۱۰.۱.۲ توسعه‌پذیری ، مقیاس‌پذیری، قابلیت پیکربندی و انعطاف‌پذیری

در EUP نقاط توسعه پیش‌بینی نشده است و خود متدولوژی بسیار سنگین است و به جز Maintenance همه فعالیت‌ها را دارد و توسعه دادنش معنایی ندارد.

از نظر مقیاس‌پذیری برای بحرانی و ریسک مشابه آنچه برای USDP گفتیم است و نمی‌توان گفت Life Critical را به خوبی پشتیبانی می‌کند زیرا فرمالیزمش شبه فرمالیزم است. از نظر سایز پروژه ولی کاملاً مقیاس‌پذیر است چون علاوه بر مدل‌سازی کامل و دید معماری که از RUP به ارث برده است و دید مشترک بین اعضا ایجاد می‌کند، فعالیت مدیریت پروژه را هم دارد و علاوه بر همه‌ی این‌ها فعالیت‌های مخصوص مدیریت سازمانی و ارتباط پروژه و افراد پروژه با سازمان را در نظر دارد که برای پروژه‌های خیلی بزرگ ضروری هستند.

قابلیت پیکربندی متدولوژی هست و توصیه هم شده است ولی به دلیل پیچیده بودن متدولوژی کار به شدت دشواری است. در مورد خود RUP که ابزار کامل دارد هم این کار دشوار است ولی توجه کنیم که برای بخش‌هایی از EUP که در RUP نیست ابزار هم نداریم. پس قابلیت پیکربندی هست ولی عملا انقدر دشوار است که مگر برای پروژه‌ای بسیار بزرگ و طولانی اجرایش غیر منطقی است.

انعطاف‌پذیری فرآیند تا حدی در متدولوژی هست. در فعالیت چتری Software Process Improvement در که در discipline های هفت‌گانه‌ی Enterprise Management است امکان تاثیر گذاشتن پروژه‌های دیگر سازمان بر فرآیند پروژه ما وجود دارد. البته این انعطاف در حدی که مثلا در متدولوژی Crystal دیدیم نیست زیرا EUP در حد یک چهارچوب که بخواهیم در بازبینی‌ها پرش کنیم تعیین نشده ولی به هرحال تا حدی انعطاف دارد.

شباهت‌ها در هر دو متدولوژی اثری از توسعه‌پذیری دیده نمی‌شود. هر دو متدولوژی فقط تا سطح Essential Money از بحرانی بودن را پشتیبانی می‌کنند. هر دو متدولوژی می‌توانند برای پروژه‌های با تعداد زیاد فرد درگیر استفاده شوند. هر دو متدولوژی قابلیت پیکربندی دارند.

تفاوت‌ها برای بخشی از پیکربندی متدولوژی EUP می‌توان از ابزار RMC استفاده کرد ولی برای USDP ابزاری نداریم که کار را سخت‌تر می‌کند. متدولوژی EUP طبق آن‌چه در بالا گفتیم قابلیت انعطاف دارد ولی این انعطاف در USDP دیده نشده است. از نظر مقیاس‌پذیری با سایز پروژه می‌توان گفت EUP مقیاس‌پذیرتر است چون مدل‌های سازمانی دارد و به فعالیت‌های مدیریت سازمانی مثل ارتباط افراد با سازمان توجه دارد.

سایر نکات ـ

۱۱.۱.۲ حیطه‌ی کاربرد

به گفته‌ی خود متدولوژی هدف اصلی این متدولوژی اضافه کردن فعالیت‌هایی است که در عمل مورد نیاز است ولی شرکت Rational چون آن فعالیت‌ها را در ابزارش نداشته در RUP نیاورده است. بنابر این طبق گفته‌ی متدولوژی این متدولوژی برای همه پروژه‌هایی که به ارتباط با سازمان نیاز دارند کاربرد دارد. در عمل هم به دلیل سنگین بودن و پیچیدگی متدولوژی و تعداد زیاد مدل‌ها استفاده ازین متدولوژی در پروژه کوچک با تعداد افراد کم درگیر مطرحیت ندارد و حتی باعث کند شدن سرعت افراد هم می‌شود. در جایی اعمال آن سودمند است که فعالیت‌های سازمانی واقعاً مفید باشد. مثلاً وقتی یک سازمان بزرگ مثل دانشگاه هستیم و برای بخش‌های مختلف نرم‌افزار می‌سازیم یا مباحثی مثل دولت الکترونیک که ده ها وزارت خانه با هزاران بخش و زیر بخش که هر کدام ممکن است نرم‌افزار جدایی بخواهند و یا شرکت‌های نرم‌افزاری که با فروش نرم‌افزار پول در می‌آورند و نیاز به زیرساخت سازمانی برای نرم‌افزارهای مختلف دارند.

شباهت‌ها هر دو متدولوژی سنگین وزن هستند و سایز پروژه (از نظر تعداد افراد درگیر) باید نسبتاً بزرگ باشد تا اعمال متدولوژی منطقی شود.

تفاوت‌ها EUP از USDP هم سنگین‌وزن‌تر است و وقتی تفکر سازمانی و ارتباط افراد و پروژه با سازمان لازم باشد بهتر از USDP کاربرد دارد.

سایر نکات در کل به دلیل ساده‌تر بودن و در عین حال جزئیات بیشتر می‌توان گفت از نظر تعداد پروژه USDP در مجموعه‌ی بزرگ‌تری از پروژه‌ها کاربرد دارد.

۲.۲ بخش مدل‌سازی

۱.۲.۲ پشتیبانی از مدل‌سازی سازگار، دقیق و بدون ابهام شی‌گرا

این متدولوژی بر اساس توسعه‌ی متدولوژی RUP ساخته شده است بنابر این زبان اصلی مدل‌سازی آن همان UML است و بیشتر نکات مربوط به UML را در فصل قبل درباره‌ی USDP توضیح دادیم بنابر این اینجا فقط به بررسی مدل‌های جدیدی که اضافه یا کمتر هستند می‌پردازیم.

در این متدولوژی استفاده از DFD به خصوص در جایی که مدل‌سازی سازمانی میکنیم توصیه شده است. به این ترتیب امکان مدل‌سازی وظیفه‌ای در سطح درشت دانگی سازمان به آن اضافه شده است. ضمناً منعی از کشیدن DFD برای سطوح دیگر هم ندارد و می‌توان مدل‌سازی سطح سیستم یا زیرسیستم را به شکل وظیفه‌ای با آن انجام داد.

یک نکته‌ی مثبت دیگر این است که این متدولوژی بر مبنای RUP است و در RUP در فعالیت Business-Modeling موارد کاربرد تجاری را مستقل از مرز سیستم می‌کشیم. بنابر این اگر از usecase استفاده کنیم (در ادامه می‌گوییم که استفاده از usecase اجباری نیست) در واقع در فعالیت BusinessModeling نمودار وظیفه‌ای سطح سازمان را در قالب BusinessUsecaseModel می‌سازیم و همچنین BusinessObjectModel هم تولید می‌شود که در واقع نمودار کلاس سطح سازمان است و شامل کلاس‌های خارج از سیستم‌ما هم هست.

نکته‌ی منفی مدل‌سازی متدولوژی این است که usecase ها محوریت ندارند و اجباری به استفاده از آن‌ها نیست و حتی می‌توان از سند متنی برای توصیف نیازمندی‌ها استفاده کرد که ابهام دارد و معیار دقیق و بدون ابهام‌بودن مدل‌سازی زیر سوال می‌رود.

یک نکته‌ی منفی دیگر این است که چون بر اساس RUP است و در RUP تحلیل یک فعالیت اختیاری است بنابر این گذار مدل‌ها از فضای قلمرو مسئله به قلمرو جواب الزامی ندارد و به طور مثال می‌توان نمودار Component-Diagram را مستقیم و بدون PackageDiagram ساخت.

شباهت‌ها زبان اصلی مدل‌سازی در هر دو متدولوژی UML است که پوشش خوبی از مدل‌سازی رفتاری و ساختاری در سطوح درشت دانگی مختلف دارد. هر دو متدولوژی در قالب OCL پشتیبانی از فرمالیزم دارند.

تفاوت‌ها مدل‌سازی وظیفه‌ای در سطح سازمان در EUP بهتر انجام می‌شود چون DFD نمودار کاملاً مناسبی برای این کار است و هرچند با usecase هم می‌توان آن را انجام داد ولی کیفیت خوبی ندارد و البته فرآیند استخراج usecase های سازمانی بدون در نظر گرفتن سیستم در USDP به کلی حذف شده است پس مدل‌سازی وظیفه‌ای سازمانی فقط در EUP هست. ضمناً در سطوح دیگر (حتی در سطح درون شی‌ای) اگر استفاده از DFD موضوعیت داشته باشد می‌توان از آن استفاده کرد و در سطوح پایین‌تر از زیرسیستم هم مدل‌سازی وظیفه‌ای داشت زیرا usecase حداکثر می‌تواند تا سطح زیرسیستم را مدل‌سازی وظیفه‌ای کند پس در کل به دلیل عدم پیروی صریح از UML مدل‌سازی وظیفه‌ای قوی‌تر شده است. انجام فعالیت تحلیل در EUP که بر اساس RUP است اختیاری است ولی در USDP اجباری است بنابر این گذار مدل‌ها از قلمرو مسئله به قلمرو جواب در USDP آهسته و کامل‌تر از EUP صورت می‌گیرد. استفاده از usecase به عنوان مدل‌نیازمندی هیچ اجباری ندارد و این دقیق و نامبهم بودن مدل‌سازی EUP را نسبت به USDP زیر سوال می‌برد زیرا هیچ قیدی نداریم و می‌توان حتی از سند متنی هم برای نیازمندی‌ها استفاده کرد.

سایر نکات ـ

۲.۲.۲ وجود راهکارهای رفع ناسازگاری و مدیریت پیچیدگی مدل‌ها

از آنجایی که بیشتر مدل‌سازی متدولوژی بر مبنای UML است نکاتی که در مورد USDP در این معیار گفتیم اینجا هم صادق است و فقط به تفاوت‌ها می‌پردازیم.

برای رفع سازگاری راهکاری بیشتر از USDP ارائه نشده است و مدل‌های جدیدی هم تولید می‌شوند که وضع را بدتر می‌کند. همچنین آن تمرکز و صحت‌سنجی نسبی که با usecase محقق شده بود دیگر وجود ندارد.

از نظر مدیریت پیچیدگی در نمودار DFD امکان تشکیل سلسله‌مراتب لایه‌ای فراهم شده است تا از flat شدن و ناخوانا و نافهم شدن نمودار جلوگیری شود. همچنین برای PortfolioManagement اسنادی تولید می‌شود که در واقع دسته‌بندی پروژه‌های سازمان را نشان می‌دهد و آن را هم می‌تواند نوعی مدیریت پیچیدگی در مدل‌ها دانست. ضمناً در فعالیت جدایی به اسم EnterpriseArchitecture به معماری مجموعه نرم‌افزارهای ساخته شده (یا در خدمت) سازمان جداگانه توجه می‌شود که نشان می‌دهد به مدل‌های معماری (مدل‌های معماری در واقع مدل‌هایی هستند که مدیریت پیچیدگی می‌کنند) در سطح سازمان توجه شده است.

شباهت‌ها هر دو متدولوژی به دلیل تعدد بالای مدل‌های تولیدی در حفظ ناسازگاری محصولات دچار مشکل هستند و حفظ سازگاری نیازمند هزینه زمانی قابل توجهی است. هر دو متدولوژی با مدل‌کردن هر جنبه از سیستم در نمودارهای جداگانه و همچنین ارائه سازوکاری‌های گروه‌بندی عناصر در مدل‌های بزرگ و جلوگیری از flat شدن نمودارها سطح خوبی از مدیریت پیچیدگی را دارند.

تفاوت‌ها تعدد بیشتر مدل‌ها در EUP و از بین رفتن محوریت usecase حفظ ناسازگاری مدل‌ها در EUP سخت‌تر کرده است و ازین نظر کار در USDP ساده‌تر است. از نظر مدیریت پیچیدگی نکته‌ی مثبتی که در EUP دیده می‌شود توصیه به مدیریت پیچیدگی مدل‌های سازمانی با در آوردن معماری سازمانی و یا اسناد فعالیت PortfolioManagement است که البته در USDP چون کلا دغدغه‌ی سازمانی نداریم همچین مدل‌هایی وجود ندارد.

سایر نکات ـ

فصل ۳

مقایسه‌ی متدولوژی‌های USDP و OPM

۱.۳ بخش فرآیند

۱.۱.۳ تعریف متدولوژی

متدولوژی عملاً به عنوان چرخه‌ی عمومی ایجاد نرم‌افزار تعریف شده است و به شکل فرآیند محور تعریف شده است. تعریفش از این نظر که پوشش کامل درشت دانه را داراست کامل است ولی با توجه به اینکه جزئیات عملی لازم برای انجام متدولوژی در آن نیست در عمل برای استفاده باید تکنیک‌های مختلف به آن اضافه شود تا کامل شود پس نمی‌توان تعریف فرایند متدولوژی را کامل دانست.

ضمناً در حین تعریف که فرآیندمحور انجام شده است تا حدی به نحوه‌ی تکمیل OPD اشاره شده است ولی اشاره‌ای به نقش‌های درگیر در فرایندها نکرده و ناقص است.

زبان مدل‌سازی OPD (و معادلش OPL) است که قبل از عرضه‌ی متدولوژی به شکل دقیقی تعریف شده بود. اگر چه به شکل کلی گفته شده است در فرآیند این نمودار چگونه ساخته و تکمیل شود ولی جزئیات کافی برای ساخت و تکمیل نمودار در عمل در متدولوژی نیامده و ازین نظر هم تعریف ناقص است.

از آنجا که متدولوژی خیلی کلی و سطح بالاست صحبتی از تکنیک‌ها و قواعد نشده است.

فعالیت‌های چتری به شکل صریح در متدولوژی نیستند ولی تحلیل امکان‌سنجی اولیه انجام می‌شود که نشان می‌دهد کمی به مدیریت ریسک توجه دارد.

شباهت‌ها هر دو متدولوژی فرآیند محور تعریف شده‌اند. فعالیت مدیریت ریسک در دو متدولوژی کمی دیده می‌شود.

تفاوت‌ها فرآیند از نظر کامل بودن با اینکه در OPM پوشش کلی بیشتری دارد ولی چون جزئیات خیلی خیلی کمی دارد می‌توان گفت تعریف USDP کامل‌تر است. از نظر زبان مدل‌سازی UML زبان غنی‌تری از OPD است و به خصوص جنبه‌ی رفتاری را که اصلا نمی‌توان در OPD نشان داد UML به خوبی

مدل می‌شود ولی خب در مشکلUML حفظ سازگاری مدل را داریم که در OPD نداریم. در فرآیند USDP نحوه‌ی تکمیل و ساخت مدل‌ها خیلی دقیق است ولی در OPM کلی گویی شده است و علاوه بر آن در USDP نقش‌ها هم گفته شده است که بر OPM برتری دارد.

سایر نکات ـ

۲.۱.۳ پوشش چرخه‌ی عمومی تولید نرم‌افزار

این متدولوژی تحلیل امکان‌سنجی و استخراج نیازمندی‌ها را در فاز Initiating از خود انجام می‌دهد. در فاز Developement مدل‌سازی قلمرو مسئله، طراحی معماری و طراحی جزئی را در خود دارد و به پیاده‌سازی و تست و بخشی از استقرار هم می‌پردازد. در فاز Deploying فعالیت استقرار تکمیل می‌شود و به Maintenance می‌پردازیم. در نهایت به مرگ پروژه و کارهای لازم برای آن هم در همین فاز توجه شده است. پس پوشش کامل است اما جزئیات آن بسیار کم است و متدولوژی در حد یک توصیه نامه‌ی کلی یک کتاب آشنایی با مهندسی نرم‌افزار است!

شباهت‌ها هر دو متدولوژی فعالیت‌های تحلیل امکان سنجی و مدل‌سازی قلمرو مسئله و استخراج نیازمندی‌ها وطراحی معماری وطراحی جزئیات و پیاده‌سازی و تست و استقرار پوشش داده می شوند.

تفاوت‌ها در OPM به فعالیت‌های نگهداری و پشتیبانی و بالانگهداشتن نرم‌افزار و همچنین مرگ پروژه توجه شده که در USDP وجود ندارند. ولی در مواردی که در هر دو متدولوژی پوشش داده شده است جزئیات و توضیحات USDP کامل‌تر و جامع‌تر است. مثلاً برای استخراج نیازمندی‌ها تمام کارهای و مدل‌های لازم برای استخراج نیازمندی‌ها در USDP مشخص شده است ولی در OPM فقط یک اشاره‌ی کلی به آن شده است. یا مثلاًدر طراحی معماری در UML هم معماری قلمرو مسیله هم معماری قلمرو جواب و هم پیکربندی فیزیکی سیستم را داریم ولی در OPM پیکربندی فیزیکی سیستم را نمی‌توان نشان داد.

سایر نکات ـ

۳.۱.۳ پشتیبانی از فعالیت‌های چتری

مدیریت ریسک

چون در فاز Initiating باید توجیه اقتصادی و scope پروژه مشخص شود بنابر این تحلیل امکان سنجی اولیه تا حدی در این متدولوژی هست و به کاهش ریسک اقتصادی کمک می‌کند. اما اثر دیگری از سایر روش‌های کاهش ریسک در آن نیست. همچنین پیکربندی فیزیکی را مدل نمی‌کند که خود ریسک جدی در زمان استقرار ایجاد می‌کند.

البته وجود فعالیت تست و اهمیت مجزا به آن می‌تواند در کاهش ریسک موثر باشد. ورود نسبتاً سریع به قلمرو جواب با استخراج معماری هم در کاهش ریسک موثر است.

شباهت‌ها هر دو متدولوژی با تحلیل امکان‌سنجی اولیه سعی در کاهش ریسک‌های اصلی دارند. هر دو متدولوژی فعالیت تست را صراحتاً تعریف می‌کند که می‌تواند جلوی برخی ریسک‌ها را بگیرد. هر دو متدولوژی مشارکت فعال و دائمی کاربر که می‌توانست جلوی ریسک را بگیرد ندارند. هر دو متدولوژی معماری را زود استخراج می‌کنند که باعث می‌شود زود به قلمرو جواب وارد شویم و ریسک کاهش یابد.

تفاوت‌ها در USDP از روش‌های چرخشی‌تکراری بودن، برنامه‌ریزی بر مبنای ریسک، ساخت پروتوتایپ، حد قابل قبولی از Continous V&V ، حدی از CI و همچنین قابلیت استفاده از روش‌های فرمال وجود دارد که در OPM نیست و USDP خیلی بهتر ریسک را مدیریت کرده است.

سایر نکات در این معیار USDP برتری مطلق دارد و نسبی نیست.

مدیریت پروژه

در این متدولوژی به شکل صریح به فعالیت‌های plan و schedule و monitor کردن آن پرداخته نشده است.

شباهت‌ها ندارند.

تفاوت‌ها در USDP سه کار اصلی مدیریت پروژه تعریف شده است و در OPM نیست.

سایر نکات متدولوژی USDP برتری مطلق دارد.

تضمین کیفیت

به دلیل وجود مدل واحد و امکان نسبی‌ای که برای ردیابی اجزا مدل به نیازمندی هست می‌تواند کیفیت بهبود یابد. همچنین چک مکرر سیستم در Maintenance که توصیه شده است برای حفظ کیفیت است و نکته‌ی مثبتی است. وجود فعالیت تست هم می‌تواند بر کنترل کیفیت محصولات تاثیر مثبت بگذارد.

شباهت‌ها هر دو با رهگیری به نیازمندی‌ها می‌توانند به بهبود کیفیت کمک کنند. هر دو با وجود فعالیت تست می‌تواندن تا حدی کیفیت محصولات را مورد بررسی قرار دهند.

تفاوت‌ها در OPM به حفظ کیفیت در فاز نگهداری توجه شده و در USDP چون آن فاز را ندارد اشاره‌ای هم نشده است. در USDP به دلیل فرمالیزم و وجود چک‌لیست کیفی و تعریف معیارهای کیفیت در فاز Elaboration و امکان اعمال بازبینی مکرر می‌توان کنترل و تضمین کیفیت داشت که در OPM نیست.

سایر نکات هر چند نکات مثبتی در OPM هست ولی در کل می‌توان گفت USDP برتری دارد.

۴.۱.۳ بی‌درزی و همواری گذار

در این متدولوژی فقط یک مدل وجود دارد که به بی درزی کمک می‌کند و عدم معرفی مدل‌های جدید باعث همواری گذار هم می‌شود. در عین حال عدم مدل‌سازی رفتاری و تجمیع همه جنبه‌ها در یک نمودار بزرگ و پیچیده می‌تواند بی‌درزی را زیر سوال ببرد زیرا مدل‌های رفتاری برای پیاده‌سازی لازم هستند و وجود ندارند.

شباهت‌ها مدل‌های ساختاری هر دو متدولوژی به تدریج از قلمرو مسئله به جواب می‌روند که بی درزی ایجاد می‌کند.

تفاوت‌ها مدل‌سازی رفتاری به کلی در OPM نیست که باعث درز است ولی در USDP مدل‌های رفتاری هم به تدریج از قلمرو مسئله به جواب می‌روند. همچنین تکراری چرخشی بودن USDP همواری‌گذار را آسان‌تر می‌کند. البته یک نکته‌ی مثبت در OPM هست که فقط یک مدل تکمیل می‌شود و به همواری کمک می‌کند ولی در USDP با اینکه بیشتر مدل‌ها به شکل تکمیل شونده جلو می‌روند، ولی به هر حال مدل جدید هم تولید می‌کنیم که به همواری آسیب می‌زند.

سایر نکات ـ

۵.۱.۳ میزان توجه به نیازمندی‌های وظیفه‌ای و غیروظیفه‌ای

در این متدولوژی در فاز Initiating به استخراج نیازمندی‌های وظیفه‌ای پرداخته می‌شود. بنابر این فعالیت استخراج نیازمندی‌ها و مدل‌سازی آن‌ها را دارد. و مبتنی بر نیازمندی‌ها پیش رفته زیرا مراحل بعدی تحلیل و طراحی بر اساس آن پیش میرود. البته برای خود نیازمندی مدل خاصی ندارد و ازین نظر ضعیف است. امکانی برای تغییر یا تکمیل نیازمندی‌ها در این متدولوژی دیده نشده است زیرا متدولوژی آبشاری است. یک نکته‌ی دیگر این است که در نمودار OPD می‌توان تا حدی شبیه usecase مدل‌سازی نیازمندی را انجام داد و اگر آن گونه پیش برویم متدولوژی می‌تواند نیازمندی‌رانه باشد ولی در عمل آنچه در مثال‌های متدولوژی انجام شده بیشتر شبیه DFD است که قابلیت رهگیری خوبی به نیازمندی ندارد و دیگر نیازمندی‌رانه نیست.

شباهت‌ها استخراج و مدل‌سازی نیازمندی‌ها در هر دو متدولوژی هست. هر دو متدولوژی مبتنی بر نیازمندی هستند.

تفاوت‌ها امکان تغییر و یا تکمیل نیازمندی‌ها به دلیل آبشاری بودن فرآیند در OPM دیده نشده است ولی در USDP هست. همچنین متدولوژی USDP به دلیل Usecase Driven بودن نیازمندی‌رانه هم هست ولی در مورد OPM گفتیم که با وجود این که امکان انجام مدل‌سازی نیازمندی‌رانه با نمودارش ممکن است ولی در عمل در مثال‌های توصیه شده توسط متدولوژی کاری شبیه کشیدن DFD انجام شده که رهگیری مستقیم به نیازمندی در آن دیگر غیربدیهی است و نمی‌توان گفت نیازمندی‌رانه خواهیم بود. در آخر نیازمندی‌های غیروظیفه‌ای هم در USDP جدی‌تر گرفته می‌شوند.

سایر نکات در کل USDP برتری مطلق دارد و برتری در OPM در این معیار دیده نمی‌شود.

۶.۱.۳ قابلیت تست، میزان ملموس‌بودن و قابلیت رهگیری به نیازمندی‌ها

قابلیت تست

در OPM فقط یک نمودار داریم که می‌تواند به تست‌پذیری کمک کند ولی از طرف دیگر این تک‌نموداره بودن باعث پیچیدگی شدید آن نمودار می‌شود و می‌دانیم تست کردن محصول پیچیده دشوار است.

شباهت‌ها ندارند.

تفاوت‌ها در OPM تعداد نمودار به یک کاهش یافته که در مقابل تعداد زیاد نمودار USDP به تست‌پذیری کمک می‌کند. از سوی دیگر در USDP هر تک مدل خیلی خیلی ساده‌تر و قابل فهم‌تر از OPM است و از آن جهت به تست‌پذیری بهتر می‌انجامد.

سایر نکات در کل برای سیستم خیلی بزرگ OPD آنقدر پیچیده می‌شود که تست‌کردن تعداد زیاد نمودار USDP ساده‌تر از تست کردن یک نمودار به این بزرگی و پیچیدگی می‌شود.

میزان ملموس‌بودن

از نظر ملموس‌بودن محصولات برای کاربر مشکل جدی هست. نمودار OPD برای ساده‌ترین سیستم‌ها هم می‌تواند بسیار شلوغ شود و نمی‌توان یک جنبه‌ی سیستم را جداگانه در آن دید و در کل مدلی که از دید کاربر باشد در آن وجود ندارد ولی خب نمودار سلسله مراتبی است و شاید مراتب خیلی سطح بالای آن قابل فهم باشند. ملموس بودن برای سازندگان نرم‌افزار هم دچار مشکل است. زیرا این همه پیچیدگی و دشواری درک را دارد در عین حال هنوز جنبه‌های رفتاری و درون شی‌ای که برای برنامه‌نویسان مفید است را ندارد. همچنین مدل متنی OPL برخلاف ادعای متدولوژی هیچ فایده‌ای برای تولید نرم‌افزار ندارد.

شباهت‌ها ملموس بودن نسبی برای ذی‌نفعان قلمرو مسئله در هر دو متدولوژی هست.

تفاوت‌ها هم در زمینه قلمرو مسئله و هم قلمرو جواب که برنامه نویسان استفاده می‌کنند، به دلیل پیچیدگی زیاد OPD می‌توان گفت مدل‌های USDP ملموس‌تر هستند.

سایر نکات ـ

قابلیت رهگیری به نیازمندی‌ها

تک مدله بودن این متدولوژی می‌تواند باعث آسانی رهگیری به نیازمندی‌ها شود ولی در عمل همین تک مدله بودن آنقدر مدل را پیچیده می‌کند که دنبال کردن اجزای مدل به نیازمندی بسیار دشوار می‌شود. ضمناً گفتیم که امکان

مدل‌سازی نیازمندی رانه با OPD هست ولی عملا چیزی که در مثال‌های متدولوژی زده شده است بیشتر به DFD شبیه است که قابلیت رهگیری به نیازمندی‌ها را ندارد. ضمناً مدل‌سازی رفتاری هم ندارد که باعث می‌شود در نهایت ترتیب اجرا در کد به هیچ شکلی قابل ردیابی خوبی به مدل نداشته باشد.

شباهت‌ها ندارند.

تفاوت‌ها به کمک سناریوهایی که برای تحقق usecase باید ساخته شوند در USDP رهگیری به نیازمندی را داریم ولی در OPM در عمل به دلیل پیچیدگی زیاد مدل قابلیت رهگیری به نیازمندی‌ها مناسب نباشد. همچنین در OPM هر چقدر هم خوب مدل‌سازی کنیم مدل‌سازی رفتاری درستی ندارد و رهگیری جنبه‌های رفتاری به نیازمندی غیرممکن است.

سایر نکات ـ

۷.۱.۳ میزان مشارکت فعال کاربر

مشارکت فعال کاربر در این متدولوژی دیده نمی‌شود و مشارکتش در حد استخراج نیازمندی‌ها و تایید نرم‌افزار در مرحله‌ی Validation است.

شباهت‌ها در هر دو متدولوژی مشارکت نسبی کاربر برای استخراج نیازمندی‌ها هست.

تفاوت‌ها هرچند هر دو استخراج نیازمندی‌ها را دارند ولی مشارکت فعال کاربر در USDP بیشتر دیده می شود زیرا استخراج نیازمندی‌ها فعالانه‌تر است و از پروتوتایپ هم استفاده می‌شود و به شکل آبشاری نیست و تا انتهای پروژه قبل از Transition انجام می‌شود. همچنین چرخشی تکراری بودن USDP امکان مشارکت کاربر در جلسات بازبینی و برنامه‌ریزی را فراهم کرده است.

سایر نکات ـ

۸.۱.۳ قابلیت اجرا و سادگی و موثر بودن اجرا

به این دلیل که متدولوژی بسیار ساده و انتزاعی تعریف شده است به قابل اجرا بودن آن کمک می‌کند ولی در عمل چون هیچ بخشی از آن صلب نیست امکان اجرای عملی آن در هر نوع پروژه‌ای دچار لطمه می‌شود و در کل نمی‌توان گفت قابل اجراست. از نظر موثر بودن اجرا به ابزار خاصی وابستگی نداریم که نکته‌ی خوبی است. در این متدولوژی به مدیریت پروژه در قالب زمان‌بندی و مدیریت افراد توجه نشده است که به اجرای موثر لطمه می‌زند. تنها نکته‌ی مثبتی که دیده می‌شود توجه به طراحی معماری است و معماری مدلی است که می‌تواند بین افراد تیم تمرکز مشترک ایجاد کند و باعث اجرای راحت‌تر و موثرتر متدولوژی شود. تکنیک خطاخیز خاصی در این متدولوژی دیده نمی‌شود و این هم جلوی لطمه خوردن به سادگی اجرا را می‌گیرد.

شباهت‌ها هر دو متدولوژی با استخراج معماری دید متمرکز ایجاد کرده‌اند و هر دو از استفاده از روش‌های خطاخیز جلوگیری کرده‌اند که باعث اجرای موثر می‌شود.

تفاوت‌ها فرآیند در OPM به نسبت USDP ساده است که قابلیت اجرا را بالا می‌برد ولی در عین حال انتزاعی هم هست که به قابلیت اجرا ضربه می‌زند. معماری در هر دو متدولوژی باعث دید متمرکز شده است ولی در USDP معماری جدی‌تر گرفته می‌شود و بعد از تکمیل در Elaboration تا انتها محوریت اصلی است و این محوریت در OPM کمرنگ است و این محوریت جدی‌تر باعث تمرکز جدی‌تر و موثربودن اجرای بیشتر است. متدولوژی USDP برای مدل‌هایش نیاز جدی به ابزار دارد که البته گفتیم لزوماً نکته‌ی بدی نیست و نکته‌ی خوبی هم می‌تواند باشد. فعالیت‌های مدیریت پروژه در USDP هستند ولی در OPM دیده نمی‌شوند که به اجرای موثرتر USDP می‌انجامد.

سایر نکات با توجه به تفاوت‌های عنوان شده به جز سادگی که در OPM هست و ممکن است در پروژه کوچک آن را قابل اجراتر از USDP کند اگر سایز پروژه کمی بزرگ شود USDP هم قابلیت اجرای بهتری دارد هم موثرتر اجرا خواهد شد.

۹.۱.۳ مدیریت پیچیدگی فرآیند

این متدولوژی خود را در سه فاز Initiating و Developing و Deploying تعریف کرده و در داخل هر کدام فعالیت‌ها را جداگانه تعریف کرده تا فهم فرآیند ساده‌تر شود.

شباهت‌ها هر دو متدولوژی با تعریف فازها و قراردادن فعالیت‌های ریزدانه در داخل فازها سعی کرده‌اند از پیچیدگی فرآیند و flat شدن آن جلوگیری کنند.

تفاوت‌ها در USDP در هر فاز چرخه ثابتی تکرار می شود ولی در OPM فعالیت‌های هر فاز جداگانه مطرح شده است که درک آن برای استفاده کنندگان از متدولوژی می‌تواند اسان‌تر باشد.

سایر نکات هر دو متدولوژی مدیریت پیچیدگی فرآیند دارد ولی در نهایت می‌توان گفت فرایند OPM ساده‌تر فهمیده می‌شود و ازین نظر بهتر است.

۱۰.۱.۳ توسعه‌پذیری ، مقیاس‌پذیری، قابلیت پیکربندی و انعطاف‌پذیری

این متدولوژی نقاط توسعه و بخش‌هایی که باید توسعه داده شوند یا قابل توسعه هستند را مشخص نمی‌کند پس توسعه‌پذیر نیست.

از نظر مقیاس‌پذیری بحرانی وضع خوبی ندارد و حتی برای سیستم Discretionary Money هم قابل استفاده نیست چون فرمالیزم که اصلا ندارد و مدل‌سازی‌اش هم ناقص است ومدل‌سازی رفتاری ضعیفی دارد. از نظر سایز پروژه به معنی تعداد افراد درگیر هم وضع خوبی ندارد زیرا مدل‌سازی تک مدله است و کار کردن تعداد زیادی فرد

روی یک مدل بسیار دشوار است و وقتی همزمان انجام شود می‌تواند باعث خطا هم بشود. ضمناً مدیریت روابط افراد هم جدی گرفته نشده است و با افزایش سایز تیم مشکل جدی خواهیم داشت.
در این متدولوژی امکان پیکربندی فرایند برای پروژه خاص دیده نشده پس قابلیت پیکربندی ندارد و در متدولوژی نقاط و فعالیت‌هایی که بخواهند فرآیند را در حین اجرا اصلاح کنند یا تغییر دهند هم نداریم پس انعطاف‌پذیر هم نیست. البته اینکه میگوییم پیکربندی نداریم یعنی مثل Crystal صراحتاً مجموعه‌ای از practice ها را نداده که انتخاب کنیم و مجبوریم خودمان با مراجعه به منابع دیگر بفهمیم چگونه متدولوژی را تکمیل کنیم وگرنه اگر بخواهیم در عمل از OPM استفاده کنیم به پیکربندی نیاز داریم زیرا فرایند انتزاعی است و قابل اجرا نیست.

شباهت‌ها هر دو متدولوژی از قابلیت توسعه‌پذیری و انعطاف‌پذیری برخوردار نیستند.

تفاوت‌ها متدولوژی USDP امکان پیکربندی اولیه را دارد ولی در OPM همچین پیش‌بینی نشده است پس ازین نظر USDP برتری دارد. همچنین مقیاس‌پذیری از نظر بحرانی بودن در USDP بهتر است چون فرمالیزم نسبی داریم و از نظر تعداد افراد درگیر هم مقیاس‌پذیرتر است چون تعدد و تنوع مدل داریم و فعالیت‌های مدیریت پروژه جدی‌تر است و معماری هر چند در هر دو هست ولی در USDP جدی‌تر است که بین افراد دید مشترک ایجاد می‌کند.

سایر نکات ـ

۱۱.۱.۳ حیطه‌ی کاربرد

خود متدولوژی کاربرد خاصی ذکر نمی‌کند و با توجه به اینکه متدولوژی عملاً چرخه‌ی عمومی ایجاد نرم‌افزار است با اضافه کردن فعالیت‌ها و practice های لازم می‌توان آن را برای هر پروژه‌ای استفاده کرد. البته به دلیل ضعف مدل‌سازی مقیاس‌پذیری آن پایین است و در پروژه‌های بحرانی و یا پروژه‌های بزرگ با تعداد افراد زیاد قابل استفاده نخواهد بود.

شباهت‌ها هر دو متدولوژی برای ساخت سیستم‌های شی‌گرا قابل به کاربردن هستند

تفاوت‌ها وقتی سایز پروژه کوچک باشد و پروژه بحرانی نباشد OPM (با فرض پیکربندی و از بین بردن انتزاعی بودنش) قابل کاربرد است ولی با افزایش سایز پروژه دیگر مدل‌سازی این متدولوژی جواب نمی‌دهد و در آن پروژه‌ها USDP کاربرد پیدا می‌کند.

سایر نکات درست است USDP سنگین وزن است ولی به دلیل مدل‌سازی غنی‌تر در کل می‌توان گفت حیطه‌ی کاربرد USDP بیشتر است و برتری دارد.

۲.۳ بخش مدل‌سازی

۱.۲.۳ پشتیبانی از مدل‌سازی سازگار، دقیق و بدون ابهام شی‌گرا

زبان مدل‌سازی این متدولوژی نمودار OPD است و البته یک سند متنی معادل در قالب OPL هم برای هر OPD وجود دارد.

امکان مدل‌سازی وظیفه‌ای در OPD در قالب process های این نمودار هست و ساختار هم با روابطی که بین object ها و process های این نمودار مدل می‌شوند قابل نمایش است. اما هیچ مدل‌سازی رفتاری صریحی در این متدولوژی نیست و برای استخراج رفتار باید به شکل ضمنی Invocation فرآیندهای نمودار را بررسی کرد تا از آن‌ها رفتار استخراج شود.

با توجه به دیده شده مراحل استخراج نیازمندی و تحلیل و سپس طراحی در متدولوژی این گذار در نمودار هم می‌تواند مشخص باشد و در ابتدا فقط شامل اشیا و فرآیندهای قلمرو مسئله است و به تدریج اشیا و فرآیندهای دیگری که برای پیاده‌سازی و تحقق کامپیوتری لازم هستند اضافه خواهند شد. البته این گذار کامل نیست زیرا برخی جنبه‌های قلمرو جواب مثل پیکربندی فیزیکی سیستم در متدولوژی مدل نمی‌شوند.

از نظر سطوح مختلف درشت دانگی هر چند امکان مدل‌سازی سطح سازمان به شکل وظیفه‌ای و ساختاری با OPD هست (چون OPD مشابه DFD سلسله مراتبی است) ولی در متدولوژی توصیه نشده است. به دلیل همین سلسله مراتبی بودن OPD امکان مدل‌سازی در دو نوع ساختاری و وظیفه‌ای در سطح‌های سیستم و زیرسیستم و بین شی‌ای هم وجود دارد (زیرسیستم‌ها را با Object هایی که در دل خود اشیا و وظایف دیگری هستند مدل می‌کند). امکان مدل‌سازی ساختاری و وظیفه‌ای داخل اشیا هم با این متدولوژی هست زیرا می‌توان داخل اشیا را باز کرد ولی خب مشکلی که دارد این است که یک شی ممکن است در بخش‌های مختلفی از نمودار ظاهر شده باشد و مثلاً اگر می‌خواهیم attribute های یک شی را در بیاوریم باید برویم در تمام بخش‌های نمودار دنبال آن شی بگردیم و ببینیم در هر بخش چه زیر اشیائی دارد. مدل‌سازی رفتاری به کلی ضعیف است و در هیچ کدام از سطوح وجود ندارد. ضمناً استخراج ضمنی آن هم بسیار ضعیف‌تر از یک نمودار رفتاری واقعی مثل SequenceDiagram است زیرا در SequeunceDiagram در سطح نمونه‌ی اشیا صحبت می‌کنیم و یک شی می‌تواند چندبار ظاهر شود ولی در OPD در سطح کلاس صحبت می‌کنیم و حتی اگر به شکل ضمنی هم رفتار استخراج کنیم، تمام جنبه‌های رفتاری استخراج نخواهند شد.

هیچ پشتیبانی از فرمالیزم در OPD دیده نشده است.

شباهت‌ها هر دو متدولوژی مدل‌سازی ساختاری و وظیفه‌ای در سطح سیستم تا بین اشیا را خوب انجام می‌دهند. هر دو متدولوژی به مدل‌سازی سازمانی توجه جدی ندارند.

تفاوت‌ها مدل‌سازی رفتاری سطح سیستم در UML در قالب StateTransitionDiagram قابل انجام است ولی در OPD نداریم. مدل‌سازی رفتاری بین زیرسیستم‌ها با InteractionOverview قابل انجام است و مدل‌سازی رفتاری خود زیرسیستم‌ها در قالب StateTransitionDiagram هم ممکن است

و این مدل‌سازی‌های رفتاری در OPD وجود ندارد. مدل‌سازی رفتاری بین اشیا هم در USDP با کمک SequenceDiagram با دقت بالا قابل انجام است و در مدل متدولوژی OPM امکان پذیر نیست. مدل‌سازی ساختار داخل اشیا در USDP به شکل واضح در ClassDiagram هست ولی همانطور که گفتیم در OPD استخراج این ساختار ضمنی و دشوار است و USDP بهتر عمل می‌کند. مدل‌سازی رفتار داخل متدها هم با ActivityDiagram در USDP انجام می‌شود که در OPD ممکن نیست. در مورد مدل‌سازی وظیفه‌ای ولی OPD چون می‌توان نمودارهایی شبیه DFD تولید کرد می‌تواند بهتر از USDP انجام شود به خصوص اینکه usecase حداکثر در سطح زیرسیستم است ولی با OPD می‌توان وظایف درون‌شی‌ای را هم مدل کرد. ضمناً پیکربندی فیزیکی سیستم در USDP با DeploymentDiagram مدل می‌شود و نمودار خیلی مهمی است که در OPD قابل نمایش نیست.

در آخر از نظر فرمالیزم هم USDP با وجود OCL پشتیبانی دارد ولی هیچ پشتیبانی از فرمالیزم در OPD نیست و USDP بهتر عمل کرده است.

سایر نکات به طور خلاصه می‌توان گفت در مدل‌سازی ساختاری و رفتاری USDP کاملا بر OPM برتری دارد ولی مدل‌سازی وظیفه‌ای در OPM می‌تواند بهتر انجام شود. از نظر گذار از قلمرو مسئله به قلمرو جواب و پیاده‌سازی در UML برخی نمودارها جدید تولید می‌شوند مثلا در ابتدا PackageDiagram را داریم و در قلمرو جواب ComponentDiagram است ولی در OPD این پیشروی به قلمرو جواب با اضافه شدن جزئیات و فرآیندها و کلاس‌های جدید به نمودار انجام می‌شود که ازین نظر که نمودار جدید تولید نمی‌شود تا حدی باعث هموازی می‌شود ولی پیچیدگی نمودار می‌تواند کار را سخت کند و جداکردن برخی نمودارها در قلمرو مسئله و جواب می‌تواند تولید آن‌ها در عمل را ساده‌تر کند.

۲.۲.۳ وجود راهکارهای رفع ناسازگاری و مدیریت پیچیدگی مدل‌ها

در این متدولوژی با مدل کردن تمام جنبه‌های مد نظر متدولوژی در یک نمودار ناسازگاری مدل وجود ندارد زیرا تمامی آنچه باید مدل شود در یک مدل حضور دارد و تعدد مدل نداریم پس به طور ذاتی مدل سازگار است و نیاز به تسهیلات فرآیندی حفظ سازگاری ندارد.

سلسله‌مراتبی بودن نمودار OPD مصداق مدیریت پیچیدگی است و در واقع تکنیک لایه‌بندی است.

شباهت‌ها هر دو متدولوژی با تشکیل سلسله مراتب از اجزای نمودارها و انجام Layering به مدیریت پیچیدگی مدل پرداخته‌اند.

تفاوت‌ها در UML علاوه بر وجود سلسله مراتب هر جنبه از سیستم در نمودار جدا مدل شده است که باعث فهم آسان‌تر و استخراج آسان‌تر اطلاعات شده است و مدل‌ها خلوت‌تر شده‌اند و خواندن آن‌ها دشوار نیست در حالیکه در OPD جنبه‌های مختلف مخلوط شده و در یک نمودار نمایش داده می‌شوند و نمودار شلوغ است و بدتر از آن اینکه برخی جنبه‌ها صراحتاً از نمودار قابل استخراج نیست (مثل Attribute

ها) پس در کل مدیریت پیچیدگی USDP وضع مطلوب‌تری دارد. اما این مدیریت پیچیدگی به قیمت تعدد مدل‌ها تمام شده است و باعث شده است که از نظر حفظ سازگاری OPD برتری مطلقی به UML که در USDP استفاده شده است داشته باشد.

سایر نکات ـ

۳.۳ سایر نکات

جدای از سطح بالا بودن تعریف فرآیند و همچنین ضعف‌های مدل‌سازی یک نکته‌ی منفی دیگر در متدولوژی دیده می‌شود که کار آماده‌سازی محیط کاربر برای نصب نرم‌افزار را در Development قرار داده است که کمی عجیب است و وقتی متدولوژی آبشاری تعریف شده بهتر بود در همان فاز Deployment گفته می‌شد.

شباهت‌ها ندارند

تفاوت‌ها در USDP برخی کارهای مربوط به فاز Transition مثل آماده‌سازی اسناد زودتر انجام می‌شود ولی نصب نرم‌افزار معمولا در خود Transition است و حتی اگر زودترهم رخ دهد با توجه به چرخشی‌تکراری بودن متدولوژی منطقی است ولی در متدولوژی OPM که چرخشی‌تکراری نیست این زودتر نصب کردن در فاز Development توجیهی ندارد. یک نکته مثبت دیگر در USDP در مورد بستن قرار داد است که اولا به آن و نجوه‌ی اجرایش اشاره شده و دو مرحله‌ای هم هست که هم ریسک را کاهش می‌دهد و هم باعث می‌شوند برنامه‌ریزی درست‌تر باشد ولی در OPM با اینکه کل چرخه‌ی ایجاد را پوشش داده به آن اهمیت جدی نداده است و فقط یک بار scope سیستم تعیین می‌شود و قرارداد بر همان اساس بسته می‌شود.

سایر نکات ـ

فصل ۴

مقایسه‌ی متدولوژی‌های USDP و FOOM

۱.۴ بخش فرآیند

۱.۱.۴ تعریف متدولوژی

متدولوژی به شکل فرآیند محور تعریف شده است و آبشاری است و دو فاز تحلیل و طراحی را دارد. بنابر این بخش عمده‌ی چرخه‌ی تولید نرم‌افزار را پوشش نداده و تعریف فرآیند اصلا کامل نیست و نحوه‌ی توسعه‌ی متدولوژی به یک متدولوژی کامل توصیف نشده است. علاوه بر آن برخی بخش‌های فرآیند تعریف شده ابهام دارند یا ناقص هستند. مثلا نحوه‌ی دقیق انجام تبدیل Transaction ها در قالب پخش کردن آن در میان متدهای کلاس‌های دیگر گفته نشده و کلی گویی شده است. همچنین به جز کلاس‌های منو و Form مورد نظر معلوم نیست دیگر چه جزئیات طراحی باید به DataModel یا همان نمودار کلاس اضافه شوند و این کار چگونه انجام شود.

محصولاتی که در طول فرآیند تولید می‌شوند با دقت تعریف شده‌اند ولی متدولوژی به نقش‌های درگیر در فرآیندها توجهی ندارد. البته محصولاتش حتی در حیطه‌ی خود متدولوژی یعنی طراحی و تحلیل کامل نیست زیرا همه‌ی کلاس‌های قلمرو جواب (به خصوص کلاس‌های پایگاه‌داده) استخراج نمی‌شوند.

زبان مدل‌سازی مخصوص متدولوژی است و شامل نمودار کلاس و OODFD و MessageChart و مجموعه‌ای از مستندات متنی است که البته مستندات متنی‌اش ساخت‌یافته است و ازین نظر نکته‌ی مثبتی است و جلوی ابهام را گرفته است.

تکنیک‌ها و قواعد هم در حد قابل قبولی تعریف شده‌اند مثل نحوه‌ی استخراج Transaction ها و یا نحوه‌ی انجام واکشی و یا ذخیره‌سازی داده درون کلاس‌ها توضیح داده شده است. (البته تکنیکش از نظر شی‌گرایی منفور است ولی به هر حال توضیح داده شده‌است و به خواننده واگذار نشده است که نکته‌ی مثبتی است).

فعالیت‌های چتری در متدولوژی صراحتاً تعریف نشده‌اند و به جز رهگیری به نیازمندی‌ها تکنیک دیگری که نمود مدیریت ریسک یا تضمین کیفیت باشد در آن دیده نمی‌شود.

شباهت‌ها هر دو متدولوژی فرآیند محور هستند و محصولات تولیدی را به خوبی توضیح داده‌اند. هر دو متدولوژی

فرآیند ساخت نرم‌افزار را کامل پوشش نمی‌دهند.

تفاوت‌ها در USDP فرآیند از نظر پوشش ساخت نرم‌افزار کامل‌تر است. در USDP علاوه بر محصولات به نقش‌ها هم در فرآیند توجه شده است. زبان مدل‌سازی و محصولاتی که در USDP تولید می‌شوند جامع‌تر هستند و می‌توان تمام جنبه‌های سیستم (به جز مدل وظیفه‌ایش که ضعف‌هایی دارد) را در آن به خوبی مدل کرد و در مقایسه با FOOM خیلی کامل‌تر است. تکنیک‌ها در USDP خیلی کامل‌تر هستند. فعالیت‌های چتری در USDP خیلی جدی هستند ولی در FOOM عملاً توجهی به آن‌ها نشده است. فرآیند USDP خیلی دقیق و سازگار تعریف شده است ولی همانطور که گفتیم در مورد تبدیل Transaction ها به متد در متدولوژی FOOM ابهام وجود دارد و کلی گویی کرده است.

سایر نکات ـ

۲.۱.۴ پوشش چرخه‌ی عمومی تولید نرم‌افزار

این متدولوژی فعالیتی برای استخراج نیازمندی‌ها و تحلیل اولیه‌ی امکان‌سنجی ندارد. مدل‌سازی مسئله اما در قالب فاز Analysis از آن در قالب DFD و نمودار کلاس و در نهایت OODFD انجام می‌شود. ضمناً در فاز طراحی هم ترنزکشن‌هایی که در ابتدا استخراج می‌شوند در واقع در فضای مسئله هستند و هنوز جزئیات طراحی ندارند. نمود جدی از طراحی معماری در این متدولوژی دیده نمی‌شود ولی در فاز Design به طراحی جزئی کلاس‌ها توجه شده است. در ادامه هیچ کدام از فعالیت‌های پیاده‌سازی و تست و استقرار و نگهداری را ندارد پس پوشش به شدت ناقص است.

شباهت‌ها هر دو متدولوژی به مدل‌سازی قلمرو مسئله و طراحی جزیی می‌پردازند و هر دو متدولوژی به فعالیت‌های نگهداری و پشتیبانی و بالانگه‌داشتن نرم‌افزار و مرگ نرم‌افزار توجه ندارند.

تفاوت‌ها تحلیل امکان‌سنجی و استخراج نیازمندی و همچنین طراحی معماری و تست و پیاده‌سازی و استقرار در USDP هست که در FOOM دیده نشده است. ضمناً مراحلی که در هر دو متدولوژی هست در USDP به شکل غنی‌تری انجام می‌شود (به دلیل مدل‌سازی جامع‌تر که دربخش مدل‌سازی می‌بینیم).

سایر نکات با توجه به نکات بالا USDP به طور مطلق پوشش بهتری از FOOM دارد.

۳.۱.۴ پشتیبانی از فعالیت‌های چتری

مدیریت ریسک

محوریت Transaction ها در این متدولوژی می‌تواند باعث Continous Verification شود که ریسک را کاهش می‌دهد. اما نمودی از سایر روش‌های کاهش ریسک در آن دیده نمی‌شود. همچنین عدم توجه به استقرار فیزیکی نرم‌افزار و نکات مربوط به پیاده‌سازی پایگاه‌داده ریسک جدی پیاده‌سازی و استقرار دارد.

شباهت‌ها در هر دو متدولوژی استفاده از Continous Verification می‌تواند ریسک را کاهش دهد. در هر دو متدولوژی حضور فعال و دائمی کاربر یا نماینده‌اش دیده نمی‌شود.

تفاوت‌ها در مورد حضور کاربر درست است که در هر دو به شکل دائمی نیست ولی در کل در USDP حتی اگر دائمی نباشد ولی حضور فعال کاربر را در قالب استخراج نیازمندی و بازخورد پروتوتایپ داریم. ضمناً USDP به دلیل چرخشی‌تکراری بودن و تحلیل امکان‌سنجی اولیه برنامه‌ریزی بر مبنای ریسک، ساخت پروتوتایپ، حد قابل قبولی از Continous Validation ، حدی از CI و همچنین قابلیت استفاده از روش‌های فرمال و ورود سریع به قلمرو جواب با استخراج سریع معماری وجود دارد که در FOOM دیده نمی‌شود.

سایر نکات با توجه به نکات بالا در این معیار USDP برتری مطلق دارد.

مدیریت پروژه

در این متدولوژی اثری از فعالیت‌های مدیریت پروژه نیست.

شباهت‌ها ندارند.

تفاوت‌ها در USDP سه کار اصلی مدیریت پروژه تعریف شده است و در FOOM نیست.

سایر نکات متدولوژی USDP برتری مطلق دارد.

تضمین کیفیت

همانطور که گفتیم Transaction ها در واقع مشابه usecase قابلیت رهگیری به نیازمندی را فراهم می‌کنند و این به افزایش کیفیت کمک می‌کند اما رفتار دیگری که باعث افزایش کیفیت شود در این متدولوژی دیده نمی‌شود.

شباهت‌ها هر دو با رهگیری به نیازمندی‌ها به بهبود کیفیت کمک کرده‌اند

تفاوت‌ها در USDP چک لیست فازها را داریم و معیارهای کیفی در پایان Elaboration تعریف می‌شوند و به تست اهمیت داده می‌شود و امکان بازبینی مکرر محصولات هست و فرمالیزم دارد و همه‌ی این‌ها در بهبود و کنترل کیفیت محصولات موثرند و در FOOM ناموجود هستند.

سایر نکات متدولوژی USDP برتری مطلق دارد.

۴.۱.۴ بی‌درزی و همواری گذار

محوریت Transaction در این متدولوژی تا حدی باعث بی درزی می‌شود. ضمناً مدل غیر شی‌گرای OODFD به زور ایجاد شده است و در واقع به شکل جعلی شی گرا شده است و تغییر پارادایم است ولی خب با کمک

Transaction ها سعی شده است این پارادایم متفاوت کمتر حس شود و گذر هموار است. همچنین این متدولوژی به استخراج کلاس‌های DB نمی پردازد که در هنگام پیاده‌سازی باعث ایجاد درز می‌شود. مدل‌سازی رفتاری هم در فاز تحلیل نیست و ناگهان در فاز طراحی انجام می‌شود و می‌توان این را هم نمود درز دانست.

شباهت‌ها محوریت مفهومی سناریوی نیازمندی در هر دو متدولوژی به بی درزی کمک می‌کند. مدل‌های ساختاری و وظیفه‌ای در هر دو به آرامی از قلمرو مسئله به جواب می‌روند و درز ندارد. هر دو متدولوژی مدل غیرشی‌گرای وظیفه‌ای دارند (usecase و OODFD) و هر دو سعی کرده‌اند با اعمال تکنیک این مدل‌ها را شی‌گرا کنند و درز را بپوشانند.

تفاوت‌ها مدل‌سازی رفتاری در فاز تحلیل FOOM نیست و ناگهان در طراحی انجام می‌شود ولی در USDP از ابتدا هست و ازین نظر بی‌درزتر است. همچنین تمام کلاس‌ها و جزئیات قلمرو جواب در USDP استخراج میشود ولی در FOOM فقط کلاس‌های UI توجه شده‌اند که کافی نیست و ازین نظر در پیاده‌سازی درز خواهیم داشت و به پیکربندی فیزیکی هم در FOOM توجهی نشده که این هم باعث ایجاد درز در پیاده‌سازی و استقرار می‌شود. با توجه به درزهای پوشانده نشده بالا در FOOM و در کنارش چرخشی تکراری بودن USDP می‌توان گفت USDP از نظر همواری وضعیت بهتری دارد.

سایر نکات ـ

۵.۱.۴ میزان توجه به نیازمندی‌های وظیفه‌ای و غیروظیفه‌ای

این متدولوژی به استخراج نیازمندی‌ها نمی‌پردازد ولی نمودار کلاس اولیه و OODFD و در ادامه Transaction ها را بر اساس نیازمندی‌ها در می‌آورد بنابر این می‌توان آن را مبتنی بر نیازمندی دانست. ضمناً از زمانی که Transaction ها استخراج شدند تا انتها دیگر مدل نیازمندی همان Transaction است و همه چیزهای دیگر بر اساس آن ساخته می‌شود پس می‌توان آن را در مرحله‌ی طراحی نیازمندی‌رانه هم دانست ولی مرحله‌ی تحلیلش نیازمندی‌رانه نیست. متاسفانه امکان تکمیل و تغییر نیازمندی‌ها در این متدولوژی دیده نشده است و نمودی از نیازمندی‌های غیروظیفه‌ای هم در آن دیده نشده است.

شباهت‌ها هر دو متدولوژی در مرحله‌ی طراحی با سناریوهای استفاده از سیستم نیازمندی‌رانه هستند.

تفاوت‌ها متدولوژی FOOM در مرحله‌ی تحلیل فقط مبتنی بر نیازمندی است ولی USDP در تمامی مراحل نیازمندی‌رانه است و بهتر عمل کرده است. در USDP امکان تغییر و تکمیل نیازمندی‌ها هست و به استخراج نیازمندی‌ها هم پرداخته می‌شود و به نیازمندی‌های غیروظیفه‌ای هم توجه جدی‌تری می‌شود و در این موارد بهتر عمل کرده است. ضمناً usecase کلاً از Transaction مدل بهتری است زیرا اولاً زودتر استخراج می‌شود که باعث می‌شود در فاز تحلیل هم نیازمندی‌رانه باشیم. دوماً usecase ها حتماً اتمیک هستند ولی در مورد Transaction اتمیک بودن لزومی ندارد. سوماً توصیف usecase هیچ چیزی از درون سیستم ندارد ولی Transaction شامل فرآیندهای برگ داخل سیستم است.

سایر نکات با توجه به موارد بالا برتری در FOOM دیده نمیشود و USDP در این معیار برتری دارد.

۶.۱.۴ قابلیت تست، میزان ملموس‌بودن و قابلیت رهگیری به نیازمندی‌ها

قابلیت تست

این متدولوژی به نسبت متدولوژی‌های دیگر مطرح شده مدل‌های کمتری دارد و ساده هستند که به تست‌پذیری آن‌ها کمک می‌کند (البته این کم بودن مدل‌ها هزینه دارد و باعث ایجاد درز در پیاده‌سازی شده است) البته اگر سیستم بزرگ شود سادگی مدل‌ها از بین می‌رود و مدل‌ها مدیریت پیچیدگی خوبی ندارند که به تست‌پذیری ضربه خواهد زد. چون OODFD کاملاً شی گرا نیست ارتباط آن با سایر مدل‌ها واقعی نیست و جعلی ایجاد شده است و این ارتباط جعلی می‌تواند به تست‌پذیری محصولات ضربه بزند. ضمناً محوریت Transaction باعث Verification پیوسته می‌شود که تست را آسان می‌کند.

شباهت‌ها به دلیل محوریت داشتن سناریوهای رفتاری سیستم می‌تواند تا حدی به تست‌پذیری در هر دو متدولوژی کمک شود.

تفاوت‌ها درست است که مدل‌های FOOM کمتر هستند و برای سیستم کوچک ساده هستند که به تست‌پذیری کمک می‌کند ولی اگر سیستم بزرگ شود به دلیل عدم مدیریت پیچیدگی در مدل این سادگی از بین می‌رود ودر آنجا USDP با وجود مدل‌های بیشتر تست‌پذیری بهتری دارد. ضمناً جعلی بودن ارتباط OODFD با سایر محصولات باعث می‌شود که وابستگی محصولات دقیق مشخص نباشد و ممکن است تست‌پذیری در اثر استفاده‌ی اشتباه افراد از مدل‌ها آسیب ببیند ولی همچین مشکلی در UML وجود ندارد و ارتباط و نحوه‌ی تولید محصولات از یکدیگر مشخص است.

سایر نکات ـ

میزان ملموس‌بودن

محصولات برای کاربر ملموس هستند چون نمودار کلاس اولیه و OODFD اولیه و توصیف اولیه‌ی Transaction ها در قلمرو مسئله هستند و جزئیات جواب ندارد. برای برنامه نویس‌ها هم ملموس هستند زیرا با اضافه کردن جزئیات به آن‌ها کلاس‌های UI هم در می‌آید. البته محصولات ناقص است و کلاس‌های DB استخراج نشده که مطلوب نیست ولی به هرحال محصولاتی که تولید شده برای تولید کد مفید است. البته خود OODFD از نظر ملموس بودن زیر سوال است و در پیاده‌سازی کد مستقیم مفید واقع نمی‌شود.

شباهت‌ها بیشتر محصولات تولید شده در هر دو متدولوژی برای ذی‌نفعان قلمرو مسئله و برنامه نویسان مفید است.

تفاوت‌ها در USDP مدلی که واقعا نفعی برای برنامه نویس نداشته باشد نداریم ولی OODFD در مرحله تولید کد دور انداخته می‌شود و پیاده‌سازی بر اساس متدهای کلاس‌ها و TransactionClass که ممکن است هنوز برخی عملیات‌ها در آن پیاده‌سازی شده باشند انجام می‌گیرد. پس می‌توان گفت در USDP وضع ملموس‌بودن مدل‌ها بهتر است.

سایر نکات اگر ذی‌نفعان قلمرو مسئله افرادی باشند که DFD برایشان قابل فهم‌تر باشد (مثل مهندسین صنایع) می‌توان گفت ملموس‌بودن از نظر کاربر و مشتری در FOOM سطح بهتری دارد.

قابلیت رهگیری به نیازمندی‌ها

به دلیل اینکه پس از استخراج Transaction ها همه چیز بر اساس آن‌ها ساخته می‌شود بنابر این رهگیری مدل‌های طراحی به نیازمندی وضع خوبی دارد. البته مدل‌هایی که قبل از Transaction ساخته می‌شوند یعنی OODFD و کلاس‌های اولیه این رهگیری را ندارند.

شباهت‌ها هر دو متدولوژی به دلیل نیازمندی‌رانه بودن امکان رهگیری به نیازمندی‌ها را محقق کرده‌اند

تفاوت‌ها در USDP از ابتدا usecase در می‌آید و تمام محصولات تحت تاثیرش تولید می‌شوند ولی در FOOM به دلیل استخراج دیر Transaction محصولات تولیدی تا قبل از آن قابلیت رهگیری صریح به نیازمندی را ندارند.

سایر نکات ـ

۷.۱.۴ میزان مشارکت فعال کاربر

مشارکت فعال کاربر در این متدولوژی دیده نشده است و حتی در حد مشارکت برای استخراج نیازمندی‌ها هم وجود ندارد.

شباهت‌ها ندارند.

تفاوت‌ها کاربر و مشتری در USDP در قالب استخراج نیازمندی و بستن قرار داد و انجام Validation مکرر درگیر هستند و در FOOM هرگز وجود ندارد.

سایر نکات ـ

۸.۱.۴ قابلیت اجرا و سادگی و موثر بودن اجرا

قابلیت اجرای متدولوژی ازین نظر که فرآیند نسبتاً ساده است و تعداد محصولات تولید شده چندان زیاد نیست وضعیت خوبی دارد. البته با توجه به ضعف مدل‌سازی (مثلا عدم وجود فرمالیزم و مدل‌سازی رفتاری ضعیف و

عدم مدل‌سازی پیکربندی فیزیکی) در سیستمی که این مدل‌ها مهم باشند قابل اجرا نیست. مثلا در سیستم realtime اصلاً قابل اعمال نیست چون مدل رفتاری و timing ندارد پس قابلیت اجرایش محدود به سیستم‌های تجاری آن هم در ابعاد کوچک است زیرا سیستم‌های تجاری بزرگ به مدل‌سازی رفتاری و پیکربندی فیزیکی شدیداً نیاز دارند. به دلیل سادگی نسبی فرایند می‌تواند در موثر بودن اجرایش تاثیر بگذارد. لزوماً به ابزار هم نیاز ندارد که نکته‌ی مثبتی است. همچنین محوریت Transaction ها از یک جایی به بعد می‌تواند بین افراد تیم برای ساخت محصولات تمرکز مشترک ایجاد کند که برای موثر بودن اجرا تاثیر مثبت دارد. ولی نکات منفی هم هست. استفاده از تکنیک‌های خطا خیز در این متدولوژی دیده می‌شود که جدی‌ترین آن مسئله TransactionClass است که توصیه‌ی جدی به حذفش نشده است و می‌تواند باعث ایجاد GodClass شود و در آینده به ما آسیب بزند که موثر بودن اجرا را زیر سوال می‌برد. همچنین فعالیت مدیریت پروژه جدی گرفته نشده است و موثر بودنش را زیر سوال می‌برد. وابستگی خاصی به ابزار دیده نمی‌شود که با توجه به سادگی نسبی متدولوژی نکته‌ی مثبتی در جهت سادگی اجراست.

شباهت‌ها Transaction و usecase هر دو نوعی تمرکز ایجاد کرده‌اند که باعث موثر شدن اجرا می‌شود.

تفاوت‌ها در مورد تمرکزی که گفتیم در هر دو وجود دارد USDP بهتر عمل می‌کند اولا usecase مدل بهتری از Transaction است و زودتر استخراج می‌شود و دوما در USDP علاوه بر نیازمندی معماری‌محور هم هستیم که تمرکز اضافی ایجاد می‌کند. در USDP مدیریت پروژه را داریم که ازین نظر باعث می‌شود از FOOM موثرتر اجرا شود. تکنیک خطاخیز بزرگی در USDP نیست ولی اشاره کردیم که در متدولوژی FOOM روش‌هایی پیشنهاد شده که با اصول شی‌گرایی سازگار نیست و ازین نظر هم USDP موثرتر اجرا می‌شود. تنها مزیت FOOM از نظر قابلیت اجرا و سادگی اجرا فرآیند ساده‌تر است ولی خب این ساده‌تر بودن به قیمت مقیاس‌پذیری کمتر است ودر پروژه‌ی بزرگ USDP موثرتر اجرا می‌شود ولی در ازایش در پروژه کوچک یا متوسطی که ذی‌نفعان پروژه به DFD علاقه داشته باشند (مثل مهندسین صنایع) متدولوژی FOOM می‌تواند موثرتر اجرا شود.

سایر نکات مقایسه کلی سخت است که بگوییم کدام ساده‌تر و موثرتر اجرا می‌شود و همانطور که در بخش آخر تفاوت‌ها گفتیم بسته به اندازه پروژه و افراد درگیر ممکن است FOOM با تمام ضعف‌هایش بهتر از USDP اجرا شود.

۹.۱.۴ مدیریت پیچیدگی فرآیند

متدولوژی به دو فاز کلی تحلیل و طراحی شکسته شده است. داخل هر کدام از این فازها هم فعالیت‌های هر فاز جداگانه و به ترتیب مشخص شده‌اند و خروجی هر فعالیت هم مشخص است. برخی فعالیت‌ها مثل استخراج منوها به شکل دقیق و در قالب یک الگوریتم توضیح داده شده‌اند و پیچیدگی آن کاملاً از بین رفته است.

شباهت‌ها هر دو با تعریف فازها و قراردادن فعالیت‌ها داخل فازها سعی کرده‌اند پیچیدگی فرآیند را کنترل کنند تا فهم و اجرایش آسان‌تر شود.

تفاوت‌ها متدولوژی FOOM به شکل تکراری چرخشی نیست و کارهای هرفاز متفاوت است ولی در USDP مجموعه فعالیت‌های داخل هر فاز همیشه یکی است که باعث می‌شود در کل FOOM راحت‌تر فهمیده شود. البته جدول محصولات و Definition of done محصولات هر فاز در USDP کمک می‌کند این دشواری فهم فرایند کمتر شود.

سایر نکات ـ

۱۰.۱.۴ توسعه‌پذیری ، مقیاس‌پذیری، قابلیت پیکربندی و انعطاف‌پذیری

این متدولوژی ناقص است و در عمل برای اجرا به توسعه نیاز دارد ولی به هر حال نقاط توسعه و فعالیت‌ها و prac-tice هایی که باید اضافه شوند صریحاً مشخص نشده است و نمی‌توان گفت توسعه‌پذیری دارد.
از نظر مقیاس‌پذیری برای ساخت سیستم‌های بحرانی هیچ فرمالیزمی در آن نیست وحتی مدل‌سازی رفتاری هم ضعیف است و وضع خوبی نداریم و حداکثر می‌توان برای ساخت Discretionary Money ها از آن استفاده کرد.
از نظر مقیاس‌پذیری از جنبه‌ی سایز پروژه به معنی تعداد افراد درگیر هم وضع خوبی ندارد زیرا مدل‌سازی معماری که هماهنگی ایجاد می‌کند را ندارد و به فعالیت‌های مدیریت پروژه و افراد درگیر و روابط آن‌ها توجهی نشده است. البته همچنان مقیاس‌پذیری‌اش از متدولوژی‌هایی که هیچ مدل‌سازی ندارند بهتر است چون به هر حال مدل‌سازی در آن جدی گرفته می‌شود.
قابلیت پیکربندی در آن دیده نشده است و توصیه‌ای برای پیکربندی متدولوژی برای پروژه‌ی خاص در تعریف متدولوژی نیست.
هیچ پیش‌بینی برای اصلاح و بازبینی فرآیند در حین اجرا وجود ندارد و هیچ نمودی از انعطاف‌پذیری را در خود ندارد.

شباهت‌ها هر دو متدولوژی توسعه‌پذیری و انعطاف‌پذیری را در خود جای نداده‌اند.

تفاوت‌ها در مورد توسعه‌پذیری درست است هر دو صراحتاً تعریفی ندارند ولی در عمل برای اجرای FOOM به توسعه‌ی آن نیاز داریم ولی خب USDP به جز فاز نگهداری نسبتاً کامل است و بدون توسعه هم قابل اجراست. در مورد پیکربندی، پیکربندی USDP ممکن است و بسیار هم ضروری است که در FOOM وجود ندارد و USDP برتری دارد. از نظر مقیاس‌پذیری بحرانی بودن USDP برتری مطلق دارد زیرا اولا فرمالیزم در قالب OCL را داراست و دوماً مدل‌هایش غنی هستند و به جز مدل‌سازی رفتاری ضعیف همه جنبه‌ها را پوشش می‌دهند. از نظر مقیاس‌پذیری با تعداد افراد درگیر هم USDP برتری مطلق دارد زیرا جدای از تنوع بیشتر مدل‌ها و جدی بودن فعالیت‌های مدیریت پروژه به معماری توجه ویژه شده است که باعث هماهنگی و جهت مشترک بین افراد در تیم‌های بزرگ می‌شود.

سایر نکات ـ

۱۱.۱.۴ حیطه‌ی کاربرد

طبق گفته‌ی خود متدولوژی این متدولوژی برای سیستم‌های Data Intensive است. در عمل هم به دلیل عدم وجود فرمالیزم به سیستم‌های بحرانی کاربرد پیدا نمی‌کند. همچنین برای پروژه‌های خیلی بزرگ که تعداد افراد در آن خیلی زیاد باشد به کار نمی‌آید زیرا دید معماری متمرکز کننده در آن نیست و فعالیت مدیریت پروژه‌ای ندارد. پس فقط برای سیستم‌های اطلاعاتی تا سایز متوسط به کار می‌آید والبته در جایی که ذی‌نفعان از DFD خوششان بیاید (مثل جایی که مهندسین صنایع درگیر پروژه باشند) کاربردش جدی‌تر می‌شود.

شباهت‌ها هر دو برای ساخت شی‌گرای نرم‌افزار شی‌گرا کاربرد دارند.

تفاوت‌ها چون USDP سنگین‌تر است فقط جایی که اندازه پروژه بزرگ است کاربردش معقول است و در جایی که اندازه پروژه به معنی تعداد افراد درگیر چندان زیاد نیست استفاده از FOOM کاربرد دارد. ضمناً جایی که ذی‌نفعان از روش‌های SASD خوششان بیاید استفاده از FOOM کاربرد دارد. ضمناً اگر سیستم ریسکی و بحرانی شود دیگر FOOM به دلیل عدم فرمالیزم به کار نمی‌آید و در آنجا USDP هنوز می‌تواند به کار برود.

سایر نکات در کل با وجود سنگین‌تر بودن USDP عملاً گستره‌ی کاربرد بیشتری دارد زیرا تنها برتری FOOM استفاده از روش‌های SASD است و پر از نکات منفی است که باعث می‌شود کاربردش کم باشد. ضمناً مدل‌سازی در USDP بسیار متنوع است و حتی می‌توان سیستم‌های realtime را با UML مدل کرد و هر چند USDP بهترین متدولوژی برای ساخت چنین سیستم‌هایی نیست ولی امکانش هست ولی این کار در FOOM به کلی ممکن نیست.

۲.۴ بخش مدل‌سازی

۱.۲.۴ پشتیبانی از مدل‌سازی سازگار، دقیق و بدون ابهام شی‌گرا

مدل اصلی FOOM همان OODFD است که در کنار آن امکان تولید مستند متنی و MessageChart و نمودار کلاس هم وجود دارد.

مدل‌سازی وظیفه‌ای در قالب OODFD در آن دیده می‌شود و مدل‌سازی ساختاری هم در قالب نمودار کلاس صورت می‌گیرد. مدل‌سازی رفتاری هم در قالب MessageChart و یا PseudoCode قابل انجام است ولی در ادامه می‌بینیم که این مدل‌سازی رفتاری غنی نیست.

گذار مدل‌ها از قلمرو مسئله به جواب تا حدی هست چون در ابتدا فقط کلاس‌های قلمرو مسئله در OODFD هستند و در ادامه با استخراج فرم‌ها و منوها کلاس‌های UI اضافه می‌شوند ولی در کل این گذار ناقص است. اولاً کلاس‌های DB که جز مهم‌ترین کلاس‌های قلمرو جواب هستند استخراج و مدل نمی‌شوند و دوماً پیکربندی فیزیکی

سیستم مدل نمی‌شود و سوماً برخی مدل‌ها در قلمرو مسئله تولید نمی‌شوند به خصوص مدل‌سازی رفتاری که در فاز طراحی شروع می‌شود و در تحلیل مدل‌سازی رفتاری ندارد.

از نظر سطوح مختلف مدل‌سازی، مدل‌سازی وظیفه‌ای سازمانی با OODFD ممکن است ولی متدولوژی توصیه‌ای به انجام این‌کار ندارد و مدل‌سازی رفتاری و ساختاری سازمانی هم دیده نمی‌شود. مدل‌سازی وظیفه‌ای سطح سیستم مجددا با سطح صفرم در OODFD که مشابه DFD سازمانی است انجام می‌شود. نمودار ساختاری سطح سیستم در آن دیده نمی‌شود (البته در ClassDiagram اگر مرز سیستم مشخص باشد و کلاس‌های درون و بیرون سیستم دیده شوند می‌توان آن را نمودار ساختاری سیستمی دانست ولی متدولوژی چنین توصیه‌ای ندارد). توصیف اولیه و سطح بالای سناریوهای Transaction را شاید با خوش‌بینی بتوان نمودار رفتاری سیستمی دانست چون سناریوی تعامل مربوط به endpoint های سیستم را نشان می‌دهد ولی از سوی دیگر کاملاً هم سیستمی نیست زیرا فرآیندهای قرار گرفته در آن در سطح برگ‌های OODFD هستند پس در کل نمودار رفتاری سیستمی خوبی نیست. مدل‌سازی وظیفه‌ای برای زیر سیستم‌ها هر چند با OODFD قابل انجام است ولی متدولوژی توجه جدی به استخراج زیرسیستم‌ها ندارد و در کل دیده نشده است. مشابهاً مدل‌سازی رفتاری و وظیفه‌ای هم برای زیرسیستم‌ها دیده نشده است. ساختار بین اشیا را با نمودار کلاس مدل می‌کند و در قالب OODFD به شکل وظیفه‌ای آن را توضیح می‌دهد. مدل‌سازی رفتاری بین اشیا در قالب توصیف Transaction ها محقق می‌شود. مدل‌سازی ساختاری درون شی‌ای با همان نمودار کلاس به شکل مطلوبی محقق می‌شود. مدل‌سازی وظیفه‌ای در سطح داخل شی ندارد ولی مدل‌سازی رفتاری داخل اشیا هم خوب است و می‌توان به انتخاب از MessageChart یا PseudoCode استفاده کرد.

هیچ پشتیبانی از فرمالیزم در مدل‌سازی متدولوژی دیده نشده است.

شباهت‌ها گذار نسبی از قلمرو مسئله به قلمرو جواب در مدل‌های هر دو متدولوژی هست. مدل‌سازی وظیفه‌ای سطح سیستم در هر دو متدولوژی هست. مدل‌سازی ساختاری بین‌شی و درون‌شی در هر دو متدولوژی دیده می‌شود و مدل‌سازی رفتاری بین‌شی و درون‌شی هم در هر دو متدولوژی دیده می‌شود. مدل‌سازی سطح سازمان در هر دو متدولوژی جدی گرفته نشده است.

تفاوت‌ها هر چند در هر دو متدولوژی گذار از فضای مسئله به جواب دیده می‌شود ولی در USDP بسیار کامل‌تر است چون مثلا مدل‌سازی رفتاری قلمرو جواب در FOOM دیده نمی‌شود که در USDP هست از طرف دیگر پیکربندی فیزیکی در USDP هست ولی در FOOM وجود ندارد.

مدل‌سازی سطح زیرسیستم در USDP هم به شکل وظیفه‌ای و هم رفتاری و هم ساختاری در قلمرو مسئله و جواب قابل انجام است که در FOOM هیچ اثری از آن دیده نمی‌شود.

پشتیبانی از فرمالیزم در UML در قالب OCL هست ولی مدل‌های متدولوژی FOOM پشتیبانی از فرمالیزم ندارند.

سایر نکات با توجه به نکات بالا در کل مدل‌سازی USDP برتری و پوشش کامل‌تری از FOOM دارد. یک نکته

در مورد این معیار بحث سازگار بودن مدل‌هاست و در FOOM سعی شده است به شکل جعلی ارتباط میان نمودار OODFD که واقعاً شی‌گرا نیست با کمک Transaction ها به دنیای شی‌گرا برقرار شود و به آن نمودار رفتار اضافه شود. این کار در عین جذابیتی که دارد کار غیربدیهی است و امکان خطا در وقوع آن بسیار بیشتر از کار روش‌مند کشیدن ActivityDiagram در UML برای usecase ها و سپس کشیدن خطوط شنا در آن برای شی‌گرا کردنش است. پس مدل‌سازی UML علاوه بر اینکه کامل‌تر هست شی‌گراتر و دقیق‌تر هم هست.

۲.۲.۴ وجود راهکارهای رفع ناسازگاری و مدیریت پیچیدگی مدل‌ها

در این متدولوژی تعدد مدل‌ها وجود دارد که به ناسازگاری دامن میزند و تکنیک‌های رفع ناسازگاری در زبان مدل‌سازی که شامل تک مدله‌بودن و یا تعریف دقیق وابستگی میان مدل‌ها (شبیه آنچه Fusion انجام می‌دهد) دیده نمی‌شود. با توجه به محوریت داشتن Transaction ها این موضوع می‌تواند تا حدی به یکپارچگی مدل‌ها کمک کند ولی مشابه آنچه در مورد SequenceDiagram در USDP گفتیم این محوریت داشتن در حد یک کمک نسبی است همچنان باید برای حفظ سازگاری دستی چک کردن مدل‌ها را انجام دهیم که هزینه‌بر است.

در زمینه مدیریت پیچیدگی می‌توان در نمودار OODFD مثل DFD عادی یک سلسله مراتب لایه‌ای ساخت و از flat شدن آن جلوگیری کرد. همچنین اینکه تمام جنبه‌های سیستم در یک یا دو مدل نشان داده نمی‌شود و برای جنبه‌ی رفتاری و ساختار مدل جدا دارد هم تلاشی در جهت جلوگیری از پیچیده شدن مدل هاست.

اما در مورد نمودار کلاس این متدولوژی هیچ توصیه‌ای برای دسته‌بندی و مدیریت پیچیدگی آن‌ها در قالبی مثل package ندارد. همچنین سایر مدل‌های متدولوژی مثل MessageChart یا خود Transaction ها هم امکان دسته‌بندی و نشستن در ساختارهای سلسله مراتبی را ندارند.

شباهت‌ها هر دو متدولوژی به دلیل وجود مدل‌های جداگانه برای هر جنبه از سیستم دچار ضعف در حفظ سازگاری مدل‌ها هستند و وجود سناریوهای ارزیابی تا حدی جلوی این ناسازگاری را می‌گیرد. در هر دو متدولوژی هر جنبه از سیستم مدل جدا دارد که جلوی پیچیده شدن مدل‌ها را گرفته است. در هر دو متدولوژی روش‌هایی برای دسته‌بندی اجزای مدل‌ها و ساخت سلسله‌مراتب معماری از اجرا برای مدیریت پیچیدگی و آسان شدن فهم مدل‌ها دیده می‌شود.

تفاوت‌ها تکراری‌چرخشی بودن USDP می‌تواند کمک کند ناسازگاری مدل‌ها مدیریت و رفع شود ولی این مورد در FOOM دیده نمی‌شود.

در زمینه مدیریت پیچیدگی در USDP در مورد بیشتر نمودارها از جمله نمودارهای رفتاری و ساختاری آن هم امکان ساخت سلسله مراتب هست که در تحلیل این بخش از USDP به ان اشاره کردیم ولی در FOOM امکان دسته‌بندی اجزای ClassDiagram و یا نمودارهای رفتاری آن یعنی MessageChart دیده نشده است که USDP بهتر عمل کرده است.

سایر نکات از نظر حفظ سازگاری در کل وضعیت هردو خوب نیست ولی می‌توان گفت USDP مدیریت پیچیدگی مدل‌هایش کامل‌تر و بهتر صورت گرفته است.

۳.۴ سایر نکات

یک مشکل بزرگ متدولوژی این است که متدولوژی شی‌گرا است ولی خیلی از اصول ساده‌ی شی‌گرایی در آن رعایت نشده است و تعدادی پادالگو توصیه شده است. مثل TransactionClass که می‌تواند GodClass شود و یا توصیه شده هر کلاس مسئول نگه‌داری instance هایش باشد در حالیکه باید برای این کار کاتالوگ مجزا تعریف می‌شد. یک نکته جالب در مورد متدولوژی شباهت نحوه ساخت مدل‌های طراحی با Refactoring است که یک مدل درشت و سنگین می‌سازد و به تدریج با بیرون کشیدن رفتار و داده از آن آن را سبک می‌کند.

تفاوت‌ها متدولوژی USDP به جز usecase صد در صد شی‌گراست و مشکلی ندارد و تکنیک‌های ضد شی‌گرایی هم پیش‌نهاد نکرده است.

فصل ۵

مقایسه‌ی متدولوژی‌های USDP و Fusion

۱.۵ بخش فرآیند

۱.۱.۵ تعریف متدولوژی

متدولوژی به شکل فرآیند محور تعریف شده است و پوشش آن از چرخه ایجاد نرم‌افزار محدود به تحلیل و طراحی و پیاده‌سازی است پس نمی‌توان تعریفش را کامل دانست چون هیچ توصیه‌ای هم به نحوه گسترش به یک متدولوژی کامل ندارد. ضمناً یک نقطه از فرآیند هست که مبهم و غلط است و آن تبدیل Overall Object Model به System Object Model است و فقط در صورتی قابل انجام است که کلاس‌هایی که هم در داخل سیستم موجودند و هم خارج سیستم از قبل تشخیص داده شوند ولی متدولوژی هیچ اشاره‌ای به این موضوع نمی‌کند.

زبان مدل‌سازی خاص خود متدولوژی است و از زبان‌های متدولوژی‌های دیگر الهام گرفته است. وابستگی مدل‌ها به شکل صریح (و نه ضمنی) تعریف شده است که نکته‌ی مثبتی است. همچنین در طول فرآیند مشخص شده است که کدام مدل‌ها در کدام قسمت فرآیند ساخته می‌شوند. یک نکته‌ی منفی در مورد محصولات این است که گذار تدریجی از قلمرو مسئله به جواب کامل نیست و ویژگی‌ها و جزئیات قلمرو جواب و پیاده‌سازی مدل‌سازی نمی‌شوند و ناگهان در تولید انجام می‌شوند.

در عمده‌ی تعریف فرآیند به نقش‌های درگیر در هر فرآیند اشاره نشده است ولی در استخراج Overall Object Model توصیه شده است که از خبرگان قلمرو مسئله برای استخراجش کمک بگیریم.

از نظر فعالیت‌های چتری هر چند به صراحت تعریف و تاکید نشده‌اند ولی امکان رهگیری مستمر به نیازمندی‌ها هم در جهت کاهش ریسک و هم افزایش کیفیت تاثیر مثبت دارد.

شباهت‌ها هر دو متدولوژی فرآیندمحور تعریف شده‌اند و محصولات تولیدی در فرآیند به خوبی تعریف شده است.

تفاوت‌ها از نظر پوشش چرخه ایجاد نرم‌افزار USDP هم مراحل بیشتری را پوشش می‌دهد هم دقت بیشتری دارد و هم تکنیک‌ها و قواعد را تعریف کرده است که در Fusion که متدولوژی کوچکی است دیده نمی‌شود.

در USDP علاوه بر محصولات نقش‌های درگیر در فرآیند هم تعریف شده‌اند که در Fusion دیده نشده است. فعالیت‌های چتری کمی در Fusion هم هست ولی در USDP به شکل های متنوع و جدی‌تری هست (که در بخش فعالیت‌های چتری دقیق‌تر معرفی می‌شود). از نظر مدل‌سازی و محصولات هم USDP برتری کامل دارد چون هم جامع‌تر است و هم گذارش از قلمرو مسئله به پیاده‌سازی آهسته و کامل است و در Fusion این گونه نیست.

سایر نکات ـ

۲.۱.۵ پوشش چرخه‌ی عمومی تولید نرم‌افزار

این متدولوژی هم خود را کامل معرفی می‌کند اما فازهای Maintenace را به کلی ندارد. در فاز Definition نیازمندی‌ها را ورودی مرحله‌ی Analysis فرض می‌کند و تحلیل امکان‌سنجی هم انجام نمی‌دهد و فقط مدل‌سازی قلمرو مسئله را در فاز Analysis از متدولوژی دارد. در فاز ایجاد درباره‌ی Deployment هیچ حرفی نمی‌زند. همچنین هیچ شکلی از طراحی معماری چه در زمینه روابط المان‌ها و چه ویژگی‌های مهم قلمرو جواب در مراحل این متدولوژی دیده نمی‌شود. در مورد تست کردن پیاده‌سازی هم هیچ نسخه‌ای نمی‌پیچد. در فاز Design متدولوژی عملاً فقط به طراحی جزئی پرداخته می‌شود.

شباهت‌ها هر دو متدولوژی به مدل‌سازی قلمرو مسئله و طراحی جزئی و پیاده‌سازی می‌پردازند. هر دو متدولوژی شامل فعالیت‌های نگهداری و بالانگه‌داشتن نرم‌افزار و پشتیبانی و مرگ آن نیستند.

تفاوت‌ها در USDP به استخراج نیازمندی‌ها می‌پردازیم درحالیکه در Fusion نیازمندی‌ها در قالب سند متنی از بیرون پرتاب می‌شود. تحلیل امکان‌سنجی هم در USDP هست و در Fusion نیست. به طراحی معماری هم در USDP توجه شده است که در Fusion نیست. به تست و استقرار هم در USDP توجه شده است که در Fusion وجود ندارد. در بخش‌هایی که پوشش مشترک است هم عملاً USDP کامل‌تر است مثلا در طراحی جزئی در USDP کلاس‌های قلمرو جواب استخراج می‌شوند که در Fusion این کار صورت نمی‌گیرد.

سایر نکات با توجه به نکات بالا پوشش USDP به شکل مطلق از Fusion بهتر است.

۳.۱.۵ پشتیبانی از فعالیت‌های چتری

مدیریت ریسک

در این متدولوژی رهگیری مکرر به نیازمندی در قالب TransactionScenario هست که می‌تواند Conti-nousVerification و حدی از ContinousValidation ایجاد کند که ریسک را کاهش می‌دهد. همچنین امکان

اعمال فرمالیزم را دارد که ریسک ساخت سیستم‌های بحرانی را کاهش می‌دهد. وجود چک لیست‌هایی که در انتهای فاز تحلیل مدل‌ها با آن بررسی شوند هم می تواند به کاهش ریسک بینجامد.

شباهت‌ها هر دو متدولوژی با رهگیری مکرر به نیازمندی ریسک را کاهش داده‌اند. هر دو متدولوژی با انجام یک چک لیست در پایان فازها کیفیت و صحت محصولات را می‌سنجند که به کاهش ریسک می‌انجامد. هر دو متدولوژی از فرمالیزم نسبی برخوردارند که به کاهش ریسک می‌انجامد.

تفاوت‌ها در USDP حضور نسبی کاربر، چرخشی‌تکراری بودن، برنامه‌ریزی بر مبنای ریسک، توجه جدا به فعالیت تست، حدی از CI، تحلیل امکان‌سنجی اولیه و استفاده از پروتوتایپ و ورود سریع به قلمرو جواب برای مدیریت ریسک هست که در Fusion نیست.

سایر نکات در این معیار USDP برتری مطلق دارد.

مدیریت پروژه

در این متدولوژی اثری از فعالیت‌های مدیریت پروژه نیست.

شباهت‌ها ندارند.

تفاوت‌ها در USDP سه کار اصلی مدیریت پروژه تعریف شده است و در Fusion نیست.

سایر نکات متدولوژی USDP برتری مطلق دارد.

تضمین کیفیت

یک نکته‌ی مثبت کنترل کیفیت که در Fusion دیده می‌شود این است که مدل‌ها با همدیگر تکمیل می‌شوند و امکان Verification بین مدل‌ها وجود دارد. (برای مثال اگر یک فراخوانی در Object Interaction Diagram وجود دارد باید حتماً دید مورد نیاز برای این فراخوانی در Visibility Graph وجود داشته باشد.) ضمناًاز طریق چک لیست‌ها در مرحله‌ی تحلیل صحت مدل‌های تحلیل سنجیده می‌شود که عمل Verification را روش‌مند می‌کند. مهم‌تر از همه‌ی این‌ها وجود Transaction Scenraio هاست که تا انتها برای صحت سنجی مدل‌های ساخته شده استفاده می‌شود و باعث ردگیری قوی به نیازمندی‌ها و Verficiation تمام محصولات با نیازمندی‌ها می‌شود که می‌تواند کیفیت را انجام دهد. همچنین در Operation Schema ها امکان تعریف پیش‌شرط و پس‌شرط هست که حدی از فرمالیزم است و کیفیت را افزایش می‌دهد. اعمال چک لیست بر محصولات در پایان فاز تحلیل هم بر تضمین کیفیت محصولات موثر است.

شباهت‌ها هر دو متدولوژی رهگیری به نیازمندی‌ها را با سناریوهای استفاده از سیستم دارند که کیفیت را افزایش می‌دهد. هر دو متدولوژی حدی از فرمالیزم را دارند که به بهبود کیفیت می‌انجامد. هر دو متدولوژی چک‌لیست محصولات در انتهای فاز دارند که به حفظ کیفیت کمک می‌کنند.

تفاوت‌ها در USDP امکان بازبینی مکرر محصولات هست و فعالیت جداگانه تست تعریف شده است و معیارهای کیفی خاص پروژه در Elaboration قابل تعریف است و این‌ها در Fusion دیده نشده‌اند.

سایر نکات می‌توان گفت USDP برتری مطلق دارد.

۴.۱.۵ بی‌درزی و همواری گذار

تمامی مدل‌های این متدولوژی شی‌گرا هستند و از این نظر درز پارادایم وجود ندارد. از نظر همواری هم متدولوژی در مراحل اولیه هموار است. زیرا هر چند در مرحله‌ی طراحی مدل‌های جدیدی ایجاد می‌شوند اما این مدل‌ها تحت تاثیر مدل‌های تحلیل (و دیگر مدل‌های طراحی) ایجاد می‌شوند و از صفر ایجاد نمی‌شوند. ضمناً Transaction Scenario ها هر چند در حد usecase در UP محوریت اصلی نیستند، ولی در واقع معیار اصلی ساخت و صحت سنجی تمام مدل‌های تحلیل و طراحی و پیاده‌سازی هستند و وجود این محوریت باعث می‌شود درز زیادی بین فازهای متدولوژی حس نشود. اما در مرحله‌ی طراحی به کد درز و همچنین ناهمواری عجیبی دیده می‌شود. به دلیل دیده نشدن کلاس‌ها و جزئیات قلمرو جواب در طراحی تمامی آن‌ها به عهده‌ی برنامه‌نویسان گذاشته شده که هم مبهم است و مدل‌ها درز دارند و هم گذار را سخت می‌کند زیرا می‌دانیم که حجم کل کلاس‌ها سه تا چهار برابر تعداد کلاس‌های تحلیل است و استخراج تمام این کلاس‌های اضافی و ارتباط آن با کلاس‌های قبلی به عهده‌ی برنامه‌نویس گذاشته شده است. یک درز دیگری هم که دیده می‌شود عدم وجود مدل کوچکتر از سطح سیستم در مرحله‌ی تحلیل است که کمی به همواری بین دو فاز آسیب می‌زند و در فاز دوم باید ناگهان مدل‌های طراحی در سطح شی را از مدل های تحلیل سطح سیستم بسازیم.

شباهت‌ها هر دو متدولوژی محوریت مفهومی سناریوهای نیازمندی را دارند که به بی‌درزی کمک می‌کنند. در هر دو متدولوژی با وجود تولید مدل‌های جدید در برخی مراحل این مدل‌ها از صفر تولید نمی‌شوند و حاصل افزایش دقت مدل‌های قبلی هستند که باعث می‌شود همواری آسیب جدی نبیند. گذار قابل قبولی از مدل‌های ساختاری از تحلیل به طراحی در هر دو متدولوژی هست.

تفاوت‌ها در Fusion به کلی کلاس‌ها وجزئیات قلمرو جواب استخراج نمی‌شوند و درز بزرگی بین طراحی و پیاده‌سازی هست ولی این درز در USDP وجود ندارد. همچنین مدل‌های رفتاری هم در Fusion پس از مطرح شدن در فاز تحلیل در طراحی تکمیل نمی‌شوند که مصداق درز است ولی در USDP مدل‌های رفتاری از ابتدا تا انتها با ما هستند. عدم مدل‌سازی پیکربندی فیزیکی سیستم هم در Fusion باعث درز در پیاده‌سازی است و این مشکل در USDP وجود ندارد. با توجه به درزهای پوشانده نشده‌ی بالا در USDP و همچنین چرخشی‌تکراری بودن USDP می‌توان گفت USDP با وجود مدل‌های بیشتر همواری بهتری از Fusion دارد.

سایر نکات ـ

۵.۱.۵ میزان توجه به نیازمندی‌های وظیفه‌ای و غیروظیفه‌ای

این متدولوژی نیازمندی‌ها را چیزی می‌بیند که از سمت کاربر به ما پرتاب می‌شود و هیچ فعالیتی برای استخراج و ثبت آن‌ها ندارد. مدل صریحی برای نشان دادن اولیه‌ی نیازمندی‌ها ندارد و نیازمندی‌ها در قالب یک سند متنی بدون ساختار هستند ولی در ادامه نیازمندی‌ها را از یک سند متنی به Transaction Scenario ها تبدیل می‌کند. این سناریوها در ادامه برای انجام صحت‌سنجی مدل‌ها استفاده می‌شوند بنابر این این متدولوژی تا حدی نیازمندی‌رانه هست. این متدولوژی هیچ پیش‌بینی برای تغییر یا ایجاد نیازمندی‌های جدید ندیده است و به نیازمندی‌های غیروظیفه‌ای هم هیچ اشاره‌ای ندارد (پس هر چه به عنوان نیازمندی می‌شناسد همان نیازمندی‌های وظیفه‌ای هستند و نیازمندی‌رانه بودن آن در حد نیازمندی‌های وظیفه‌ای است).

شباهت‌ها هر دو متدولوژی نیازمندی‌رانه هستند.

تفاوت‌ها توجه داریم که کلاس‌ها در USDP بعد از استخراج Usecase ها در می‌آیند و می‌توان گفت استخراج کلاس‌ها هم در USDP نیازمندی‌رانه هست ولی در Fusion قبل از استخراج سناریوها کلاس‌ها استخراج می‌شوند پس آن بخش را صرفاً مبتنی بر نیازمندی‌ها انجام می‌دهیم. ضمناً در USDP به استخراج نیازمندی‌ها و تغییر نیازمندی‌ها و نیازمندی غیروظیفه‌ای هم توجه شده است که از Fusion که این‌ها را ندارد بهتر عمل کرده است.

سایر نکات متدولوژی USDP برتری مطلق دارد.

۶.۱.۵ قابلیت تست، میزان ملموس‌بودن و قابلیت رهگیری به نیازمندی‌ها

قابلیت تست

در این متدولوژی تعدد مدل‌ها به دلیل مدل‌سازی هر جنبه در نموداری جدا را داریم که به تست‌پذیری آسیب می‌زند. ولی اطلاعات تکرار شونده در مدل‌ها نداریم. به علاوه وابستگی بین مدل‌ها صراحتاً (و نه به شکل ضمنی در فرآیند) تعریف شده که به تست‌پذیری می‌کند. البته در خود مدل‌ها مکانیزم‌های مدیریت پیچیدگی مدل وجود ندارد و اگر سایز پروژه بزرگ شود مدل‌ها پیچیده می‌شوند که تست‌پذیری آن‌ها را آسیب می‌زند.

شباهت‌ها هر دو به دلیل تعدد مدل مشکل تست‌پذیری دارند.

تفاوت‌ها در Fusion وابستگی مدل‌ها صریح در نموداری تعریف شده و اگرچه این وابستگی در USDP هم هست ولی به شکل ضمنی در فرآیند است که Fusion بهتر عمل کرده. تعدد مدل‌ها در Fusion هست ولی خب به اندازه‌ی USDP تعدد مدل ندارد. با این وجود در USDP مکانیزم‌های مدیریت پیچیدگی مدل‌ها وقتی بزرگ شوند کامل است و این مکانیزم‌ها در Fusion نیست که به تست‌پذیری مدل‌ها وقتی پروژه بزرگ شود و مدل‌ها هم بزرگ شوند آسیب می‌زند.

سایر نکات ـ

میزان ملموس‌بودن

در مدل‌های تحلیل این متدولوژی مدل پیچیده و به دردنخور و غیرقابل فهم برای مشتریان و ذی‌نفعان غیر متخصص وجود ندارد و ازین نظر ملموس است. در واقع این متدولوژی اولین متدولوژی بود که دید کاربر را مهم دانست و مدل‌های تحلیل آن با تاکید بر این اهمیت ساخته می‌شوند. از نظر برنامه نویسان هم تمامی مدل‌ها برای تولید کد شی‌گرا به کار می‌آیند و ملموس هستند. البته این نکته که جزئیات قلمرو جواب اضافه نمی‌شود کار برنامه نویسان را سخت می‌کند ولی نمی‌توان گفت ناملموس است و همچنان مدل‌های ساخته شده برای تولید کد مفید هستند.

شباهت‌ها هر دو متدولوژی مدل‌هایی می‌سازند که برای برنامه‌نویسان و ذی‌نفعان مفید است.

تفاوت‌ها مکانیزم‌های مدیریت پیچیدگی در USDP کامل‌تر است و مثلا برای نمودار کلاس PackageDiagram را داریم که می‌توان مدل‌ها را برای مشتری قابل فهم‌تر کند و از پیچیدگی آن‌ها جلوگیری کند ولی این مکانیزم‌ها در Fusion وجود ندارند. ضمناً مدل‌های مربوط به قلمرو جواب چون در USDP کامل‌تر است و جزئیات قلمرو جواب را دارد می‌توان گفت برای برنامه‌نویسان هم ملموس‌تر خواهد بود.

سایر نکات ـ

قابلیت رهگیری به نیازمندی‌ها

به دلیل وجود Transaction Scenario ها و نیازمندی‌رانه بودن امکان رهگیری مدل‌ها به نیازمندی‌ها هم آسان است.

شباهت‌ها هر دو متدولوژی به دلیل نیازمندی‌رانه بودن امکان رهگیری به نیازمندی‌ها را محقق کرده‌اند

تفاوت‌ها مشابه آن‌چه در مورد Transaction در FOOM گفتیم اینجا هم usecase مدل بهتری از TransactionScenario است.

سایر نکات ـ

۷.۱.۵ میزان مشارکت فعال کاربر

در این متدولوژی در استخراج نیازمندی‌ها کاربر و مشتری درگیر نمی‌شوند و نیازمندی در قالب سند متنی می‌آید ولی اشاره شده است که برای شناسایی کلاس‌ها در OverallObjectModel از متخصصان قلمرو مسئله یا Domain Expert ها استفاده شود.

گفتیم که این متدولوژی تاکید دارد که مدل‌سازی‌ها از دید کاربر باشد ولی این لزوماً به معنای مصاحبه و ارتباط با کاربران نیست و صرفاً می‌گوید مدل‌های تحلیل باید از دید کاربر ساخته شود.

شباهت‌ها برای بخشی از مدل‌ها در هر دو متدولوژی از متخصصان قلمرو مسئله استفاده می‌شود.

تفاوت‌ها در Fusion فقط برای شناسایی کلاس‌ها از ذی‌نفعان خارج از تیم پروژه استفاده می‌شود ولی در USDP علاوه بر آن برای استخراج نیازمندی‌ها و بازخورد گرفتن در قالب prototype ها هم استفاده شده است و USDP برتری دارد. علاوه بر این‌ها USDP چرخشی تکراری هم هست که فرصت خوبی برای جلسات مکرر با کاربر و مشتری فراهم می‌کند.

سایر نکات

۸.۱.۵ قابلیت اجرا و سادگی و موثر بودن اجرا

فرآیند متدولوژی نسبتاً ساده‌است و تعداد کارهایی که باید انجام شوند آنقدری زیاد نیست که به قابلیت اجرا و سادگی اجرا کمک می‌کنند. وجود Transaction Scenario ها تا حدی باعث ایجاد تمرکز می‌شود و به اجرای موثر متدولوژی کمک می‌کنند. وابستگی ابزاری هم نداریم که از هزینه‌ها می‌کاهد و می‌تواند باعث سادگی اجرا شود. به جز اینکه استخراج نیازمندی‌ها از کاربر را به شکل جدی ندارد تکنیک خطاخیزی در آن دیده نمی‌شود. فعالیت‌های مدیریت پروژه هم در آن وجود ندارد که اجرای موثر متدولوژی را از بین می‌برد.

شباهت‌ها هر دو با محوریت قرار دادن نیازمندی در قالب usecase و Transaction Scenario بین افراد تیم تمرکز مشترک ایجاد میکنند که باعث سهولت و موثربودن اجرا می‌شود.

تفاوت‌ها در USDP به مدیریت پروژه توجه ویژه شده است و در Fusion موجود نیست که به اجرای موثرتر USDP منجر می‌شود. همچنین در USDP علاوه بر نیازمندی معماری هم در قالب Architectural Baseline تمرکز ایجاد می‌کند و به اجرای موثر کمک می‌کند که در Fusion دیده نمی‌شود. ضمناً USDP به مدیریت ریسک اهمیت می‌دهد و به طور مثال برای استخراج نیازمندی مکرر با کاربر تماس دارد ولی در Fusion از روی سند متنی نیازمندی در می‌آورد که روش خطاخیزی است و اجرای موثر را زیر سوال می‌برد. متدولوژی USDP عملا برای استفاده به پیکربندی نیاز دارد ولی Fusion به همین شکل هم قابل اجراست که نکته‌ی مثبتی برای Fusion است. یک نکته‌ی مثبت Fusion این است که مدل‌هایش به اندازه USDP زیاد نیستند و لزوماً به ابزار نیاز ندارد که قابلیت اجرایش را بالا می‌برد.

سایر نکات در کل از نظر قابلیت اجرا برای پروژه‌های کوچک یا متوسط Fusion بهتر است زیرا USDP نیاز به شخصی‌سازی دارد که به صرفه نیست. ولی به دلیل ندیدن فعالیت‌های مدیریت پروژه و نداشتن فعالیت‌های مهمی مثل تست و استقرار و تحلیل امکان‌سنجی اولیه در پروژه‌های بزرگ و جدی Fusion قابل اجرا نیست و آنجا USDP است که قابلیت اجرا دارد. در زمینه سادگی اجرا هم همین حرف را می‌توان زد و در پروژه کوچک Fusion به دلیل فرآیند و مدل‌های سبک‌تر راحت‌تر اجرا می‌شود ولی مقیاس‌پذیر نیست و در پروژه‌های بزرگ اتفاقا به دلیل نداشتن مدیریت پروژه و موارد دیگری که اشاره کردیم اصلاً موثر اجرا نمی‌شود ودر آنجا USDP بهتر عمل می‌کند.

۹.۱.۵ مدیریت پیچیدگی فرآیند

این متدولوژی خود را در قالب ۳ فاز تحلیل ـ طراحی و کد (پیاده‌سازی) تقسیم کرده است که نوعی لایه‌بندی است. در هر فاز هم کارهای ریزدانه برای ساخت مدل‌ها را جداگانه تعریف کرده است که تکه‌تکه‌سازی است.

شباهت‌ها هر دو با تعریف فازها و قراردادن فعالیت‌ها داخل فازها سعی کرده‌اند پیچیدگی فرآیند را کنترل کنند تا فهم و اجرایش آسان‌تر شود.

تفاوت‌ها متدولوژی Fusion به شکل تکراری چرخشی نیست و کارهای هرفاز متفاوت است ولی در USDP مجموعه فعالیت‌های داخل هر فاز همیشه یکی است که باعث می‌شود در کل Fusion راحت‌تر فهمیده شود. البته جدول محصولات و Definition of done محصولات هر فاز در USDP کمک می‌کند این دشواری فهم فرایند کمتر شود.

سایر نکات ـ

۱۰.۱.۵ توسعه‌پذیری ، مقیاس‌پذیری، قابلیت پیکربندی و انعطاف‌پذیری

در Fusion نقاط توسعه و مواردی که باید توسعه داده شوند صراحتاً مطرح نمی‌شود پس نمی‌توان گفت توسعه‌پذیر است. البته چون متدولوژی پوشش کامل ندارد در عمل هنگام استفاده باید توسعه داده شود تا تست و استقرار و نگهداری به آن اضافه شود.

مقیاس‌پذیری‌اش از نظر میزان بحرانی بودن نسبتاً مناسب است و در قالب Operation Schema ها و همچنین Life Cycle Model نوعی فرمالیزم را می‌توان دید. البته این در حدی نیست که نرم‌افزار Life Critical بسازیم زیرا در آن سطح دیگر فرمالیزم کامل می‌خواهیم. مقیاس‌پذیری از نظر اندازه پروژه (تعداد افراد درگیر) چندان مناسب نیست. زیرا مدل معماری و معماری جدی گرفته نشده‌اند و فعالیت‌های مدیریت پروژه‌ای که در پروژه با تعداد افراد درگیر بالا لازم است تعریف نشده‌اند. البته به هر حال مدل‌سازی دارد و مدل‌ها متنوع هستند که اجازه می‌دهد هر کس مدل مجزا بسازد به طور موازی و به مقیاس‌پذیری کمک می‌کند اما آن دید معماری واحد که به همه جهت مشترک بدهد وجود ندارد.

قابلیت پیکربندی ندارد و پیشبینی برای تنظیم متدولوژی با پروژه تعبیه نشده است. قابلیت انعطاف و بازبینی فرآیند در حین اجرا هم ندارد.

شباهت‌ها هر دو متدولوژی توسعه‌پذیری و انعطاف‌پذیری را در خود جای نداده‌اند. از نظر مقیاس‌پذیری بحرانی بودن در هر دو متدولوژی وضع خوبی است و فرمالیزم نسبی پشتیبانی می‌شود.

تفاوت‌ها در مورد توسعه‌پذیری درست است هر دو صراحتاً تعریفی ندارند ولی در عمل برای اجرای Fusion به توسعه‌ی آن نیاز داریم ولی خب USDP به جز فاز نگهداری نسبتاً کامل است و بدون توسعه هم قابل اجراست. در مورد پیکربندی، پیکربندی USDP ممکن است و بسیار هم ضروری است که در Fusion

وجود ندارد و USDP برتری دارد. از نظر مقیاس‌پذیری با سایز پروژه به معنی افراد درگیر USDP برتری مطلق دارد چون مدل‌ها متنوع‌تر هستند که افراد بیشتری درگیر می‌شوند و با معماری‌محور بودن به همه افراد جهت مشترک میدهد و با مدیریت پروژه‌ی جدی افراد و کارهایشان را ساماندهی می‌کند که در Fusion نیست و مقیاس‌پذیری‌اش از نظر اندازه زیر سوال است. از نظر پیکربندی هم هیچ پیش‌بینی در Fusion نیست ولی همانطور که در مقایسه‌ی قبل با FOOM هم گفتیم USDP نه تنها توصیه به پیکربندی دارد بلکه عملا واجب هم هست.

سایر نکات ـ

۱۱.۱.۵ حیطه‌ی کاربرد

این متدولوژی به عنوان یک متدولوژی عمومی برای تحلیل و طراحی شی‌گرای سیستم‌ها مطرح شده است. به دلیل فرمالیزم نسبی‌ای که در آن هست می‌تواند در حوزه‌های نسبتاً بحرانی هم استفاده شود (البته آنقدری قوی نیست که بخواهیم با آن سیستم Life Critical بسازیم زیرا فرمالیزم آن محدود است). ضمناً مدل‌هایش ضعیف هستند و مثلاً نمی‌توان با آن سیستم realtime ساخت چون مدل‌های لازم برای آن سیستم‌ها را ندارد پس بیشتر برای ساخت سیستم‌های اطلاعاتی تجاری مفید است. از نظر تعداد افراد درگیر در پروژه هم کاربردش محدود است چون مدل‌های معماری را جدی نمی‌گیرد و مدیریت پروژه هم ندارد و اگر تیم بزرگ شود دیگر قابل اعمال نیست.

شباهت‌ها هر دو برای ساخت شی‌گرای نرم‌افزار شی‌گرا کاربرد دارند. ضمناً هر دو حدی از فرمالیزم را پشتیبانی می‌کنند که می‌تواند برای ساخت سیستم‌های بحرانی مفید واقع شود و در آن حوزه هم کاربرد دارند.

تفاوت‌ها چون USDP سنگین‌تر است فقط جایی که اندازه پروژه بزرگ است کاربردش معقول است و در جایی که اندازه پروژه به معنی تعداد افراد درگیر چندان زیاد نیست استفاده از Fusion کاربرد دارد.

سایر نکات در کل می‌توان گفت USDP حیطه‌ی کاربرد گسترده‌تری دارد. در جایی که پروژه کوچک باشد اعمال USDP منطقی نیست ولی به دلیل تنوع بیشتر مدل‌سازی و در نظر گرفتن همه فعالیت‌ها به جز نگهداری نرم‌افزار، طیف گسترده‌تری از انواع مختلف سیستم‌ها را می‌توان با USDP ساخت.

۲.۵ بخش مدل‌سازی

۱.۲.۵ پشتیبانی از مدل‌سازی سازگار، دقیق و بدون ابهام شی‌گرا

مدل سازی ساختاری در قالب System Object Model و Visibility Graph و Inheritance Graph و Class Description دارد. مدل‌سازی رفتاری در قالب System Lifecycle Model و Object Interaction Graph دارد. مدل Operation Schema را می‌توان بیشتر وظیفه‌ای دانست زیرا نه ساختار چیزی را نشان می‌دهد

و نه ترتیب رفتار را نشان می‌دهد بلکه خود رفتار را توصیف می‌کند. ضمناً مدل دورانداختنی System Interface Diagram هم وظیفه‌ای است زیرا ترتیبی در آن وجود ندارد. همچنین Transaction Scenario مدلی رفتاری است. پس هر سه نوع مدل‌سازی در این متدولوژی دیده شده است (البته هر سه نوع در همه سطوح تجرید در دسترس نیستند).

گذار منطقی از تحلیل به طراحی در این متدولوژی دیده نمی‌شود و در واقع کاری که در طراحی‌اش انجام می‌دهد امروزه در تحلیل هم انجام می‌شود. مشخص نیست کجا کلاس‌ها و جزئیات قلمرو جواب به کلاس‌ها اضافه می شود و همه‌ی این ها به مرحله‌ی Coding سپرده شده که یک ضعف است.

از نظر سطوح مختلف مدل‌سازی سازمانی در هیچ کدام از سه نوع در متدولوژی دیده نمی‌شود. مدل‌سازی سطح سیستمی به شکل ساختاری در SystemObjectModel دیده می‌شود. نمودار SystemInterfaceDiagram هم یک مدل وظیفه‌ای سطح سیستمی است و SystemLifecycleModel یک مدل رفتاری سطح سیستم است. مدلی برای مدل‌سازی زیرسیستم‌ها و یا روابط آن‌ها در متدولوژی پیش‌نهاد نشده است. مدل سازی ساختاری بین اشیا با کمک VisibilityGraph و همچنین InheritanceGraph صورت می‌گیرد. مدل‌سازی رفتاری بین اشیا هم با کمک TransactionScenario ها انجام می‌شود. مدل‌سازی ساختار درون اشیا با Class-Description انجام می‌شود ولی مدل‌سازی رفتاری درون شی‌ای پیشنهاد نشده است.

با تعریف پیش‌شرط‌ها و پس‌شرط‌ها در OperationSchema و همچنین امکان تعریف ترتیب انجام عملیات سیستم در LifeCycleModel پشتیبانی نسبی از فرمالیزم وجود دارد که البته این فرمالیزم فقط در سطح سیستم است و به سطح اشیا یا زیرسیستم‌ها منتقل نمی‌شود که یک ضعف است.

شباهت‌ها در هر دو متدولوژی هر سه نوع مدل‌سازی رفتاری و ساختاری و وظیفه‌ای هست. در هر دو متدولوژی پشتیبانی از فرمالیزم تا حدی هست. مدل‌سازی سطح سیستم در هر سه نوعش در هر دو متدولوژی امکان‌پذیر است. امکان مدل‌سازی رفتاری و ساختاری بین اشیا در هر دو متدولوژی هست. امکان مدل‌سازی ساختاری درون اشیا در هر دو متدولوژی هست.

تفاوت‌ها گذار مدل‌ها از فضای مسئله به فضای جواب و پیاده‌سازی به شکل مطلوب در USDP هست ولی در Fusion عملاً کلاس‌های قلمرو جواب به مدل‌ها اضافه نمی‌شوند و همه آن‌ها به عهده برنامه‌نویسان در فاز Coding گذاشته شده است. مدل‌سازی زیرسیستم‌ها در هر سه نوع در USDP به شکل مطلوب وجود دارد ولی در Fusion هیچ نوعی مدل‌سازی زیرسیستم محیا نشده است. مدل‌سازی رفتاری درون اشیا (چه در قالب نمودار حالت و چه در قالب توصیف بدنه‌ی متدها با ActivityDiagram) در USDP هست ولی در Fusion دیده نشده است. فرمالیزمی که در UML هست در تمام سطوح قابل اعمال است ولی فرمالیزم Fusion محدود به عملیات‌ها با درشت دانگی سیستمی است.

سایر نکات ـ

۲.۲.۵ وجود راهکارهای رفع ناسازگاری و مدیریت پیچیدگی مدل‌ها

در این متدولوژی تعدد مدل‌ها وجود دارد که ناسازگاری آسیب می‌زند ولی وابستگی میان مدل‌ها به شکل دقیق و صریح (و نه به شکل ضمنی در قالب فرآیند) تعریف شده است که کار را برای چک کردن سازگاری آسان می‌کند. همچنین یک چک لیست در پایان هر فاز اعمال می‌شود که می‌تواند جلوی ناسازگاری مدل‌ها را بگیرد و در آخر محوریت داشتن TransactionScenario می‌تواند تمرکز ایجاد کند و جلوی ناسازگاری را تا حدی بگیرد.

این متدولوژی با تعدد مدل حفظ ناسازگاری را سخت کرده ولی در ازایش هر جنبه از نرم‌افزار در مدل جداگانه‌ای نمایش داده می‌شود که فهم آن را ساده می‌کند از شلوغی و پیچیدگی مدل جلوگیری می‌کند. امکان نمایش Collection ها در نمودارهای این متدولوژی هم به سبک‌تر شدن مدل نمایشی و شلوغ نشدن آن کمک کرده است.

شباهت‌ها هر دو متدولوژی به دلیل تعدد مدل از مشکل ناسازگاری مدل رنج می‌برند. هر دو متدولوژی با داشتن سناریوهای نیازمندی و ایجاد تمرکز تا حدی به جلوگیری از ناسازگاری مدل‌ها کمک می‌کنند. در هر دو متدولوژی جنبه‌های مختلف سیستم در مدل‌های مختلف نمایش داده می‌شود تا پیچیده نشوند. در هر دو متدولوژی مکانیزم‌های دسته‌بندی و ایجاد سلسله‌مراتب از اجزای مدل‌ها (مثل package در USDP و Collection در Fusion) هست.

تفاوت‌ها یک نکته مثبت در جهت رفع ناسازگاری که در Fusion هست ودر USDP نیست نشان دادن صریح روابط مدل‌ها در یک نمودار وابستگی مدل‌ها است. از طرف دیگر چرخشی‌تکراری بودن USDP می‌تواند باعث اصلاح تدریجی ناسازگاری‌ها شود که در Fusion نیست. برای مدیریت و دسته‌بندی و ایجاد سلسله مراتب از اجزای مدل‌ها در UML خیلی وضع بهتری داریم و نمودارهای Pakcage و Component و همچنین InteractionOverview را داریم و در Fusion تنها مصداق دیده شده Collection ها هستند پس مدیریت پیچیدگی مدل‌ها در قالب لایه‌بندی در USDP خیلی قوی‌تر انجام شده است که البته با حجم بزرگ‌تر مدل‌ها ضروری هم هست.

سایر نکات ـ