بسمه تعالی

محمدرضا سابقی ۹۹۲۰۱۴۲۱

متدولوژی ایجاد نرم افزار

استاد: دکتر رامسین

تمرین اول

پاییز ۱۳۹۹

فهرست مطالب

۱ مقایسه دو متدولوژی BON و Booch

۱.۱ معیارهای فرآیندی

۱.۱.۱ تعریف

این معیار، نیازمندی ای که مطرح می کند این است که متدولوژی مدنظر باید به خوبی تعریف شده باشد(تعریفش به خوبی مستند شده باشد). یک توصیف خوب صفات جامع، روشن، منطقی، صحیح و دقیق، با ریز جزئیات و بدون تناقض را به همراه دارد. حال برای رسیدن به درک درستی از آن باید مفاهیم زیر ثبت شود:

- **چرخه عمر[۱] و واحدهای کاری[۲]**

در BON: این متدولوژی تشریح می کند که از ۳ فاز اصلی(در بالاترین سطح کاری) تشکیل شده است که هر فاز، ۳ گام یا وظیفه[۳] دارد که در مجموع این متدولوژی تعداد ۹ وظیفه را در بر می گیرد؛ تعداد ۶ تا برای فاز تحلیل و تعداد ۳ تا تمرکز بر طراحی دارند. متدولوژی اجازه می دهد که تقدم_تاخر ها در گام های ۹ گانه به شرط رسیدن به هدف پروژه، به درستی رعایت نشود، اما سرانجام همه ی انواع مدل های گفته شده باید تولید شوند. متدولوژی BON به خوبی می تواند در این جا از توصیف چرخه‌ی عمر بربیاید و واحد ها کاری را به ریز تشریح کند. تحلیل، طراحی را تا حد مطلوبی پوشش می دهد و بعد می گوید که داخل هر کدام دقیقا چه کارهایی انجام می شود و این نکته قابل تامل است که نامی از فاز دیگری ذکر نمی شود؛ اما این مربوط به معیاری دیگر می شود. درنهایت می توان این نتیجه را گرفت که دقت و جزییات بالایی از فرآیند را این متدولوژی هم برای فاز تحلیل و ۶ گام آن و هم برای فاز طراحی و ۳ گام آن

[۱] lifecycle
[۲] work-units
[۳] task

توصیف می کند(جدول ۳۸ و ستون اول و دوم در مقاله ی Paige-Ramsin که شمای منسجم و کلی از چرخه عمر و واحد های کاری به تصویر می کشد، شاهدی بر این مدعاست)

در Booch: به نظر می آید که متدولوژی Booch نیز در تعریف چرخه ی عمر خود بسیار غنی عمل کرده است. ۵ فاز اصلی درشت دانه خود که به صورت متوالی تکرارشونده ای به اجرا در می آیند و به آن ها فرآیند های Macro می گوید، تیتر گویایی را به همراه خود دارند(اشاره به بند ۴.۴.۱ و نمودار ۱۳ مقاله ی Paige-Ramsin). از طرفی داخل هر چرخه ی عمر، به صورت مفصلی واحد کاری های تکرار شونده ذکر شده اند که ۴ گام می باشند و آن ها را فرآیند های Micro می نامد(اشاره به بند ۴.۴.۲ و تبیین ۴ گام فرآیند Micro در نمودار ۱۴ مقاله Paige-Ramsin). با این تفاسیر، می توان گفت که متدولوژی از نظر بیان دقیق چرخه ی عمر و واحدهای کاری بسیار تبیین درستی داشته است.

لذا به نظر می رسد دو متدولوژی از این نظر، قوی ظاهر شده اند.

- **تولیدکنندگان**[۴]

در BON: این متدولوژی قادر به پاسخگویی به این سوال نیست که در فرآیند توسعه، چه افرادی در چه نقش هایی باید کار کنند و محصولات مدنظر را تولید کنند. در هیچ جای مقالات، این تصریح بیان نشده است. لذا از این نظر متدولوژی ضعف دارد.

در Booch: در متدولوژی Booch نیز اثری از اینکه چه افراد و نقش هایی درگیر در وظایف هستند یافت نمی شود که از این جهت مانند BON می باشد.

- **زبان مدلسازی**[۵]

[۴] producers
[۵] modeling languages

در BON: متدولوژی BON مجموعه ای از مدل ها را ارائه می دهد که خاص همین متدولوژی است و در ۱۰ مورد، آن ها را به همراه تعریف کامل هر کدام آورده است؛ پس هر محصول مدلسازی در متدولوژی که درنتیجه ی هر گام ساخته یا آپدیت می شود، تشریح شده است. شامل: نمودار سیستم، نمودار خوشه، نمودار ها کلاس، دیکشنری کلاس، معماری ایستا، اینترفیس های کلاس، نمودار ایجاد، نمودار رخداد، نمودار سناریو و سناریو های آبجکت(استفاده از جدول ۳۹ در مقاله Paige-Ramsin). این را نیز باید در نظر گرفت که متدولوژی سعی می کند تا به طور کلی از زبان مستقل باشد اما عمیقا تحت تاثیر شیوه های assertion زبانی به نام Eiffel و طراحی با قرارداد بوده است.

در Booch: اتفاق مشابهی در Booch نیز مانند BON رخ داده است. Booch تصمیم گرفته است که از زبان مدلسازی مخصوص به خود استفاده کند. این زبان البته بسیار شباهت زیادی به UML دارد که بعدا نیز واضع Booch یکی از سه نفر طراح UML نیز بود.

- **محصولات**[۶]

در BON: نکته ی بسیار خوب متدولوژی این می باشد که فرآیند در توضیح هر گام چه در تحلیل و چه در طراحی، همه ی آنچه که باید در اثر آن وظیفه ساخته شوند یا حتی ویرایش شوند را به صراحت بیان شده اند(جدول ۳۸ از مقاله ی Paige-Ramsin که در ستون آخر، خواستار محصولات هر گام می باشد، شاهدی بر این مدعا می باشد). محصولات و مدل ها در دو قسمت مدل های Static و Dynamic وجود دارند که در معیار های دیگر به مدل ها خواهیم پرداخت؛ اما خود تعریف آن ها، به خوبی پشتیبانی شده است.

در Booch: هرچند که تصریح دقیق از خواسته ی محصول هر فرآیند Macro

[۶] products

به چشم نمی خورد اما انتظاری از مفهوم آن ها برمی آید. به عنوان مثال در re-establish core quirements که درک بهتر مساله را با شناسایی نیازمندی ها مقدر می سازد و به همین نیت گنجانده شده است، برای به عینیت رسیدن نیازمندی ها از دید مشتری است. لذا هدف مشخص هست. حال در هر چرخه تکرار شونده به کمک فرآیند Micro به صورت دقیقی محصولات خواسته شده مشخص شده اند(مثلا محصولات اشاره شده در نمودار ۱۵ و ۱۶ مقاله Paige-Ramsin نمونه ای از ادعا هستند). آنچه که بدیهی است تعریف کامل محصولات مورد انتظار در جریان هر گام می باشد که مشتمل بر ایجاد و ویرایش است.

- **تکنیک ها و قواعد**[۷]

در BON: تکنیک ها در واقع بیان کننده ی چگونگی اجرای work-unit ها می باشند. توجه داریم که چیستی هر واحد کاری پیش از این گفته شد، اما اینکه با چه تکنیک هایی و چگونه به انجام می رسند، باید جداگانه ذکر شود. با این تفاسیر می توان با مراجعه به مقاله Paige-Ramsin و مطالعه ریز جزییات برای هر یک از ۹ گام در طول چرخه عمر فرآیند متدولوژی از این واقعیت پی برد که نحوه ی انجام هر واحدکاری چه گونه است. در آنجا همچنین موارد تولیدی یعنی محصول هر گام نیز ذکر شده است. در نتیجه می توان به جزییات چگونگی کار ها در فرآیند پی برد که خود دلیلی بر پشتیبانی خوب متدولوژی از تکنیک ها به جزییات می باشد.(بند های ۹گانه ی توضیح هر گام در مقاله Paige-Ramsin شاهدی بر این مدعا می باشد).

در Booch: در این متدولوژی نیز دقیقا بیان شده است که جز اینکه مقصود از هر واحدکاری چیست، چگونه آن واحد کاری انجام می شود. این توضیح را می توان در جزییات ریز بند ۴.۴.۲ مقاله Paige-Ramsin به وضوح و سهولت پیدا کرد. هر ۴ فعالیت تکرار شونده ی Micro دقیق مشخص شده اند که چه

[۷] techniques and rules

گونه وصول می شوند و از این نظر غنی عمل می کند.

- **موارد مربوط به فعالیت های چتری**

در BON: در متون، صحبت یا رجوعی به موارد مربوط به فعالیت های چتری برای متدولوژی BON پیدا نمی شود. یعنی نسبت به فعالیت های مدیریتی سطح بالا که ناظر به مرحله ی خاصی از چرخه عمر نبوده و همه ی آن را پوشش می دهد، توجهی صورت نگرفته است و صرفا در هر وظیفه، معیار پذیرشی را مطرح می کند. البته این امر نیز بدیهی می نماید؛ چرا که BON را می توان از متدولوژی های Seminal دانست که عمدتا تعریف درستی از فعالیت های چتری در این طبقه از متدولوژی ها یافت نمی شود.

در Booch: در اصل در متدولوژی Booch آن چیزی که مدنظر بوده است این است که با تعریف فرآیند های Macro بتوان فعالیت های چتری را دربرگرفت. فعالیت های درشت دانه ای که به چرخه ی عمر خاصی متصل نبوده و ناظر به کل فرآیند توسعه می باشد. به طور مشخص در مقاله Paige-Ramsin نیز تشریح شده است که بخش های عمده ی فرآیند Booch رویه های عملی پایه مدیریت نرم افزار هستند؛ مانند تضمین کیفیت، مستندسازی و زمان بندی با نظر مشتری.

به نظر می رسد در این بخش تفاوت عمده ای میان دو متدولوژی پدیدار شده است که از این نظر Booch تعریف به نسبت بهتری را ارئه می دهد و BON ضعیف تر عمل می کند.

- **نحوه تعریف کردن**

در BON: این متدولوژی از آنجا که توجهی به role یا producers ندارد نمی تواند role-centered باشد. از طرفی از آنجا که در جریان توضیح فرآیند، فاز ها را متذکر شده و بعد ذیل هر کدام، فعالیت های داخلش یعنی وظیفه ها

را بیان می کند و سپس به همراه هرکدام مصنوعات را تصریح می کند، پس process-centered می باشد.

در Booch: دقیقا به مانند BON شاهد نحوه تعریف process-centered هستیم. مقاله به ما می گوید که معنای هر واحد کاری چیست و در راستای فاز و ذیل آن یعنی وظایف، محصولات کدامند که از این حیث BON و Booch رفتاری مانند هم از خود بروز می دهند.

۲.۱.۱ پوشش چرخه عمر عمومی ایجاد نرم افزار

معیار بیان می کند که یک متدولوژی در چه حدی چرخه ی عمومی [۸] ایجاد نرم افزار را پوشش می دهد. کما اینکه می دانیم چرخه ی عمومی ایجاد نرم افزار در سه بخش کلی تقسیم می شود: تعریف مساله[۹]، ایجاد[۱۰]، نگهداری[۱۱] که هر کدام ناظر به بخشی از lifecycle متدولوژی است و در صورت تحقق کامل همه ی آنها، متدولوژی مدنظر full-lifecycle خواهد بود. حال هر کدام را با شرح بیشتر بررسی خواهیم کرد.

- **تعریف مساله**

  * کاوش قلمرو مساله و مدلسازی آن

در BON: گام های کافی برای کاویدن قلمرو مساله جهت پیدا کردن ابعاد چیستی سیستم در متدولوژی BON تعریف شده است؛ به حدی که ۶ تا از ۹ گام تعریف شده ی فرآیند به تحلیل صورت مساله اختصاص می یابد. با مراجعه به مقاله Paige-Ramsin و بند ۴.۸ که توضیح اولیه BON آمده است؛ به صراحت ذکر شده است که BON متدولوژی کاملی برای پوشش فاز های عمومی تحلیل و طراحی ایجاد نرم افزار است. گام های ۱ تا ۶ که در دو فاز gathering و describing می باشند، به خوبی کاوش را در قلمرو مساله فراهم می کنند.

در Booch: متدولوژی Booch در دومین فعالیت درشت دانه ای که تحت عنوان develop a model of the system's desired behaviour معرفی شده است، تمرکز بر روی تحلیل را در دستور کار قرار می دهد و بدیهی است که در ابتدای چرخه ی عمر ایجاد نرم افزار از اهمیت بسیار

[۸] generic
[۹] defenition
[۱۰] development
[۱۱] maintenance

زیادی برخورددار است و به مرور و با چرخش تکرار فرآیند های Micro ارزش به سمت فعالیت های درشت دانه ی بعدی می رود. این متدولوژی پوشش کاوش قلمرو مساله را به همراه مدل هایی که می سازد، به خوبی پشتیبانی می کند.(اشاره به بند ۴.۴.۱ مقاله Paige-Ramsin که فعالیت دوم را به صراحت analysis می خواند).

* استخراج نیازمندی ها

در BON: اتفاقی که متدولوژی BON انتظار دارد این است که نیازمندی ها، ورودی به گام های فرآیند هستند. حال این ورودی به چه شکلی صورت می پذیرد و چه گونه مدل می شود، تصریح نشده است؛ که خود می تواند ضعفی در این جا بوجود بیاورد. هر چند از آنجا که کلا قسمت های تعریف مساله به هم تنیده هستند، نیازمندی ها می توانند به مرور در جریان قسمت قبلی نیز تکامل یابند و به شکل و انسجام درستی در نتیجه کاوش قلمرو مساله برسند.

در Booch: فعالیت ابتدایی درشت دانه ی Booch به خوبی می تواند بخش استخراج نیازمندی ها را پاسخ دهد. در واقع آنچه که به تجربه ثابت می شود از نظر تعداد ۱۰ تا ۲۰ درصد نیازمندی های مشتری و از نظر اهمیت تا ۸۰ درصد functionality سیستم در همین قسمت چرخه ی عمر عمومی ایجاد نرم افزار تامین می شود که بدیهی است Booch با در نظر گرفتن فرآیند Macro به نام core reqirements for software establish به تاسیس نیازمندی های هسته ای نرم افزار، اهتمام ویژه ای می ورزد.

* تحلیل امکان پذیری

در BON: هیچ گامی برای امکان سنجی در توضیح متدولوژی نیامده است. در واقع BON فقط مساله را می تواند به خوبی بکاود اما قادر به بررسی

تحلیل امکان پذیری نخواهد بود. بلافاصله نیز بعد از قلمرو مساله، زمانی که دنبال راه حل در قلمرو طراحی و نرم افزار هستیم صرفا جزیی سازی و افزایش مدل ها انجام می شود وخبری از امکان سنجی نخواهد بود.

در Booch: امکان سنجی به طور خیلی مشخصی در متدولوژی گنجانده نشده است. بدین معنا که در سه مرحله اول فرآیند Macro و قبل از پیاده سازی، مخصوصا در مرحله ساخت معماری می توان اندکی از امکان سنجی را متصور شد؛ از آنجا که به دنبال طراحی های نرم افزاری هستیم از آنچه که از تحلیل و نیازمندی های دو مرحله قبل بدست آمده و چیستی مساله مشخص شده است و حال به دنبال افزودن جزییات طراحی هستیم. البته ذکر می کنیم که هیچ تصریحی ذکر نشده و به مانند BON می توان گفت که تفاوت بین این دو از این حیث بسیار کم است.

- **ایجاد**

* طراحی معماری

در BON: این متدولوژی درک درستی از طراحی معماری را تنها به همراه طراحی جزییات می دهد؛ به این منظور که ۳ وظیفه ی آخر فرآیند خود را پس از تکمیل مدل های مساله، به طراحی اختصاص می دهد و مدل های Static و Dynamic خود را می سازد و در نهایت پشتیبانی به نسبت خوبی را در طراحی چه معماری سیستم و چه تفصیلی ارائه می دهد. هر آنچه از مدلسازی نیز نیاز دارد را مطالبه می کند و آن ها را یا می سازد یا قبلی ها را ویرایش می کند. توجه به مقاله Paige-Ramsin نیز که در کاوش قلمرو مساله به آن ارجاع شد، در این زمینه یاری کننده خواهد بود.

در Booch: Booch یکی از فعالیت های درشت دانه ی خود را در اصل برای معماری بنا نهاده است. با ارجاع به مقاله Paige-Ramsin و بند

۴.۴.۱ صراحت بیان درباره ی گام سوم فرآیند Macro به عنوان طراحی مشهود است. توجه داریم که در اینجا طراحی مدل های به دست آمده در مراحل قبل و همچنین جنبه های دیگر طراحی را شامل می شود. یعنی فقط architectural design نبوده که detailed design نیز را پوشش می دهد تا برای پیاده سازی در گام بعد، مستندات کامل دقیقی از طراحی تفصیلی ارائه شده باشد و برای برنامه نویس بسیار شفاف باشد.(فعالیت های Micro این امکان را کاملا مشخص می کنند).

* طراحی با جزییات

در BON: قبلا آن را در طراحی معماری پوشش داده ایم.

در Booch: در طراحی معماری این مورد نیز برای Booch بیان شد.

* برنامه نویسی

در BON: متدولوژی BON هیچ گونه فرآیندی برای پیاده سازی و برنامه نویسی ندارد و در این چرخه عمومی ایجاد نرم افزار نقصی را از خود نشان می دهد.

در Booch: مرحله ی تکامل که از آن با عنوان the implementation evolve یاد می شود، پیاده سازی طراحی را برعهده می گیرد و تکامل برنامه نویسی در اجرای فرآیند نرم افزار به خوبی پوشش داده می شود. بدین سبب، این متدولوژی بخش درشت دانه ای از فرآیند Macro ی خود را به پیاده سازی طراحی اختصاص داده و از این حیث بهتر از BON عمل می کند. لازم به ذکر است که در همین گام، کارهای مربوط به تست و آزمون و همچنین مستقرسازی نیز انجام می شود و به طور کامل، پوشش ایجاد نرم افزار را در دستور کار قرار می دهد.

* آزمون

در BON: به مانند برنامه نویسی، BON نمی تواند چرخه ی آزمون را

تعریف کند و توجهی به این بخش ایجاد نرم افزار ندارد.

در Booch: آزمون در متدولوژی Booch صراحتی برای یک واحد کاری درشت دانه ندارد اما در ذیل تکامل و پیاده سازی در برنامه نویسی گنجانده می شود.

* مستقرسازی[۱۲]

در BON: هیچ نکته ای در باب مستقرسازی فرآیند ایجاد نرم افزار در متدولوژی ذکر نشده است و نقصانی برای BON به حساب خواهد آمد.

در Booch: مستقرسازی در متدولوژی Booch صراحتی برای یک واحد کاری درشت دانه ندارد اما در ذیل تکامل و پیاده سازی در برنامه نویسی گنجانده می شود.

- **نگهداری**

در BON: هیچ توضیحی در باره ی نگهداری برای چرخه ی عمر عمومی ایجاد نرم افزار در این متدولوژی تعریف نشده است.

در Booch: آخرین فعالیت فرآیند Macro در Booch به نگهداری اختصاص دارد. در واقع Booch می خواهد که نگهداری را به طور کامل پوشش دهد و برای آن manage post-delivery evolution را که بنابر مقاله ی Paige-Ramsin به maintenance تصریح می شود، تعریف می کند. حال که نرم افزار در مرحله ی قبلی در محیط کاربر گذاشته شده است، طی واحد های کاری تکرارشونده ای اما با تمرکز بر فعالیت های آخر، نگه داری از محصول کنترل می شود. این متدولوژی از این نظر بسیار جلوتر از BON عمل می کند.

[۱۲] deployment

۳.۱.۱ پشتیبانی از فعالیت های چتری

معیار پشتیبانی از فعالیت های چتری بیشتر ناظر بر اقدامات مدیریتی و سیاست های اتخاذ شده در سطح مدیریت برای متدولوژی ها می باشد. تعداد ۳ مبحث کلی باید در این چتر به آنها توجه شود که برای هر کدام تکنیک هایی موثر می تواند به کار گرفته شود:

- **مدیریت ریسک**

منظور از مدیریت کردن ریسک این است که فعالیت های مربوط به ارزیابی ریسک و کاهش آن چگونه در داخل فرآیند متدولوژی تعریف شده است. برای این مدیریت، تعدادی تکنیک موثر تعریف شده است که برای دو متدولوژی زیر نگاهی به آن تکنیک های به کار گرفته شده خواهیم داشت.

در BON: تحلیل مقدماتی که یافت نمی شود، هیچ گونه پروتوتایپینگ چه از نظر فنی چه از نظر زمانی توصیه نشده است، هیچ گونه اولویت بندی برای نیازمندی ها چه از دید مشتری و چه از لحاظ فنی/زمانی تعبیه نشده است، در اصل فرآیندی افزایشی ـ تکرارشونده ای هم نیست تا با بازنگری امکان تکامل تدریجی را فراهم کند که در نهایت منجر به کاهش ریسک شود. یعنی بعد از قلمرو تحلیل، صرفا به طراحی رفته و ریسک اصلی که در طراحی هم می تواند باشد را نمی تواند مدیریت کند. توجهی به سریع رسیدن به نرم افزار قابل ترخیص[۱۳] ندارد و همچنین برنامه ای برای مرور های مکرر[۱۴] ندارد و خبری از یکپارچه سازی مستمر[۱۵] و تحقیق مستمر[۱۶] نیز نیست. اما نکته ی امیدوار کننده این می تواند باشد که به دخالت فعال کاربر نرم افزار باشد. به گواهی مقاله ی Paige-Ramsin مثلا در جدول

[۱۳] early release
[۱۴] iterative reviews
[۱۵] continuous integration
[۱۶] continuous verification

۴۴ که مختص بند ۴.۸.۲ می باشد، در دومین وظیفه که لیست کردن کلاس های کاندید می باشد، تایید کاربر نهایی و متخصص حوزه برای پذیرش گام نیاز می باشد که باعث بوجود آمدن validation می شود.

در Booch: بارزترین ویژگی Booch به ارمغان آوردن شیوه افزایشی ـ تکرارشونده می باشد و از تکنیک های موثر در سنجش و کاهش ریسک هست. بازنگری مرتب سه تا P معروف در متدولوژی به شدت می تواند موثر باشد تا از جلوی کار پرخطر گرفته شود(تا حد خوبی اعتبارسنجی مستمر را با خود به همراه دارد). این متدولوژی توجه اندکی به تحلیل مقدماتی در اولین Macro و یا حتی در اول Macro ی سوم خود می تواند داشته باشد تا جلوی ریسک گرفته شود. چرا که نیازمندی ها تا سطح خوبی از ارزش می تواند یافت شود و مورد سنجش قرار بگیرد. البته Booch صراحتی را برای دخالت فعال کاربر ذکر نمی کند(و فقط اشاره به دلخواه مشتری می کند) و در بقیه تکنیک ها به مانند BON می باشد.

- **مدیریت پروژه**

برای مدیریت کردن پروژه، متدولوژی باید بتواند متذکر شود که فعالیت های planning و scheduling و همچنین controlling چه گونه تعبیه شده اند.

در BON: متدولوژی BON البته در planning یک مجموعه گام را به صورت پیش نیازی تعریف می کند ولی اجازه جابجایی را فقط در صورت نزدیک تر شدن به هدف پروژه می دهد، اما زودترین و دیرترین زمان شروع و پایان را در جایی مورد توجه قرار نمی دهد. در بحث scheduling نیز به همین ترتیب که نکته ای برای محور زمان روی تقویم مشاهده نمی کنیم. در نتیجه نمی توان در monitoring and control فعالیت خاصی را بیان کرد تا دو مورد قبل را در پروژه مورد پایش قرار دهیم. مبحث مدیریت تیم یعنی ارتباطات و همکاری های بین ـ تیمی و درون ـ تیمی نیز هیچ اشاره ای نداشته است. در کل BON راهکاری

برای مدیریت پروژه ندارد.

در Booch: این متدولوژی توجه به کیفیت، کامل بودن و زمان بندی را تصریح می کند. با توجه به بند ۴.۴ مقاله Paige-Ramsin که توضیح متدولوژی است، این موضوع ذکر شده است و ما تعریف پیش نیازی ها در فرآیند متدولوژی را دیده و بنابر جداول، می توان planning و scheduling داشت و مورد control monitoring and قرار دارد. چرا که می دانیم اساس تعریف فرآیند Macro آن است که با تعیین هدف و محصولات نهایی مخصوص به خود، هفته ها و ماه ها برای آن برنامه ریزی داشت و با فرآیند Micro بتوان فعالیت های روزانه را به کار بست تا اهداف تعیین شده محقق شوند. هرچند شایان ذکر است توضیحی ذیل مدیریت تیم به مانند BON یافت نمی شود.

- **تضمین کیفیت**

برای ضمانت کردن کیفیت نرم افزار لازم هست تا فعالیت های مخصوص سنجش کیفیت و ارتقای کیفیت داخل متدولوژی تعبیه شده باشد. تکنیک هایی مطرح می شود تا به این نیازمندی متدولوژی فائق آمد که در زیر آن ها را بررسی خواهیم کرد.

در BON: این متدولوژی هیچ بازنگری فنی مکرری بر محصولات و مصنوعات را در خود ندارد. فرآیند آن متوالی و بدون برگشت به عقب است. نکته ی مثبت توصیه ذکر شده برای آوردن postcondition و precondition در operation ها می باشد که همان design by contract می باشد و بسیار به تضمین کیفیت کمک می کند. لازم به ذکر است که رهگیری نیازمندی به محصول در این متدولوژی دیده نشده است.

در Booch: کما اینکه پیشتر بیان شد، Booch با رویکرد افزایشی-تکرار شونده ی خود، امکان مرور فنی را به صورت دوره ای می تواند مهیا کند که تاثیر به

سزایی در تضمین کیفیت خواهد داشت. البته Booch استراتژی خاصی را برای بهبود رهگیری نیازمندی به محصول بیان نمی کند اما در کل به جهت این که دلخواه مشتری برای آن باارزش هست، می تواند کیفیت خوبی را تضمین کند.

۴.۱.۱ بی درزی و هموار بودن انتقال

بی درزی[۱۷] و هموار بودن انتقال[۱۸] از معیار هایی هست متدولوژی را مجبور می کند که شیفت پارادایمی نداشته باشد و اطلاعات از یک مرحله به مرحله ی دیگر از بین نرود و مغفول نماند تا ناسازگاری بوجود نیاید.

- **بی درزی**

در BON: در اصل، بی درزی از نکات مهم و قوت BON می باشد. با ارجاع به مقالات Paige - Ramsin و شخص Nerson در سال ۱۹۹۲ می توان تصریح به بی درز بودن متدولوژی را یافت. متدولوژی تماما شیء گرا فکر کرده و مدلسازی های آن بر اساس شیء انجام می شود. یعنی از این تکنیک استفاده کرده که همه ی تکالیف و مصنوعات روی یک مفهوم مشترک بیان شود و اینجا آن مفهوم مشترک، شیء است.

در Booch: متدولوژی Booch مفهوم مشترک آبجکت(کلاس) را بیان می کند ولی هیچ مجموعه ی خاصی از مدل ها را تبیین نمی کند تا با تکامل آن ها و پیشبرد تکالیف حول آنها، درز را کاهش دهیم. در واقع با بیان مفهوم مشترک از وجود درز جلوگیری می کند.

در این بین، به نظر می رسد این دو متدولوژی به مانند هم بی درز بوده و هر دو راه حل های تماما شیءگرا دارند.

- **هموار بودن انتقال**

در BON: به نظر می رسد که گذار های هموار را در BON شاهد هستیم و بین فاز ها، مراحل و گام ها، محصولی کاملا جدید تولید نشده و موارد قبلی تکمیل و بروزرسانی می شود و یا از آن ها استفاده شده تا با ساخت مدل جدید دیگر، ادامه

[۱۷] seamlessness
[۱۸] smoothness of transition

یابند(مدل های Static و Dynamic و رابطه ی آن ها با یکدیگر بنا بر مقاله Paige-Ramsin و نمودار ۴۰ آن). اینکه البته خوشبین باشیم مدل جدیدی در هیچ گام آن تولید نشود را ندارد و تنها مدل ها تکمیل نمی شوند؛ اما در هر صورت با رفتن به گام دیگر، مدل های قبلی کمک کننده به ساخت مدل جدید خواهند بود.

در Booch: در حقیقت Booch با رویکرد افزایشی-تکرار شونده ای که دارد، همواری انتقال بین گام ها را به خوبی مهیا می کند. در فرآیند Micro ما شاهد کشف جدیدی از محصول نیستیم و مدام مدل ها عوض نمی شوند؛ تنها به شکل تکاملی، بازنگری و بروزرسانی می شوند. در خود Macro ها نیز فرآیند انتقال نرم صورت می پذیرد، چون در هدف هر Macro خواسته هایی تامین شده که ورود به Macro ی بعدی را به گونه ای مهیا می سازد که ورودی های خوبی برای آن فراهم کرده است. مثلا برنامه نویس، به خوبی می داند که چگونه توسعه دهد؛ چرا که معماری سیستم به خوبی تدقیق شده است.

۵.۱.۱ مبتنی بر نیازمندی وظیفه ای و غیر وظیفه ای

برای تحقق به این نیازمندی که در سال های اخیر برای یک متدولوژی بدیهی شده است باید ویژگی های زیر در متدولوژی تعبیه شده باشد:

اولا که نیازمندی ها در ابتدای فرآیند و شروع متدولوژی باید ثبت و استخراج شوند.

دوما به تنهایی و به صورت مستقل مدل بشوند و این مستندسازی باید جداگانه و به صورت خاصی اتفاق بیافتد؛ نباید به گونه ای باشد که نیازمندی ها را مجبور باشیم از نموداری که ساختار سیستم را مثلا نشان می دهد برداشت کنیم. نیازمندی باید زود و به صورت خاص مدل شود.

سوما که از آنها برای مبنا در هر واحد کاری بعدی اسفاده کنیم.

حال با این تفاسیر، متدولوژی ها را از جنبه این معیار بررسی می کنیم:

در BON: با ارجاع به مقاله Paige-Ramsin و بررسی گام اول ذکر شده برای متدولوژی BON که به تعیین مرز سیستم توجه دارد و در زیرمجموعه ی تحلیل قرار می گیرد، اشاره ای به مشخص کردن نیازمندی ها می شود که این گام مدل های خاص خود را نیز می طلبد به نام system chart و scenario charts که تا حدودی نیازمندی ها را مدل می کنند ولی به نظر می رسد که هرچند ویژگی اول بالا رعایت می شود اما مورد دوم و سوم جایگاهی ندارد.

از آنجا که مدل سازی نیازمندی با نمودار سیستم و نمودار سناریو به دست می آید، به گواهی جدول ۳۹ مقاله ی Paige-Ramsin تنها یا لیست سناریو ها برای روشن سازی رفتار سیستم نشان داده می شود و یا تعریف سیستم و خوشه های مرتبط که با چیستی ساختمان سیستم سروکار دارند، پس مورد سوم نمی تواند به خوبی پشتیبانی شود.

و چون مبنای کار های بعدی در چرخه عمر فقط نیازمندی و مدل های آن نمی باشد، پس متدولوژی به اصطلاح reqirement-driven نیز نمی باشد که نقصی را از این بابت متوجه می شود اما از طرفی می توان آن را reqirement-based خواند؛ چرا که

گام دوم از پس گام اول که به نیازمندی می پردازد، شروع می شود.

این در حالی است که صحبتی از نیازمندی های غیر وظیفه ای در هیچ کجا در این متدولوژی یافت نمی شود.

در Booch: در مقاله ی Ramsin-Paige با صراحت، فرآیند Macro ی اول Booch کاملا اختصاص پیدا کرده است به اینکه نیازمندی های هسته را تاسیس کنیم. با مشخص شدن سریع نیازمندی های اولیه، کسر زیادی از مطلوب، تعیین می شود. فلذا ویژگی اول بالا را خوب می تواند متدولوژی پاسخگو باشد. اما نکته ی مهم تشابه آن با BON هست. جایی که از مدلسازی خاصی مثل usecase خبری نیست(عدم پشتیبانی از ویژگی دوم) و نیازمندی ها فقط روایی بیان می شوند. پس هر چند requirement-based هست اما reqirement-driven نیست(عدم پشتیبانی از ویژگی سوم). در واقع دوباره شاهد مدلسازی سناریو های تعاملی بین آبجکت ها در جهت تحقق نیازمندی ها خواهیم بود که در فرآیند درشت دانه ی دوم این اتفاق رخ خواهد داد.

توجه زیاد متدولوژی به نگهداری نیز که در آخرین فرآیند Macro مشهود بوده و تنها اهتمام جدی به آن پرداخته می شود که البته مهم ترین نیازمندی غیر وظیفه ای نیز به شمار می رود. از این حیث، Booch بهتر از BON عمل می کند.

۶.۱.۱ قابل آزمون، ملموس و قابل رهگیری به نیازمندی ها

معیار ششمی که به آن می پردازیم، شامل بر ۳ ویژگی می باشد به عنوان های: آزمون پذیری متدولوژی[۱۹]، ملموس بودن متدولوژی[۲۰] و قابلیت رهگیری به نیازمندی ها[۲۱] که هر یک به مفاهیم جداگانه ای اشاره دارند اما عمدتا ناظر بر محصولات یک متدولوژی هستند.

به طور خلاصه متدولوژی ای آزمون پذیر است که به سهولت بتوان متدولوژی و با تکیه ی بر محصولات آن، بدون ابهام آزمون را طراحی و اجرا کرد. نکته در این هست که برای تحقق چنین هدفی، متدولوژی بهتر است تا پیچیدگی محصولات خود را کاهش دهد. تعداد آنان حتی الامکان کم بوده و ساده به طوری که پیچیدگی فهم در حداقل ممکن بوده تا قابل فهم باشد. وابستگی های بین محصولات باید کم بوده و باید وابستگی های حداقلی بین آن ها به خوبی تعریف شده باشد. به عنوان مثال بهتر است تا وابستگی شدید دو نمودار را که یکی با نمودار دیگر، رابطه ی isomorphism دارند جایی در متدولوژی نداشته باشد؛ چرا که در غیر این صورت یکپارچگی و جامع بودن محصولات به خطر افتاده و تناقض می تواند به آسانی پیدا شود. تغییر در یکی، باعث انتشار تغییر می شود که این موارد به خودی خود، قابلیت آزمون پذیری را کاهش می دهد.

ملموس بودن برای یک متدولوژی آن است که محصولات متدولوژی از دید استفاده کننده ی محصول، قابل فهم و موجه باشند. طبق تعریف بدیهی است که تا حد زیادی به بیننده بستگی دارد. حال بیننده می تواند کاربر باشد و یا توسعه دهنده که در هر دو صورت محصولات خاصی متناظر می شوند. اگر از دید یک بیننده، محصولی ملموس نباشد، دلیل وجود آن مشخص و موجه نیست، لذا در ساخت آن ها دقتی نشده و استفاده ی بعدی نیز صورت نمی پذیرد که به نوعی اتلاف منابع به حساب می آید. مثلا

[۱۹] testability
[۲۰] tangibility
[۲۱] traceability to requirements

برای کاربر، محصولات قابل اجرا مانند پروتوتایپ ها و یا محصولاتی که برای او معنی داشته باشند و طبق سواد او ساخته می شود مثل usecase ها، ملموس بودن خوبی را در محصولات متدولوژی می رساند. برای توسعه دهنده نیز مصنوعاتی که خیلی واضح در چرخه عمر و واحد های کاری فرآیند، کاربرد دارند و حضور آن ها موجه هست، از ملموس بودن محصولات متدولوژی خبر می دهد.

در نهایت قابل رهگیر بودن به نیازمندی ها به این معنا می باشد که محصولات متدولوژی را به یک نیازمندی خاص یا مجموعه ای از آن ها رهگیری کرد. بدین ترتیب مصنوعات باید مستقیم یا غیرمستقیم، پیاده سازی نیازمندی ها باشد و یا از طریق سناریوهای ارزیابی مبتنی بر نیازمندی بشود این تناظر را نشان داد. بدین معنا که از سناریوهایی استفاده می شود تا تحقق نیازمندی را بسنجند و سپس سناریو را روی مصنوعات متدولوژی اجرا می کنند و مشخص می شود که آیا نیازمندی محقق شده است یا خیر و اگر شده است کدام مصنوع درگیر در تحقق نیازمندی بوده است و اینگونه تناظر را بدست می آورند.

با این تعریف های مقدماتی به سراغ دو متدولوژی زیر رفته و هر کدام را به تعاریف بالا می سنجیم. البته همانطور که ذکر شد تاکید بر روی محصولات متدولوژی خواهد بود:

- **آزمون پذیری**

در BON: مقاله Paige-Ramsin در صفحه ۴۵ و جدول ۳۹ و نمودار ۴۰ مطلب خوبی را در این زمینه به نمایش می گذارد. با ارجاع به آن، محصولات متدولوژی را می توان زیاد دانست(به تعداد ۱۰ مصنوع منحصر به فرد) اما به دلیل خوب تعریف شدن، اصل قابل فهم و ساده بودن را دارند(مبتنی بر مفهوم شیءگرایی و مخصوصا کمک گرفتن از assertion ها). نکته ی منفی اما در نموار ۴۰ و پاراگرافی از مقاله می باشد که وابستگی های داخلی آن ها را متذکر می شود که می تواند تاثیر قابل توجهی در آزمون پذیری داشته باشد. چرا که انتشار

تغییر ناشی از تغییر کردن یک محصول خواهیم داشت که نشان دهنده رابطه ی بالای بین محصولات هست. البته نکته ای که می توان مثبت دانست این هست که فرآیند متدولوژی در نمودار ۳۸ مقاله سعی می کند که در گام ها به یکباره، محصولات کاملا جدیدی ایجاد نکند و آزمون پذیری را بهبود ببخشد.

در Booch: هرچند که این متدولوژی فرآیندی پیچیده تر دارد اما در نهایت محصولاتی قابل فهم و ساده را مانند BON به تصویر می کشد. محصولات آن را هم می توان کمتر از BON دانست و در مقاله Paige-Ramsin تعداد ۶ تای آن موجود است(که بعضی نیز شباهت فراوانی به UML دارند). ماهیت افزایشی- تکرارشونده ی متدولوژی به مرور کامل شدن محصولات کمک می کند که خود کمی به آزمون پذیری است اما به مانند BON ضعف کار در آنجایی هست که روابط بین محصولات تصریح نشده است؛ هرچند همدیگر را کامل کردن نمودار ها به قابل آزمون بودن محصولات کمک می کند.

- **ملموس بودن**

در BON: محصولی در متدولوژی یافت نمی شود که برای کاربر یا توسعه دهنده، ملموس نباشد. البته کاربر خیلی نمی تواند با محصولات قابل اجرا سر و کار داشته باشد(چرا که نه ترخیص زودرس محصول داریم و نه شیوه های پروتوتایپینگ) و تنها در تعدادی از مدل ها می تواند نظرش تعیین کننده باشد. از آنجایی که تکامل و افزودن جزییات محصولات به حد قابل قبولی در حین و بعد از تحلیل نیز انجام می شود، توجه پذیری محصولات برای افراد، وضوح زیادی خواهد داشت.

در Booch: نکات BON برای Booch نیز در این حوزه درست هست؛ با این تفاوت که متدولوژی Booch توجهی به مدل سازی خاص نیازمندی ندارد و کما این که پیشتر گذشت، نیازمندی هسته ای حتی می تواند روایی بیان شده باشد.

لذا نیازمندی ای در چنین سطح انتزاع می تواند توسعه دهنده را به دردسر انداخته و گمراه کند تا از استفاده از آن بی توجه باشد و در نهایت وضوح محصول را از او بگیرد؛ که باعث کاهش ملموس بودن می شود.

- **قابل رهگیری به نیازمندی ها**

در BON: متدولوژی BON برای رهگیری به نیازمندی هایی که در گام اول سعی داشته آن ها را برای قلمروی مساله به خوبی شناسایی کند، در همان گام اول به کمک نموداری مانند scenario charts سعی در ایجاد قابلیت رهگیری دارد. بدین این صورت که این نمودار رفتار سیستم را به وسیله سناریو های شیء ای لیست می کند که امکان رهگیری را از طریق روشن کردن نیازمندی ها فراهم می کند. البته جایی تصریحی صورت نگرفته است که سناریو را باید بر روی مصنوعات اجرا کرد. از آنجا که BON یک متدولوژی reqirement-based می باشد می توان این ویژگی را نکته ای مثبت ارزیابی کرد و نیازمندی ها را به طور غیر مستقیم از مصنوعات رهگیری کرد.

در Booch: همانطور که در ملموس بودن بیان شد، ضعف اصلی Booch به نسبت BON در این قسمت می باشد که نمی تواند مستند درستی از هسته های نیازمندی بدهد. فقط فرآیند Macro یی به تاسیس نیازمندی های هسته ای می پردازد ولی برای آن مدل خاصی ارائه نمی دهد. این کار سبب خواهد شد که قابلیت رهگیری به نیازمندی به طور ویژه ای آسیب ببیند. نه الزامی برای تعریف سناریو ها و نه روشی مستقیم برای ایجاد تناظر بین نیازمندی و محصولات در این متدولوژی به نظر می رسد. اما درنهایت به مانند BON چون مبتنی بر نیازمندی است، تناظر غیر مستقیم را می توان برای آن متصور شد(هرچند که دشوار خواهد بود).

۷.۱.۱ دخالت فعال کاربر

هفتمین معیار، یکی از تکنینک های کمک کننده به مدیریت ریسک بود و باعث خلق اعتبارسنجی مستمر نیز می شد که تاثیر مستقیم مثبتی بر تضمین کیفیت داشت و حال تبدیل به یک معیار برای ارزیابی و مقایسه ی متدولوژی ها شده است. چرا که به قدری مهم است که عدم رضایت مشتری را تا حد مطلوبی از بین برده و طرز نگرش در شکاف "ما" و "آنها" را از بین می برد و تهدیدی به نام مشتری را تبدیل به فرصت می کند.

دو روشی که موثر شناخته شده است تا این معیار پوشش داده شود این است که:

۱. **نماینده ی کاربر، عضوی از تیم ایجاد باشد.**

۲. **جلسات برنامه ریزی و مرور و بازنگری با شرکت کاربران به صورت دوره ای برگزار شود.**

با این حال می خواهیم این معیار را برای دو متدولوژی زیر بررسی کرده و آن ها را با یکدیگر مقایسه کنیم:

در BON: آنچه که از دخالت کاربر به یکی از دو شکل بالا نام برده می شود و دخالت مستمری را شاهد باشیم، این متدولوژی به مانند Booch نمی تواند چیزی را عرضه کند. اما صرفا با مراجعه به مقاله Paige-Ramsin دیده می شود که در انتهای گام ها، تایید کاربر، نهایی کننده برای ورود به گام بعدی خواهد بود که از این نظر BON می تواند وضعیت خود را بهبود ببخشد.

در Booch: هرچند که تصریحی در متدولوژی برای تکنیک اول وجود ندارد اما Booch به دلیل پشتیبانی از جلسات بازنگری دوره ای، مرور و تکامل را به صورت افزایشی در دستور کار دارد که می توان گفت تا حدودی از روش دوم بالا بهره برده است. لازم به ذکر است که هرچند شرکت کاربر در جلسات را متدولوژی صریحا متذکر

نشده و می توان فقط از دید متخصصین ایجاد و توسعه به آن نگاه کرد؛ اما می توان در هر تکرار، بازخورد کاربران را سنجید.

۸.۱.۱ قابل اجرا بودن و استفاده کارآ

قابل اجرا بودن متدولوژی[۲۲] و به شکل کارآ استفاده کردن یا کارآیی متدولوژی[۲۳] در هشتمین معیار برای ارزیابی متدولوژی بیان می شود. این دو ویژگی تا حدی مهم هستندکه به صورت معیار جداگانه ای برای متدولوژی درآمده که تعریف های مطابق زیر دارد:

می گوییم متدولوژی قابل اجرا می باشد زمانی که بتوان آن را اعمال کرد؛ یعنی بتوان فرآیندی را که ذکر می کند، به کار بست. این صحبت برای محقق شدن به ۲ ویژگی وابستگی دارد. اولا آن که فرآیند خیلی پیچیده نباشد؛ چرا که پیچیدگی فرآیند نهایتا قابل اجرا نخواهد بود و یا به تمامیت قابل اعمال نیستند. مانند RUP که در حال حاضر بر همگان مشخص است که فرآیند پیچیده ی آن، قابلیت اجرایی آن را مختل می کند. سنگین بودن RUP در نهایت به ظهور RMC منجر شد.

دوما که به ماهیت و طبیعت پروژه هدف خیلی بستگی دارد که چه چیزی را هدف نهایی قرار داده است. بدیهی است که متدولوژی ای که برای موارد پروژه ای ساده تعریف شده است، نمی تواند در فرآیندی در خور اعمال برای پروژه های سنگین باشد و بالعکس. این مورد را نمی توان از ذات یک متدولوژی تحقق بخشید.

اما می گوییم یک متدولوژی به شکل کارآ استفاده می شود و کارآیی دارد زمانی که ابتدا قابل اجرا بر روی پروژه باشد و سپس چه میزان کارآیی را با خود به همراه می آورد تا هدف پروژه بدست آورده شود. بدین معنا که متدولوژی در حوزه کاربرد خودش آیا می توان به شکل موثر از آن استفاده کرد یا خیر که برای احقاق آن ۵ ویژگی ثبت می شود.

اولا که متدولوژی بدیهی است اگر اجزا پیچیده ای داشته باشد، کارآیی کاهش می یابد. سه تا P و زبان مدلسازی باید در بهینه ترین پیچیدگی ممکن شکل بگیرند.

دوما فعالیت هایی که حواس ایجادکنندگان نرم افزار را از فعالیت اصلی دور می

---

[۲۲] practicability

[۲۳] practicality

کند و یا جزییات بدون مورد و غیرضروری را منتقل می کند، حذف کرد و توجهات را به نکات اصلی معطوف کرد تا کارآیی بالاتر برود. این کار با جلسات مدیریت تیم، مدلهایی مبتنی بر نیازمندی ها و ایجاد معماری سیستم امکان پذیر خواهد بود.

سوما از وابستگی به تکنیک های خطا مثلا لزوم استفاده از همه ی نمودار های UML و یا پارادایم شیفت ناشی از حرکت های درز دار پرهیز کرد تا کارآیی افت نکند.

چهارما آنکه متدولوژی اگر به ابزار یا تکنولوژی خاصی وابسته باشد (مانند XP که برای اعمال نیاز به ابزار خاص خود دارد) تاثیر مستقیم روی کاهش کارآیی دارد.

پنجما هم اینکه اگر در متدولوژی، استراتژی ها فعالیت های مدیریتی برای پروژه پشتیبانی و پیش بینی نشده باشد و ناکافی باشد، تاثیر منفی بر روی کارآیی خواهد داشت.

حال با این تعریف اولیه به سراغ مقایسه دو متدولوژی می رویم:

- **قابل اجرا بودن**

در BON: در کل می توان این متدولوژی را قابل اعمال دانست. چرا که پیچیدگی در خود فرآیند یافت نشده و خیلی مرتب، گام ها را بیان کرده و حتی قابلیت تغییر ترتیب را نیز به شروطی داده است. یعنی می توان آن را به تمامیت، تا آخر پیش برد. مدل هایی که ساخته می شود همگی در خدمت وظایف هستند و پیچیدگی را به فرآیند متحمل نمی شوند. از طرفی بدیهی است اگر پروژه ای در طبیعت خود به نگه داری نیاز داشت، BON نمی تواند گزینه ی مطلوبی باشد.

در Booch: به طور کلی Booch از نظر اجراپذیری خیلی خوب عمل می کند. چرا که گام های مشخص آن به خوبی تبیین شده اند و پیچیدگی فرآیند مشهود نیست. در نتیجه واحد های کاری به خوبی مشخص شده و مرز درونی چرخه عمر، شفاف هست. تمامیت آن مانند BON قابل اعمال خواهد بود ولی نکته ی مثبت آن نسبت به BON این است که می تواند جوابگوی خوبی برای پروژه های ذاتا نیازمند به نگهداری باشد. در کل سنگینی فرآیند مشهود نیست و روال افزایشی ـ تکرار شونده ای که به همراه دارد، پیچیدگی را کنترل می کند.

- **کارآ بودن**

در BON: متدولوژی BON به نقش افراد کاری ندارد ولی گام های ۹ گانه ی آن به همراه ۱۰ نمودار مختلف می تواند اجزای فرآیند را پیچیده کند و تاثیر منفی را روی کارآیی داشته باشد. نیازمندی مبنای همه محصولات بعدی لزوما نبوده که احتمال حواس پرتی را بیشتر می کند. جلسات مدیریتی در آن تصریح نشده است و استراتژی خطاخیزی مانند وابستگی زیاد نمودار ها و مدل ها به هم به کار گرفته می شود(هرچند بی درزی می تواند کمک کننده ی به این مورد باشد). BON البته تاثیر گرفته از Eiffel هست اما الزاما وابسته به ابزار خاصی نخواهد بود کما اینکه در مقاله Paige-Ramsin تنها به اثرپذیرفتن متدولوژی سخن می گوید نه وابسته بودن؛ که این خود تنها نکته ی مثبت کارآیی BON می باشد. در آخر آنکه BON کنترل های مدیریتی را تصریح نمی کند و به نظر استراتژی مدیریت پروژه ی مشخصی ندارد که می تواند برای کارآیی مطلوب نباشد.

در Booch: متدولوژی Booch از این نظر اوضاع بهتری نسبت به BON دارد؛ چرا که به افراد کاری ندارد اما کل فعالیت های درشت دانه ی آن ۵ گام است و نمودار های کثیری به نسبت BON ندارد؛ پس پیچیدگی اجزای فرآیند آن کمتر خواهد بود. با بکارگیری جلسات منظم و دوره ای، سعی در حفظ تمرکز خواهد داشت. تمام سعی خود را برای دوری از استراتژی های خطا خیز به مانند BON می کند و بدین منظور مدل های خود را بی درز جلوه می دهد. سپس به طور مشخصی به ابزار یا تکنولوژی خاصی خود را محدود نمی کند و در نهایت با استراتژی های مدیریتی مثل تکراری-افزایشی تاثیر شگرفی را بر روی کارآیی خواهد گذاشت.

۹.۱.۱ قابلیت مدیریت پیچیدگی

قابلیت مدیریت پیچیدگی به عنوان معیار بعدی بیشتر روی مدیریت پیچیدگی فرآیند توجه دارد. این که واحد های کاری در چرخه عمر متدولوژی باید بتوانند قابل مدیریت باشند. این مدیریت از دو طریق البته قابل دسترسی است.

اولا به کمک تقسیم بندی[۲۴] که اجازه ی تجزیه ی یک کل پیچیده را به قسمت های تشکیل دهنده ش می دهد و به این ترتیب اجزای ساده تری را بدست می آوریم و درک و اجرای کار، تسهیل می شود. چرا که با شکستن، قابلیت فهم بالا تر رفته و از سطح انتزاع(abstract) کلی گویی دوری می کنیم. بدین ترتیب، متدولوژی می تواند فرآیند پیچیده ی خود را قابل مدیریت سازد.

دوما به کمک لایه بندی[۲۵] که اجازه می دهد تا از سطح بالا به سطح جزئی[۲۶] به صورت لایه به لایه حرکت کنیم و در این پیشروی، مفاهیم را به ترتیب درک کنیم تا درک ساده تر شود. با افزوده شدن جزییات در عمق لایه، پیچیدگی کار کاهش یافته و مدیریت می شود.

از جنبه ی مطرح شده می خواهیم دو متدولوژی را مقایسه کنیم و بررسی کنیم که هر یک آیا از شیوه های بالا استفاده کرده اند یا خیر.

در BON: این متدولوژی در کل فرآیند خود را اینگونه تعریف کرده است که در ۳ فاز و هر فاز در ۳ گام، تمام چرخه ی عمر پوشش می خورد. در حقیقت BON در تقسیم بندی می تواند موفق عمل کرده باشد و لایه بندی را نیز برای هر فاز، یک عمق گسترش داده است(و همانطور که گفته خواهد شد تا حدودی به مانند Booch عمل می کند). لذا متدولوژی، قابلیت مدیریت پیچیدگی بیشتری را از خود نشان نداده ولی در کل سطح قابل قبولی می تواند باشد.

در Booch: کل زحمتی که Booch می کشد تا بتواند پیچیدگی را مدیریت کند،

[۲۴] partitioning
[۲۵] layering
[۲۶] concrete

در ابتدا شامل شکستن فرآیند ها در فرآیند Macro به ۵ بخش و سپس در یک لایه برای Macro ها چرخه ی بازگشتی خود را بیان می کند که در خود فرآیند Micrco نیز شاهد تجزیه فرآیند به ۴ بخش هستیم و بدین ترتیب سعی دارد پیچیدگی فرآیند خود را مدیریت کند که به نظر سطح قابل قبولی از قابلیت مدیریت را شامل می شود(تفاوت اساسی با BON ندارد). اما نکته ی اساسی در ویژگی خاص Booch هست که فرآیند به آن اجازه برگشت به عقب داده و تکامل را در بروزرسانی قسمت های قبلی می تواند جستوجو کند که سطح خیلی بالاتری از مدیریت فرآیند های Macro و Micro خواهد بود و تقسیم بندی و لایه بندی، اینجا خیلی به کمک آن خواهد آمد.

۱۰.۱.۱ قابل گسترش، مقیاس پذیر، قابل پیکربندی، انعطاف پذیر

دهمین معیار درگیر در چهار تا خصوصیت تحت عنوان های گسترش پذیری[۲۷]، مقیاس پذیری[۲۸]، قابلیت پیکربندی[۲۹]، انعطاف پذیری[۳۰] می باشد که می خواهیم با بسط دادن هر کدام، بین BON و Booch مقایسه کنیم.

گسترش پذیری آن است که بتوان به متدولوژی قابلیت هایی را اضافه کرد. بدین معنا که برای پروژه های خاص، بتوان نیز متدولوژی را خاص کرد. اگر بتوان متدولوژی را به شکل هسته ی قابل گسترش[۳۱] بهترین شکل قابلیت گسترش خواهد بود؛ چرا که در زمان اجرا و اعمال متدولوژی بتوان آن را به شکل دلخواه متناسب با پروژه، گسترش داد.

نکته ی اساسی آن است که اگر قابلیت گسترش را برای متدولوژی باقی می گذاریم، باید نقاط گسترش[۳۲] را نیز مستند کرده و گفته شود که از چه قسمی و با چه مکانیزمی، گسترش می تواند انجام پذیرد.

مقیاس پذیری را می توان این گونه تعریف کرد که فرآیند متدولوژی تا چه حد می تواند روی پروژه های با سایز های مختلف اعمال شود. پس متدولوژی با مقیاس پذیری بزرگ می تواند می تواند در حوزه کاربرد خودش، هم روی پروژه های کوچک(تعداد افراد درگیر در هر لحظه کم) و هم روی پروژه های بزرگ (تعداد افراد درگیر در هر لحظه زیاد) اجرا شود و بدیهی است که هر چه گسنره سایز بیشتر باشد، متدولوژی از این نظر وضع بهتری دارد. البته جز سایز، میزان بحرانیت[۳۳] نیز اهمیت دارد که می توانند به جای هم گذاشته شوند و مانند سایز می توان با آن برخورد داشت. نکته ی

---

[۲۷] extensibility
[۲۸] scalability
[۲۹] configurability
[۳۰] flexibility
[۳۱] extensible core
[۳۲] extension points
[۳۳] criticality

آخر آن که برای رسیدن به مقیاس پذیری بالا، می توان از مدلسازی غنی کمک گرفت.

قابلیت پیکربندی یعنی فرآیند متدولوژی را در ابتدای پروژه پیکربندی کنیم تا با وضعیت پروژه ای تطابق پیدا کند. لذا اگر متدولوژی را بخواهیم از نظر قابلیت پیکربندی سنجید باید دید که برای فاکتور های خاص موقعیت پروژه ای، می توان پیکربندی را برای متدولوژی ارائه داد یا خیر.

انعطاف پذیری به سادگی بدین معنی است که آیا می توان فرآیند متدولوژی را در حین اجرا مورد بازنگری قرار داد یا خیر. برای رسیدن به چنین ویژگی ای، متدولوژی ها می توانند از دو تکنیک استفاده کنند، اول آنکه جلسات مرور فرآیند به صورت تکرار شونده ای شکل بگیرد و دوم آنکه از افراد درگیر پروژه، بازخورد ها مرتبا گرفته شده و براساس آن بازنگری مدیریتی انجام می شود. امروزه از هر دو روش به صورت ترکیبی می توان استفاده کرد. یعنی در طول اجرای متدولوژی، بازخورد ها ثبت شده و سپس در جلسات از آن ها برای بازنگری فرآیند استفاده می کنیم.

اما مقایسه BON با Booch از منظر ۴ تا ویژگی:

- گسترش پذیری

در BON: این متدولوژی نمی تواند مستندی را در این باره به طور صریح ذکر کند. نه صحبتی از افزایش قابلیت ها شده است و نه نقاطی برای اضافه شدن به هسته ی اصلی. یعنی BON برای همه مدل پروژه ها، به شکل جامع خود تا آخر بدون کمی ـ کاستی در تعداد گام ها اجرا می شود و خودش، هسته ی کل است و نه هسته ی قابل گسترش.

در Booch: در اینجا Booch نیز دقیقا مانند BON عمل می کند و همه ی توضیحات آن، اینجا نیز صادق است.

- مقیاس پذیری

در BON: خود Nerson در مقاله ی سال ۱۹۹۲ ادعا می کند که BON به

مقیاس پذیری دست یافته است. چون که فکر می کند بازنمایی کلاس ها با خوشه ها به روش های فرمال، می تواند بحرانیت را کنترل کند. می توان گفت که بله در این حد موفق بوده است و Nerson ادعای اشتباهی نمی کند و BON با شیوه ی کامل شیءگرای خود در حوزه کاربردش تا حدی روی پروژه های بزرگ تر نیز قابل اعمال است. اما از طرفی نیز دقت می کنیم که با تعریف های امروزه، پروژه های خیلی بزرگی که نیازمند درگیری افراد زیادی در هر لحظه می باشد و به نگهداری نیازمند هستند، نمی توانند BON را انتخاب درستی بدانند. چون برای سایز پروژه در تعریف امروزه، راه حلی ندارد.

در Booch: می توان گفت که Booch باز هم از ویژگی خاص خودش یعنی تکراری-افزایشی سطح خوبی از مقیاس پذیری برای پروژه های بزرگ را فراهم کرده است و سیستم بزرگ با دشواری مدیریت(تعداد افراد درگیر در هرلحظه زیاد هست) را به پیچیدگی کمی تبدیل می کند. وجود فعالیت درشت دانه ی نگهداری در آن، از نقاط قوتش در قیاس با BON می باشد که گستره ی سایز پروژه ها را افزایش می دهد. البته تنها نقطه ی ضعفی که در این زمینه دارد استفاده نکردن از روش های formalism مخصوصا برای نیازمندی ها می باشد که برای پروژه های با سطوح ریسک بالا، آن را از قابلیت اعمال دور می کند.

- قابلیت پیکربندی

در BON: از نکات این متدولوژی آن است که در ابتدای توضیح گام های خود، بنا بر ادعای مقاله Paige-Ramsin اجازه تغییر در اجرای ترتیب گام های فرآیند را می دهد؛ به شرطی که به اهداف پروژه نزدیک تر شویم و با موقعیت های خاص پروژه ای بیشتر جور شود. این همان پیکربندی می باشد.

در Booch: نکته ی ضعف Booch به نسبت BON همین جا می تواند باشد که در Booch نکته ای درباره امکان پیکربندی فرآیند با توجه به فاکتور های خاص

پروژه ای وجود ندارد.

- انعطاف پذیری

در BON: متدولوژی BON تکنیک خاصی برای انعطاف پذیری ندارد. هرچند قابل پیکربندی است؛ اما قابل پیکربندی مجدد در طول اجرای فرآیند نیست. نه بازگشتی دارد و نه تکراری در آن تصریح شده است. تذکری برای جلسات مکرر و ارائه ی بازخورد ها ندارد و در کل از نظر انعطاف پذیری ضعیف عمل می کند. این را هم دقت داریم که بازخوردهای گرفته شده از کاربران برای محصولات هست و ربطی به بازنگری در فعالیت ها و فرآیند متدولوژی ندارد.

در Booch: یکی از ویژگی هایی که Booch از آن به خوبی بهره می برد و نکته ی مثبتی در مقایسه با BON دارد همین ویژگی منعطف بودن آن هست که ناشی از ماهیت افزایشی-تکرار شونده ی فرآیند های خود هست. هم فرآیند Macro و هم .Micro متدولوژی لزوم و وجود جلسات در پایان دوره ها را متذکر می شود و هر چند نمی توان فرآیند را در ابتدای کار مورد پیکربندی قرار داد، اما در حین اعمال می توان وزن (سبک-سنگینی) فرآیند ها را تغییر داد و در موارد موردنیاز به عقب بازگشت و فرآیند های قبلی را طی کرد تا ویرایشی انجام شود. بدین گونه فرآیند، بازنگری شده و جلسات مرور فرآیند تکرار می شوند.

۱۱.۱.۱ حوزه کاربرد

حوزه کاربرد به عنوان آخرین معیار فرآیندی بیان می کند که یک متدولوژی در چه حوزه ای کاربرد دارد و برای چه تیپ از پروژه هایی استفاده نخواهد شد. خب البته بدیهی است که این معیار به context کار خیلی وابسته است و به عنوان حداقل باید متدولوژی های ایجاد سیستم نرم افزاری، سیستم های اطلاعاتی مخصوصا از نوع data-intensive را پشتیبانی کنند که حدی از مدلسازی را خواهد طلبید.
با این مقدمه دو متدولوژی زیر را از نظر حوزه کاربرد بررسی خواهیم کرد.

در BON: البته آن چه که از شواهد و قراین پیدا هست این است که متدولوژی خود به صراحت حوزه کاربردی را ذکر نکرده است؛ اما با روش شی گرای بدون درز خود، حداقل نیازمندی در حوزه کاربرد را می تواند فراهم کند. با به کارگیری روش های صوری نیز می تواند درصد قابل قبولی را در حوزه های با حساسیت ریسک بالا پشتیبانی کند. البته به دلیل نداشتن فازی برای نگه داری نمی توان آن را برای سیستم هایی که به اصطلاح compute-intensive هستند، به کار برد؛ چرا که عملیات در آنجا ساده نیست و درصد رخداد خطا بالاتر خواهد بود که این نکته، ضعفی در برابر Booch خواهد بود.

در Booch: متدولوژی Booch نمی تواند پاسخگوی سیستم های با ریسک بالا باشد؛ چون فاقد بهره گیری از روش های صوری است و BON از این بابت بهتر عمل می کند. اما به دلیل ساختار افزایشی ـ تکرار شونده ی خود می تواند پروژه های با تعداد نفرات زیاد در هر لحظه را به خوبی پشتیبانی کند؛ زیرا می تواند تمرکز را بر روی فرآیند Macro عوض کند و مدیریت را تسهیل می کند. بدیهی است که از سیستم های اطلاعاتی حساس به داده می تواند پشتیبانی کند و علاوه بر آن به دلیل Macro ی نگه داری، توانایی دارد تا در حوزه ی سیستم های اطلاعاتی حساس به محاسبه و پردازش نیز اجرا شود که نقطه ی قوتی در برابر BON خواهد بود.

۲.۱ معیارهای زبان مدلسازی

۱.۲.۱ مدلسازی شیءگرای سازگار، دقیق و بدون ابهام

اولین معیار برای سنجش متدولوژی از نظر زبان مدلسازی، تحت عنوان پشتیبانی از مدلسازی شیءگرایی سازگار(بدون تناقض)، صحیح و دقیق و بدون ابهام بیان می شود.

بدین منظور مدلسازی باید سعی خود را بکند که اولا مدل های ایزمورف هم ندهد تا تغییر در یکی باعث انتشار تغییر در مستندات دیگر نشود(هرچند که با تعریف فرآیند می توان این مورد را کنترل کرد).

دوما مدلسازی، جزییات را به اندازه کافی که ابهام از بین برود ارائه دهد، یعنی از سطح انتزاع بالا و مفهوم به سطح پایین ساده ی قابل فهم جزئی بتوان مدلسازی کرد(مدلسازی منطقی به فیزیکی).

سوما زبان مدلسازی باید بتواند هر سه دیدگاه مختلف رفتاری، وظیفه مندی و ساختاری را پشتیبانی کند و از این نظر نقصی نداشته باشد تا دست متدولوژی باز باشد.

چهارم آن که بتواند برای راههای صوری و غیر صوری جوابی داشته باشد تا بتوان از هر کدام که متدولوژی خواستار است، در اختیار بگذارد.

و در نهایت آن که بتواند سطوح درشت دانگی را به خوبی پشتیبانی کند. از مدل enterprise تا مدل های intra-Object که برای متدولوژی ضروری می نماید.

حال متدولوژی های BON و Booch که هر کدام مدلسازی خاص خود را دارند می توان با زبان مدلسازی، آن ها را مقایسه کرد:

در BON: متدولوژی از مجموعه مدلسازی استفاده می کند که مدل های خود را به ۲ دسته Static و Dynamic تقسیم می کند. در این مدل ها، به گواهی مقاله ی Paige-Ramsin و در جدول ۴۰ وابستگی زیادی بین مدل ها وجود دارد، از هر سه دیدگاه پشتیبانی می کند که مثلا چارت های کلاس نمونه ای از دیدگاه وظیفه مندی، سناریوهای شیء نمونه ای از دیدگاه رفتاری و چارت سیستم نمونه ای از دیدگاه ساختاری است. استفاده از روش های صوری در طراحی مشهود است(تشریح پیش شرط و پس

شرط ها در نمودار های طراحی) و همچنین به خوبی می تواند در فاز ها و گام های متوالی، مدلسازی منطقی را به فیزیکی تبدیل کند؛ چرا که طراحی به خوبی می تواند سیستم را بازنمایی کند و مدل های قبلی به خوبی به تکامل کمک می کنند. سطوح درشت دانگی مدل ها هم به خوبی می تواند همه ی سطوح را به جز enterprise یاری کند. برای سطح درون شیءای از اینترفیس کلاس، برای بین شیءای از چارت سناریو و سیستم و زیرسیستم از چارت سیستم و خوشه می توان با آسودگی استفاده کرد(استفاده از جدول ۳۹ مقاله ی Paige-Ramsin که تعریف مدل ها را ساده بیان کرده است). در کل ضعف زیادی جز رابطه های بین مدل ها در BON یافت نمی شود.

در Booch: متدولوژی Booch در این مقایسه نمی تواند مدلی را برای روش صوری بدهد و از این نظر در برابر BON نقطه ی منفی دارد. از آنجا که مدل های بسیار نزدیک به UML دارد، به مانند آن کامل است. چارت حالت برای وظیفه مندی، نمودار تعامل[۳۴] برای رفتاری و نمودار کلاس برای ساختاری تعریف شده است، که از این نظر مانند BON نمره ی قابل قبولی می گیرد. جزییات از سطح بالا به سطح تفصیلی خیلی کامل انجام می شود به طوری که نمودار های رفتاری آن تعاملات رو به ریز جزییات می توانند بیان کنند و مدلسازی فیزیکی کافی انجام می شود. حتی الامکان سعی دارد که از وابستگی پرهیز کند اما نمودار کلاس آن با دیاگرام آبجکت و تا حدی با چارت حالت وابستگی داشته و وابستگی حداقلی را نیز با نمودارهای تعاملی خواهد داشت، اما اگر بخواهیم با BON مقایسه کنیم، Booch از این نظر وضع مناسب تری دار؛ چرا که فرآیند تعریف شده در آن می تواند کنترل خوبی روی سازگاری داشته باشد. در آخر نیز از لحاظ درشت دانگی به مانند BON عمل کرده و تنها نقشه ای برای تصویر enterprise ندارد.

[۳۴] interaction

۲.۲.۱ مدیریت تناقض ها و پیچیدگی ها

معیار آخر ناظر بر مدیریت تناقض ها و پیچیدگی ها می باشد.

به این منظور که زبان مدلسازی به اندازه کافی راهبردها و سازوکارهایی در اختیار بگذارد که بشه جلوی ناسازگاری ها را گرفت تا تناقضی بوجود نیاید. از طرفی پیچیدگی مدلها را مدیریت کرد. دو راه مرسوم برای کمک به زبان مدلسازی وجود دارد. اول آن که قواعد معنایی باید کامل باشد. قواعد معنایی کامل مدل ها به تعریف مفاهیم مختلف در مدل ها و ارتباط دهنده ی انواع مدل ها کمک شایانی می کند. اگر قواعد به درستی بیان نشده باشد ریسک ظهور تناقض در نمودار ها زیاد می شود.

دوم آن که زبان مدلسازی باید ابزار های کنترل پیچیدگی را بدهد. مثل نمودار پکیج در UML که بهترین مثال برای این مضمون هست.

در BON: توضیح نمودار ها به طور خوب و کافی در مقاله Nerson در سال ۱۹۹۲ آمده و معنای هر کدام واضح می باشد. تکامل کم کم نمودار ها به مدلسازی کمک می کند که از این بابت ریسک ظهور تناقض کم بشود؛ چرا که با اینکه مدل ها وابستگی دارند، اما حداقل روابط بین آن ها مشخص هست. در مقاله Paige-Ramsin نیز توضیح مختصر آن ها نشان داده شده است و روابط به طور کامل تعریف شده است. زبان مدلسازی ادعا می کند با ارائه نمودار هایی به مانند نمودار خوشه سعی در تقسیم پیچیدگی دارد و آن را کنترل می کند.

در Booch: آن چیزی که Booch از آن به عنوان مدلسازی استفاده می کرده است شباهت زیادی به UML امروزی دارد. در واقع تعداد کمتری از آن دارد و به این وسیله سعی دارد پیچیدگی را کم و مدیریت کند. قواعد شیءگرای مدل ها کامل تعریف شده ولی به نظر می رسد به مانند نمی تواند نموداری مثل خوشه برای سیستم بدهد که پیچیدگی از این جنبه بشکند.

۲ مقایسه فرآیندی ۳ متدولوژی با Fusion

۱.۲ معیارهای فرآیندی

۱.۱.۲ تعریف

- **چرخه عمر و واحدهای کاری**

مقایسه با RDD: در نظر داریم که Fusion دارای ۳ فاز متوالی تحلیل، طراحی و پیاده سازی می باشد اما RDD کلا دو فاز متوالی کاوش و تحلیل را پوشش داده و تعریف می کند که بسیار کم می نماید و از این نظر متفاوت هستند و Fusion بهتر می تواند پوشش چرخه عمر داشته باشد. البته هیچ کدام فاز نگهداری و یا استقرار را ندارند، اما تحلیل و طراحی را هر دو به صورت کامل تشریح می کنند و متذکر می شوند که داخل هر کدام دقیقا چه گام هایی طی می شود و مدل ها کدامند(که در مقاله Paige-Ramsin مشهود می باشد). نکته ای که در RDD ممکن است فاز دوم آن را از روشن بودن بیاندازد، نامگذاری مبهم آن هست که به تحلیل نامگذاری شده اما در اصل طراحی است.

مقایسه با Booch: در Booch تعریف چرخه ی عمر که ۵ فاز اصلی درشت دانه دارد (و پیشتر آن ها را آوردیم)، به خوبی داخل این فاز های Macro تشریح شده است و روند افزایشی-تکرار شونده ی آن، شفاف تعریف شده است. فرآیند Micro که فعالیت های داخل هر یک بوده از ۴ مرحله تشکیل شده است که تفاوت آن با Fusion بسیار عمیق است؛ چرا که Fusion تنها ۳ فاز اصلی دارد و در هر فاز تعدادی مرحله ی متفاوت گنجانده و تعریف شده است. شباهت این دو اما در پوشش دقیق واحد های کاری در طول متدولوژی می باشد؛ هرچند که Booch تفاوت خود را در تعریف فاز نگهداری نشان می دهد.

مقایسه با OMT: که شامل ۵ فاز اصلی تحلیل، طراحی سیستم، طراحی تفصیلی،

پیاده سازی و آزمون بوده و دقیقا مانند Fusion به خوبی فرآیند های کاری را تعریف می کند. در OMT مانند هر متدولوژی قبلی ذکر شده اینجا نیز، تحلیل و طراحی به خوبی و در حد بالایی پوشا، تعریف شده است(مخصوصا تحلیل و معماری سیستم) و فاز های جداگانه برای آن در نظر گرفته شده است. تعاریف چرخه عمر، روشن، منطقی و جزئی است که تناقض هم برای آن حداقل است.

- **تولیدکنندگان**

مقایسه با RDD: به مانند Fusion صحبتی از نقش افراد درگیر در طول فرآیند نمی شود.

مقایسه با Booch: به مانند Fusion صحبتی از نقش افراد درگیر در طول فرآیند نمی شود.

مقایسه با OMT: به مانند Fusion صحبتی از نقش افراد درگیر در طول فرآیند نمی شود.

- **زبان مدلسازی**

مقایسه با RDD: در هر دو متدولوژی مدل ها به خوبی تعریف شده اند و حتی شباهتی در نمودار توارث بین آن ها یافت می شود. در حقیقت RDD از کارت های CRC و گراف Collaborators استفاده می کند که هیچ کدام در Fusion یافت نمی شود و زبان خود را استفاده می کند؛ اما هر یک برای خود تعریف سرراستی از notation های مدلسازی دارند که منطقی و با جزییات است.

مقایسه با Booch: زبان مدلسازی خود ر ا که بسیار به UML شباهت دارد، استفاده می کند و به خوبی تعریف شده و تمام مدل ها در خدمت فرآیند هستند. پس زبان خود را نیز در تعریف آورده، اما این که معیار های زبان آن چیست، در این بخش بلاموضوع است. در کل، زبان مدلسازی خود را تا سطح قابل قبولی،

روشن و منطقی و با جزییات ارائه داده است، مانند کاری که Fusion انجام داده است.

مقایسه با OMT: نکته ی اصلی در OMT استفاده از مدل سازی وظیفه ای آن به کمک DFD می باشد که متدولوژی های شیءگرا از آن به دور هستند. یعنی تفاوت عمده ی این متدولوژی با OMT در همین هست که نمودار وظیفه مندی غیر شیءگرا معرفی کرده و آن را کامل می کند که میزان سازگار بودن آن با OM و DM می تواند زیر سوال باشد.

- **محصولات**

مقایسه با RDD: محصولاتی که در RDD تولید می شود تعداد بسیار کمتری دارند ولی محصولات Fusion که با درنظر گرفتن محصولات نهایی و میانی آن بسیار زیادتر می باشد؛ هر چند که جزییات دقیق تری را نیز بیان می شود و امکان تبدیل مدل ها، فراهم می شود.

مقایسه با Booch: شباهتی که دو متدولوژی در محصولات دارند این است که محصولات هر یک به طور خیلی خوبی تعریفی شده و تقریبا کامل هستند. تفاوت این دو فقط شاید در تعداد محصولات و مدل های می باشد، اما تعریفی بسیار سازگار دارند.

مقایسه با OMT: محصولات این متدولوژی نیز به خوبی تعریف شده است و مانند Fusion کامل هست. محصولاتی که تکامل ۳ مدل نموداری به علاوه مستنداتحاصل از هر فاز می باشد که در به اصطلاح دیکشنری گردآوری می شوند.

- **تکنیک ها و قواعد**

مقایسه با RDD: به طور دقیق چگونگی اجرای واحد های کاری در RDD مشاهده می شود، کما این که در Fusion نیز چنین است. دقت داریم که تکنیک ها و

قواعد در Fusion البته کمی پیچیده تر هست، اما تفاوتی از جنبه ی ناقص بودن ندارند.

مقایسه با Booch: در Booch دقیقا بیان شده است که جز اینکه مقصود از هر واحدکاری چیست، چگونه آن واحد کاری انجام می شود. البته نیز Fusion عملکرد مشابهی دارد و به طور دقیق ذیل هر واحد کاری، چگونگی وصول آن کار را تصریح می کند. از این نظر دو متدولوژی شباهت دارند.

مقایسه با OMT: متدولوژی OMT نیز تصویر درستی از نحوه کار در هر واحد کاری را با جزییات بیان می کند، به خصوص آن که برای فاز طراحی تفصیلی، گام های خوبی را متذکر می شود و از این جهت می توان شباهت آن با Fusion را در ذکر قابل قبول دقیق با جزییات برای تکنیک ها و قواعد در روال هر فاز عنوان کرد.

- **موارد مربوط به فعالیت های چتری**

مقایسه با RDD: توجه داریم که فعالیت های چتری هیچ گونه تعریفی در RDD ندارند و در Fusion نیز تا حدودی چنین هست. به طور کلی کار ها مدیریتی در طول کل چرخه عمر و ناظر بر کل فرآیند جایی در این متدولوژی ها ندارد. البته ذکر این نکته خالی از لطف نیست که Fusion با اندک قواعد ثوری خود، سعی در ایجاد ضمانت کیفیت دارد که از این نظر متفاوت است.

مقایسه با Booch: تفاوتی که میان این دو وجود دارد این است که Fusion نمی تواند راهکار های مربوط به فعالیت چتری را از خود نشان دهد، در حالی که Booch می تواند آن ها را تعریف کند و با رویکرد افزایشی-تکرار شونده ی خود در فرآیند ها سعی در کنترل و کاهش ریسک، کارهای مدیریتی و تضمین کیفیت دارد.

مقایسه با OMT: و OMT هم نمی تواند پاسخی به فعالیت های چتری بدهد

تا از این جهت تقریبا مانند Fusion عمل کرده باشد. هیچ تعریف صریحی از جزییات ریسک یا مدیریت و یا حتی تضمین کیفیت در OMT مشاهده نمی شود(اما OMT کمی بهتر عمل می کند کما اینکه در بالاتر آمد).

- **نحوه تعریف کردن که با ارجاع به مقاله Paige-Ramsin آورده شده اند**

مقایسه با RDD: تعریف فرآیند هم در RDD و هم در Fusion به صورت process-centered می باشد.

مقایسه با Booch: تعریف فرآیند هم در Booch و هم در Fusion به صورت process-centered می باشد.

مقایسه با OMT: تعریف فرآیند هم در OMT و هم در Fusion به صورت process-centered می باشد.

۲.۱.۲ پوشش چرخه عمر عمومی ایجاد نرم افزار

- **تعریف مساله**

* کاوش قلمرو مساله و مدلسازی آن

مقایسه با RDD: این متدولوژی از نظر کاوش در قلمرو مساله، فاز اول خود را در اختیار گذاشته است که در آن کاملا سعی دارد تا کلاس ها، مسئولیت ها و همکاران کلاس را بدست بیاورد تا مساله را به خوبی بکاود اما Fusion در فاز اول خود موسوم به تحلیل، با ارائه ی یک محیطی که فعلا مرز سیستم در آن مشخص نیست، سعی می کند مدلی از نمودار کلاس بدست بیاورد و با طراحی مدل های میانی، مرز سیستم را مشخص کند و در این فاز، فقط از مرز سیستم و نه داخل آن، مدل رفتاری و وظیفه ای را نیز ارائه می دهد که در RDD چنین نیست و تمرکز از همان اول بر روی تحلیلی چیستی و وظایف سیستم با پخش کردن کار ها در کلاس ها می باشد و مسئولیت های سیستم را در کلاس ها منعکس می کند و مرز را از همان اول ارائه می دهد و در نمودار کلاس خود، خیلی کم، مدلی از وظیفه مندی را به نمایش می گذارد ولی مدل رفتاری خیر.

مقایسه با Booch: متدولوژی Booch در یک Macro ی خود به طور کلی سعی می کند که در قلمرو مساله به خوبی کاوش کند و در این راه تکنیک ها را نیز بازگو کرده و اینکه داخل آن چه اتفاقاتی می افتد که نیز به Micro ها بر می گردد. در حقیقت Booch به مانند Fusion عمل نمی کند و به دنبال ابتدا مرزبندی سیستم نیست و در همان تحلیل، چیستی سیستم را به دقت کشف می کند. این که چه کلاس هایی در درون سیستم وجود دارند و الگوهای رفتاری آبجکت ها در جهت تحقق نیازمندی ها را بازنمایی می کند که تفاوت آن با Fusion در این جا هست که Fusion

در سطح تحلیل خود به مدلسازی رفتاری و وظیفه ای درون چیستی سیستم کاری ندارد و فقط به مرز سیستم نگاه می کند و نمودار کلاس خود را حتی فاقد operation باقی می گذارد.

مقایسه با OMT: این متدولوژی نیز با Fusion تفاوت دارد و در تحلیل خود هر سه نوع مدلسازی را تا حد خوبی پیش می برد تا چیستی سیستم را دقیقا متوجه شود. مرز سیستم و درون سیستم با OM و DM و FM وقتی که مساله ی دنیای واقع به خوبی قابل فهم و بدون ابهام شد، مدلسازی می شود که باز هم از این نظر با Fusion متفاوت هست. البته شباهت Fusion با OMT در این هست که در تحلیل به رفتار و وظیفه توجه می شود، اما Fusion برای رفتار و وظیفه به داخل سیستم کاری ندارد.

* استخراج نیازمندی ها

مقایسه با RDD: هیچ فرآیندی در این زمینه در RDD تعریف نشده است و سند نیازمندی های آماده، قبل از شروع متدولوژی فراهم است و ورودی به فاز اول آن است که دقیقا همان مفروض Fusion می باشد.

مقایسه با Booch: تفاوتی که Booch با Fusion دارد در همین استخراج نیازمندی است که این متدولوژی در اولین فرآیند درشت دانه ی خود، به آن می پردازد اما Fusion استخراج نیازمندی را در نظر نمی گیرد و فرض می کند که نیازمندی ها را گروهی دیگر، طی روشی استخراج کرده اند و حال به عنوان ورودی به متدولوژی می باشد.

مقایسه با OMT: در نهایت این که OMT حرکت مثبتی که از خود نشان می دهد و کمی اوضاع استخراج نیازمندی ها را به نسبت Fusion بهبود می بخشد این هست که در فاز اول خود یعنی تحلیل با مصاحبه ها و پرسشنامه ها و... سعی در تعامل دارد تا نیازمندی ها را به خوبی شناسایی کند. یعنی آن که متدولوژی از یک مجموعه اولیه دانش شروع شده و در

جریان تحلیل این دانش را بیشتر کسب می کنند و به این ترتیب نیازمندی را تا درصد بالایی سعی در استخراج دارد؛ که تفاوت خود را با Fusion به خوبی نشان می دهد.

* تحلیل امکان پذیری

مقایسه با RDD: هیچ برنامه ای برای امکان سنجی در فرآیند متدولوژی RDD تعبیه نشده است که از این نظر با Fusion شباهت دارد. مقایسه با Booch: کار امکان سنجی خیلی خاصی در Booch تعبیه نشده است(فقط می توان برای آن اندکی از تحلیل امکان پذیری را به دلیل رویکرد افزایشی- تکرارشونده ش متصور شد اما تصریحی ذکر نشده است)، کما اینکه در Fusion نیز مشاهده نمی شود. مقایسه با OMT: متدولوژی OMT نیز تحلیل امکان سنجی را به مانند Fusion در دستور کار ندارد.

- **ایجاد**

* طراحی معماری

مقایسه با RDD: این متدولوژی می تواند پاسخ درستی برای طراحی معماری داشته باشد؛ چرا که در فاز اول خود وقتی کلاس ها به درستی شناسایی شده اند و مسئولیت ها و همکاران تشریح شده اند، خروجی به فاز دومی می رود که سعی در طراحی معماری دارد و سعی می کند که در یک فاز با مشخص کردن subsystem ها و تعمیم دادن ها در مرحله ی آخر اصرار به بیان مشخصات دقیق کلاس ها و subsystem ها داریم(طراحی تفصیلی نیز انجام می شود). از این نظر Fusion نیز در فاز دوم خود، تمام کار های مربوط به طراحی خود را می آورد و به توارث و تعمیم سازی به همراه ذکر مشخصات کلاس ها به مانند RDD اهمیت می دهد. از این رو، دو متدولوژی توانایی خوبی در پوشش طراحی معماری سیستم دارند.

مقایسه با Booch: متدولوژی Booch همانطور که پیشتر نیز آمد، در فعالیت درشت دانه ی دوم خود طراحی معماری را به خوبی مشخص می کند و آن را پوشش می دهد و تعریف آن مانند Fusion دقیق و تقریبا کامل است.

مقایسه با OMT: متدولوژی OMT فاز دوم خود را اختصاص به تکمیل ۳ مدل خاص خود می دهد به طوری که بسیار خوب و دقیق آن را تعریف کرده و مشخصا ذکر می کند که هدف در آن چیست. یعنی در قلمرو جواب، بدنبال راه حل هست و یکی از مهمترین کار هایی OMT همین است که این مرحله رو به شکل مرحله جدا با همه جزییات تعریف کرده است و طراحی معماری را از طراحی تفصیلی جدا کرده است، تفاوتی که در Fusion پیدا نمی شود.

* طراحی با جزییات

مقایسه با RDD: در واقع RDD طراحی با جزییات خود را در همان فاز طراحی سعی بر اجرا دارد که از این نظر مانند Fusion عمل می کند و فاز جداگانه ای برای آن ندارد.

مقایسه با Booch: متدولوژی Booch نیز در طراحی معماری خود به طراحی نفصیلی اهتمام می ورزد که شباهتی به Fusion دارد.

مقایسه با OMT: اما OMT توانایی دارد که فاز طراحی تفصیلی خود را جدا بیان کند و در آنجا، به کلاس های قلمرو مساله که بدست آمده اند، کلاس های قلمرو جواب پدیدار شده را اضافه می کند و جزییات را بسیار دقیق می کند. تمام attribute ها و یا operation ها با امضای کامل، کلاس های مرتبط با DB و یا اینترفیس ها و... همگی در این مرحله به قدری کامل می شود که آماده ی تحویل به برنامه نویس می شود که Fusion ضعف خود را برای نداشتن چنین فاز مجزایی نشان می دهد.

* برنامه نویسی

مقایسه با RDD: هیچ گونه فازی برای پیاده سازی در RDD درج نشده است در حالی که Fusion به برنامه نویسی در فاز آخر خود دقت می کند.

مقایسه با Booch: در Booch شاهد یک فعالیت درشت دانه برای برنامه نویسی و پیاده سازی هستیم که از این نظر به مانند Fusion پوشش خوبی فراهم کرده است.

مقایسه با OMT: متدولوژی OMT نیز مانند Fusion توانسته است فازی برای پیاده سازی در نظر بگیرد و پوشش این واحد کاری چرخه ی ایجاد نرم افزار را فراهم کند.

* آزمون

مقایسه با RDD: هیچ پوششی برای آزمون محصول در RDD گنجانده نشده است که از این نظر به مانند Fusion می باشد.

مقایسه با Booch: آزمون در متدولوژی Booch صراحتی برای یک واحد کاری ندارد اما در ذیل تکامل و پیاده سازی در برنامه نویسی تعبیه می شود که برتری اندکی نسبت به تفاوت آن با Fusion برای این متدولوژی دارد.

مقایسه با OMT: متدولوژی OMT فاز آزمون را در آخرین فاز خود تعریف می کند، جایی که با تست سناریو شامل سناریوهای تست و تست‌کیس ها، آزمون را بر روی محصولات انجام داده و نتایج را در دیکشنری متدولوژی ذخیره می کند. با این کار برتری خود را نسبت به Fusion که جایی برای آزمون ندارد، نشان می دهد.

* مستقرسازی

مقایسه با RDD: در Fusion و RDD هیچ توجهی به مستقرسازی در محیط کاربر نشده است.

مقایسه با Booch: مستقر سازی برای Booch صراحت ندارد اما ذیل تکامل و پیاده سازی انجام می شود که با این حرکت، برتری اندکی نسبت به تفاوت آن با Fusion برای این متدولوژی دارد.

مقایسه با OMT: متوجه هستیم که OMT نیز کاری به استقرار در محیط کاربر ندارد و شبیه به Fusion می باشد.

- **نگهداری**

مقایسه با RDD: نکته ای که Fusion و RDD در آن شباهت دارند این هست که هر دو، فاز یا مرحله ای برای نگهداری محصول قائل نشده اند.

مقایسه با Booch: تفاوتی که برای Booch مزیت حساب می شود، توجه ویژه ی آن به نگهداری پس از ترخیص محصول هست که برای آن فاز Macro درشت دانه در نظر گرفته شده است که Fusion از آن بی بهره می باشد. نکته در Booch این هست که با روال افزایشی-تکرار شونده ی خود وزن فرآیند را می تواند جابجا کند و باعث شود در مقاطعی، توجه ویژه ای به نگهداری داشت که در نتیجه غنی تر از Fusion می نماید.

مقایسه با OMT: این متدولوژی و Fusion نیز هیچکدام به نگهداری توجهی نداشته اند.

۳.۱.۲ پشتیبانی از فعالیت های چتری

- **مدیریت ریسک**

مقایسه با RDD: متدولوژی RDD نمی تواند از تکنیکی برای مدیریت ریسک بهره ببرد و کار خاصی را در این زمینه انجام نمی دهد، ولی Fusion با ایده ی check list ها در گذار از فاز به فاز، مدل ها را از نظر سازگاری و تکامل بررسی می کند تا ریسک کاهش یابد(که به تکنیک continuous verification و continuous validation مربوط می شود).

مقایسه با Booch: مهم ترین نکته و تکنیکی که Booch برای کاهش ریسک ارائه می دهد استفاده از روش افزایشی-تکرار شونده می باشد که امکان بازنگری و تکامل تدریجی را میسر می کند تا کار های پر ریسک را مدیریت شده پیش ببریم. تفاوت مهم آن با Fusion نیز همین هست که Fusion چنین تکنیکی ندارد و فقط برای پایداری مدل های خود مجبور به روند تکرار شونده می شود.

مقایسه با OMT: تنها نکته ی مثبت مدیریت ریسک در OMT استفاده از تکنیکی می باشد که نیازمندی کشف معماری سیستم را بسیار دقیق و زود بیان می کند تا بتواند بر مبنای ریسک، برنامه ریزی انجام دهد و نیاز با ارزش تر را زودتر انجام دهد(تقریبا تکنیک risk-based palnning می باشد). این در حالی است که Fusion فقط می تواند با بررسی check list ها و استفاده از روش ثوری در مدیریت ریسک، روش ارائه دهد.

- **مدیریت پروژه**

مقایسه با RDD: هرچند RDD توانسته است واحدهای کاری پروژه ای را از نظر رابطه ی پیش نیازی-هم نیازی تعریف کند اما هیچ planning و scheduling ی را در بر نمی گیرد و در نتیجه monitoring نیز ندارد؛ درست به مانند Fusion که روش خاصی برای مدیریت پروژه ندارد.

مقایسه با Booch: مدیریت پروژه در Booch کمی قوی تر دیده شده است و به نسبت Fusion توانایی بهتری در آن دارد؛ چرا که به کامل بودن و زمان بندی تاکید دارد، روابط بین فرآیند ها به درستی تعبیه شده است و به دلیل این که اجرای فرآیند Macro را به ماه ها نسبت می دهند، می شود برنامه ریزی را برای Macro ترتیب داد و Micro ها نیز که روزانه هستند و برنامه مدیریتی آن مشخص است.

مقایسه با OMT: صراحتی برای کارهای مدیریتی در OMT پیش بینی نشده است(تنها ترتیب روابط فعالیت ها در آن ترسیم شده است) و از این نظر با Fusion شباهت دارد.

- **تضمین کیفیت**

مقایسه با RDD: متدولوژی RDD البته تصریحی در تضمین کیفیت ندارد و تکنیک خاصی ارائه نداده است اما مثل Fusion در مدل های خود سعی در رسیدن به پایداری دارد که می تواند اندک تاثیری در تضمین کیفیت داشته باشد. البته Fusion به کمک روش های ثوری خود، ضمانت کیفیت را بهتر می تواند صادر کند.

مقایسه با Booch: مزیت بالای Booch به نسبت Fusion در این هست که با بازنگری فنی مکرر خود که در رویکرد افزایشی-تکرار شونده به وقوع می پیوندد، می تواند کیفیت بالایی را تضمین دهد که تفاوت اصلی آن با Fusion می باشد.

مقایسه با OMT: این متدولوژی فعالیت خاصی را برای بهره گیری از تضمین کیفیت پیش بینی نکرده است. تصریحی برای طراحی براساس قرارداد ندارد، استراتژی خاصی برای بهبود traceability ندارد و نمی تواند بازنگری های فنی مکرر داشته باشد(تنها مدل هی خود را گام به گام تکمیل می کند که تاثیر کمی در تضمین کیفیت دارد)؛ در حالی که Fusion با بهره گیری از روش های ثوری به تضمین کیفیت نزدیک تر است.

۴.۱.۲ بی درزی و هموار بودن انتقال

- **بی درزی**

مقایسه با RDD: متدولوژی RDD نه در فعالیت ها درز دارد و نه در زنجیره مدلسازی. به نظر می رسد که تکالیف و همچنین مصنوعات در آن بر روی آبجکت(کلاس) بیان شده و مفهوم مشترک آن می باشد. در حالی که در Fusion نیز ما شاهد بی درزی هستیم، چرا که انتقال از فاز ها و فعالیت ها بدون پارادایم شیفت و بدون تفاوت الگو های ادراک می باشد و اطلاعات خیلی کمی فقط ممکن است در تحلیل مغفول بماند که به گذار ناهموار منجر می شود. در حقیقت، Fusion مفهومی که دنبال می کند، operation و دید بین‌شیءای می باشد.

مقایسه با Booch: متدولوژی Booch نیز با مطرح کردن کلاس و آبجکت به بی درزی می رسد و از پارادایم شیفت بین مدل ها جلوگیری می کند. متدولوژی های Booch و Fusion هر دو بی درز هستند.

مقایسه با OMT: متدولوژی OMT بسیار در بی درزی جالب عمل کرده و با آنکه مانند Fusion بدون درز هست، اما مفهوم مشترکی را در دستور کار قرار نداده است؛ بله که یک مجموعه خاصی از مدل ها را دارد و همان ها را تکامل می دهد و تکالیف حول ایجاد همین مجموعه است، به عبارت دیگر محوریت بر روی مجموعه ثابتی از مدل ها می باشد که همان مدل ها در طول فرآیند تکامل می یابند.

- **هموار بودن انتقال**

مقایسه با RDD: در Fusion از نقاط قوتی که برایش مطرح می شود این است که گام به گام و فاز به فاز گذار تقریبی همواری داریم، یدین طریق که ساخت مدل های خیلی سنگین نداشته ولی جایی به مشکل می خورد که مدل های جدیدی باید در فاز های بعدی تصویر شوند؛ که گذار هموار را دچار مشکل می کند ولی

در کل نمودار ها توالی منطقی دارند و مهم تر این که بی درز هست که به گذار هموار بودن کمک می کند. این در حالی است که RDD نیز گذار های هموار دارد و برای ۲ فازی که دارد، تنها سعی در ساخت نمودار های جدید از خروجی های فاز قبل داشته و تکمیل ها را انجام می دهد و از این نظر شبیه به Fusion عمل می کند.

مقایسه با Booch: دوباره تاکید می شود که هرچند بی درزی ممکن است در متدولوژی رخ دهد اما Booch با رویکرد افزایشی-تکرار شونده ای که دارد، هموار انتقال بین گام ها را به خوبی مهیا کرده است. از این حیث Fusion شاید به خوبی Booch عمل نکند اما در هر صورت گذار به نسبت خوبی را فراهم کرده است؛ چرا که در Booch و در فرآیند Micro آن، شاهد محصول کاملا جدیدی نیستیم و مدام مدل ها عوض نمی شوند؛ تنها به شکل تکاملی، بازنگری و بروزرسانی می شوند. در خود Macro ها نیز فرآیند انتقال نرم صورت می پذیرد، چون در هدف هر Macro خواسته هایی تامین شده که ورود به Macro ی بعدی را به گونه ای مهیا می سازد که ورودی های خوبی برای تکامل فراهم گریده است.

مقایسه با OMT: در OMT کلا انتقال به شکل خوبی هموار است، چرا که هیچ محصول کاملا جدیدی در طول اجرای متدولوژی ایجاد نمی شود و از فاز به فاز با تغییر مدل مواجه نیستیم. در این مقایسه، متدولوژی OMT خیلی بهتر از Fusion عمل می کند.

۵.۱.۲ مبتنی بر نیازمندی وظیفه ای و غیر وظیفه ای

مقایسه با RDD: نیازمندی های غالبا متنی و وظیفه ای که متدولوژی را شروع می کنند، اساس کار در تحلیل قرار می گیرند تا کلاس ها، مسئولیت ها و همکاران مشخص شوند و بدین ترتیب، نیازمندی های مساله را در مسئولیت آبجکت ها نمود می دهند. پس RDD مبتنی بر نیازمندی وظیفه ای است اما نمی تواند کنترلی بر روی نیازمندی های غیروظیفه ای داشته باشد. در نظر داریم Fusion دقیقا به مانند RDD می باشد. هیچ کدام، کاری به ثبت و استخراج نیازمندی ندارند، بلکه قبل از شروع فرآیند با موجود شده باشند؛ به تنهایی و به صورت مستقل، مدلی از آن ها در دسترس نمی باشد(جز این که از قبل در شکلی مهیا شده اند که متدولوژِ به آن کار ندارد و از محصولاتش نیست) و نیازمندی از روی مدل ها برداشت می شوند. فقط آن که در هر دو متدولوژی، مبنای فاز اول می باشد.

مقایسه با Booch: در متدولوژی Booch اتفاق ثبت و استخراج نیازمندی در فاز Macro ی اول، نقطه ی قوت متدولوژی به نسبت Fusion هست، مدلسازی نیازمندی های هسته و اصلی ولی بدون مدلسازی خاصی مانند usecase و صرفا روایی دوباره شباهت را با Fusion می رساند. در نهایت این که Booch مثل Fusion در فاز بعدی خود مبتنی بر نیازمندی عمل می کند. از نیازمندی های غیروظیفه ای، مهم ترین آن یعنی نگهداری را پوشش می دهد به طوری یک فعالیت درشت دانه را به آن اختصاص می دهد.

مقایسه با OMT: ویژگی متفاوت OMT این هست که فرآیند با اندک دانشی شروع می شود اما با تعامل با کاربر و پرسشنامه ها و مطالعه ی عمیق تر به همراه مصاحبه، اطلاعات مربوط به نیازمندی استخراج می شود و چیستی سیستم بیشتر نمایان می شود. مدلسازی خاصی برای نیازمندی ها به صورت جداگانه ذکر نشده اما چون مفاهیم قلمرو مساله از آن بدست می آید، مبنای تحلیل بوده و از این دو منظر به شکل Fusion عمل می کند. OMT عمل خاصی برای مبتنی بودن بر نیازمندی غیر وظیفه ای ارائه نمی کند.

۶.۱.۲ قابل آزمون، ملموس و قابل رهگیری به نیازمندی ها

- **آزمون پذیری**

مقایسه با RDD: دقت داریم که پیچیدگی محصولات و تعداد مصنوعات در RDD خیلی کمتر از Fusion می باشد، سادگی و قابلیت فهم آن ها خیلی بهتر در قیاس با Fusion و یکپارچگی و جامع تر به نسبت این که همدیگر را کامل می کنند، اما Fusion وابستگی زیادتری بین محصولاتش هست و در تغییر فاز تحلیل به طراحی، تکامل نمودار کلاس مرزبندی صورت نمی گیرد و باعث شکستگی می شود که در نتیجه آزمون پذیری را کم می کند. در کل RDD از Fusion آزمون پذیرتر است. لازم به ذکر هست که Fusion فهم محصولاتش به نسبت سختتر است اما همچنان به نسبت امروزه، قابل فهم می باشند.

مقایسه با Booch: متدولوژی Booch محصولاتی قابل فهمی دارد که ساده هستند، تعداد آن ها به نسب کم و البته تکامل مروری آن ها بسیار به آزمون پذیری آن کمک کرده و به نسبت Fusion آزمون پذیرتر می باشد. دقت داریم که وابستگی در Fusion خیلی شدید تر بوده به طوری که در ۴ مدل اصلی فاز طراحی آن، هر تغییری مانند تغییر در توارث، باعث انتشار تغییر در ۳ مدل دیگر می شود که آزمون پذیری را کاهش می دهد.

مقایسه با OMT: این متدولوژی بسیار آزمون پذیر عمل می کند و محصولات قابل آزمونی ارائه می دهد، چرا که در کل ۳ مدل هستند و تعداد خیلی پایین تری به نسبت Fusion دارد، محصولات اصلا پیچیده نیستند و به خوبی تعریف شده اند، قابل فهم و با وابستگی های حداقلی که مرسوم هست، در حالی که در Fusion این اوضاع در مقام مقایسه بدتر می باشد. در OMT محصولات به مرور کامل شده و همدیگر را نیز کامل می کنند؛ که همه ی این ها باعث آزمون پذیری بسیار خوب OMT می شود.

- **ملموس بودن**

مقایسه با RDD: در RDD ملموس بودن محصولات بسیار واضح است. وجود کلاس ها به همراه شناسایی مسئولیت ها و همکاران، سپس توارث و زیرسیستم ها و در نهایت contract ها، همگی خیلی شفاف می توانند در پیاده سازی کمک کنند و مستقیم به کار روند. این در حالی است که Fusion نیز می تواند سطح خوبی از ملموس بودن برای محصولات خود متصور شود. همگی برای توسعه دهنده مشهود است که دلیل وجودی آن چیست و این که چه گونه روی دیگری اثر می گذارد.

مقایسه با Booch: متدولوژی Booch محصولات ملموسی را در بر دارد، فقط می توان مدلسازی نیازمندی های وظیفه ای آن را زیر سوال برد که شفافیت درستی برای توسعه دهنده می تواند نداشته باشد؛ چون روایی بیان می شود. در کل سطح خوبی از ملموس بودن را دارد که بی شباهت با Fusion نیست.

مقایسه با OMT: ملموس بودن در OMT بسیار وضوح دارد. محصولات چه اسناد هر فاز و چه تکمیل سه مدل، بسیار توجیه دارند. این که نسخه های مختلف اسناد، سورس کد و نتایج آزمون و کتابخانه کتابخانه ی کلاس ها، در دیتابیس گردآوری شود، ملموس بودن متدولوژی را برای توسعه دهنده و کاربر در محصولات می رساند. تنها نکته ی منفی در استفاده از DFD هست که خیلی استفاده ای ندارد و وجود آن، توجیه به خصوصی ندارد و باعث کاهش ملموس بودن محصولات می شود. لازم به ذکر است که Fusion نیز به خوبی OMT در ملموس بودن می تواند عمل کند.

- **قابل رهگیری به نیازمندی ها**

مقایسه با RDD: در RDD کلا تصریحی برای ارجاع مستقیم از محصولات به نیازمندی ها وجود ندارد و حتی سناریوهایی نیز برای ارزیابی مبتنی بر نیازمندی

که بشود تناظر را نشان داد، وجود ندارد. تنها به صورت غیر مستقیم می توان به کمک کلاس ها و مسئولیت هایی که برای آن ها می شود تعریف کرد(در کارت های CRC موجود)، به نیازمندی های تعریف شده رسید. در واقع RDD در مقایسه با Fusion اما ضعیف عمل می کند. در حقیقت Fusion خیلی در این معیار خوب عمل می کند. با سناریو های استفاده از سیستم در scenario transaction امکان رهگیری به نیازمندی ها به خوبی فراهم است. چون که متدولوژی operation-driven تعریف شده است، می توان operation هایی که تعریف شده اند را چک کرد و مورد رهگیری به نیازمندی قرار داد؛ بدین طریق که به گراف interaction تبدیل شوند و سپس بر همین اساس پیاده سازی شوند، درنهایت با transaction scenario تحقق نیازمندی بررسی می شود.

مقایسه با Booch: متدولوژی Booch مبتنی بر نیازمندی ای می باشد که در فاز اول Macro ی خود تا درصد بالایی از نیازمندی را کشف کرده است. اما مستندی که درست می کند مبتنی بر نمودار یا مدل خاصی مثل usecase نیست و روایی بیان می شود که نقطه ی منفی رهگیری به نیازمندی ها خواهد بود. از وجود سناریوهای تحقق هم تصریحی ذکر نشده است و بنا بودن آن بر نیازمندی، تنها به رسیدن غیر مستقیم به نیازمندی های متنی کمک می کند و امکان تحقق مستقیم را فراهم نمی کند که در این زمینه در مقایسه با Fusion ضعیف تر عمل می کند.

مقایسه با OMT: این که در فاز آخر خود، سورس کد را مورد ارزیابی قرار می دهد، ما را به طور غیر مستقیم به تحقق نیازمندی می رساند، چرا که متدولوژی، requirements-driven نبوده و تنها requirements-based هست و سناریوهای تحقق نیازمندی وظیفه ای نیز در آن به چشم نمی خورد؛ پس می توان گفت که Fusion خیلی بهتر می تواند در رهگیری به نیازمندی ها عمل کند.

۷.۱.۲ دخالت فعال کاربر

مقایسه با RDD: در نظر داریم که در Fusion چنین تصریحی که نماینده ی کاربری، عضوی از تیم ایجاد باشد و یا جلسات برنامه ریزی و مرور و بازنگری با شرکت کاربران یا متخصصین حوزه انجام بشود، ذکر نشده است، پس تشویق به دخالت مستقیم و فعالانه کاربر در Fusion منتفی است، کما اینکه RDD نیز به همین منوال عمل می کند و با هیچ روشی، دخالت مستقیم و فعالانه ی کاربر را مهیا نمی کند.

مقایسه با Booch: متدولوژی Booch با رویکرد مهم خود که افزایشی-تکرارشونده می باشد، می تواند با جلسات دوره ای خود و به کمک متخصصین حوزه، دخالت مستقیم کاربر را تا حدود بهتری به نسبت Fusion فراهم کند(هرچند که تکرار می شود صراحتی در این باب برای Booch که حتما نماینده ای از سمت مشتری باشد، وجود ندارد).

مقایسه با OMT: در فاز اول OMT که فاز تحلیل می باشد، تعامل خوبی با کاربر و مشتری ها صورت می گیرد تا دانش پیرامون قلمرو مساله افزایش یابد. این تعامل و دخالت کاربر از طریق مصاحبه و پرسشنامه انجام شده و فعالانه می باشد. بدین منظور نیازمندی های بیشتری شناسایی شده و از این حیث در این گام، بسیار بهتر از Fusion عمل می کند؛ هرچند که در گام های بعدی خود، صراحتی برای دخالت فعالانه کاربر متذکر نشده است.

۸.۱.۲ قابل اجرا بودن و استفاده کارآ

- **قابل اجرا بودن**

مقایسه با RDD: شاید بتوان گفت که RDD بسیار ساده تر از Fusion باشد، زیرا هم فاز های کمتری دارد و هم از پیچیدگی فرآیندی کمتری برخورددار است. البته هر دو قابل اجرا می باشند ولی به نظر اجرا پذیری RDD بهتر از Fusion باشد. به این منظور می دانیم که Fusion پیاده سازی را نیز در بر می گیرد در حالی که فرآیند RDD کاری به برنامه نویسی ندارد.

مقایسه با Booch: اما Booch نیز به مانند Fusion از اجرا پذیری خوبی برخورددار می باشد. پیچیدگی فرآیندی در آن کم و قابل اجرا برای پروژه ها ماهیت سنگین و نیازمندی نگهداری می باشد. شیوه ی افزایشی-تکرارشونده ی آن به اجرا شدن بهتر کمک کرده و مانند Fusion از اجرا پذیری خوبی برخورددار می باشد.

مقایسه با OMT: اینکه بتوان فرآیند را در OMT اجرا کرد و چه قدر قابل اعمال هست، به نظر بسیار بالا می باشد؛ چرا که پیچیدگی فرآیند آن بسیار پایین بوده و فاز ها و مراحل به خوبی هر چه تمام تر و با جزییات، ساده بیان شده اند. برای همین می توان گفت به تمامیت قابل استفاده بوده که حتی NASA از این متدولوژی بهره مند شده است. از این نظر در مقایسه با Fusion شباهت دارد؛ زیرا این متدولوژی نیز فرآیند ساده ای تعریف کرده و مدل ها را به خوبی با فعالیت های میانی، تعریف می کند و می سازد و همچنین سنگین نبوده و تمامیت آن قابل استفاده است.

- **کارآ بودن**

مقایسه با RDD: میزان کارآیی Fusion در حوزه کاربردی که دارد به نسبت بالا می باشد. چون که هم اجزای فرآیندی خیلی پیچیده ای ندارد که قابل درک نباشد

و هم وظایفی که حواس ایجاد کننده را از فعالیت اصلی منحرف کند، ندارد. بدین منظور سعی دارد در فاز معماری خود، معماری سیستم را کاوش کند و وابستگی ای به ابزار خاصی نداشته باشد. در کل متدولوژی کارآیی می باشد که از این نظر با RDD شباهت دارد. چرا ک RDD نیز اجزای فرآیندی پیچیده ای نداشته و سعی در کشف معماری سیستم دارد و ابزار خاصی را وابسته نمی داند. در هر دو البته کمی مدیریت پروژه ناکافی می نماید.

مقایسه با Booch: متدولوژی Booch از این نظر اوضاع بهتری نسبت به Fu- sion دارد؛ زیرا هم پیچیدگی اجزای فرآیند آن کمتر می باشد و هم با بکارگیری جلسات منظم و دوره ای، سعی در حفظ تمرکز خواهد داشت. تمام سعی خود را برای دوری از استراتژی های خطا خیز می کند و بدین منظور مدل های خود را بی درز معرفی می دهد. سپس به طور مشخصی به ابزار یا تکنولوژی خاصی خود را محدود نمی کند و در نهایت با استراتژی های مدیریتی مثل تکراری-افزایشی تاثیر شگرفی را بر روی کارآیی خواهد گذاشت.

مقایسه با OMT: نکته ی منفی که در OMT وجود دارد و کارآیی را ضعیف می کند استفاده از مدل DFD در تمام فاز های متدولوژی می باشد. در واقع استفاده از یک استراتژی خطا خیز که منجر به بیهوده شدن وقت و انرژی می شود و استفاده ی زیادی از آن در تفکر شیءگرا نخواهد شد. البته جز این، به دلیل پیچیده نبودن اجزای فرآیند، کارآیی خود را بهبود می دهد و سعی در کشف معماری سیستم در دو فاز مجزا دارد که بسیار کمک به تمرکز حواس می کند و در کل سطح خوبی از کارآیی را مانند Fusion می دهد.

۹.۱.۲ قابلیت مدیریت پیچیدگی

مقایسه با RDD: با در نظر گرفتن این که Fusion فرآیند خود را با ۳ فاز سعی دارد ساده سازی کند(استفاده از روش partitioning برای مدیریت پیچیدگی و شکستن کل پیچیده فرآیند) که در این باب، بهتر از RDD عمل کرده است؛ چرا که RDD تنها ۲ فاز کاوش(طراحی) و تحلیل(طراحی) را تعریف کرده است. البته قابلیت فهم در هر دو در سطح مناسبی بالا می باشد و کلا RDD پیچیدگی فرآیند زیادی ندارد. لازم به ذکر است که دو متدولوژی RDD و Fusion فاز هایی دارند که در هر یک، ریزدانه های فعالیت پوشش داده شده است.

مقایسه با Booch: متدولوژی Booch در مدیریت پیچیدگی خیلی خوب عمل کرده و ۵ فاز درشت دانه تکرار شونده ی خود را تعریف کرده تا تمام فرآیند آن، قابل فهم تر بوده و از پیچیدگی بیافتد. در هر کدام به ریزجزئیات(در ۴ مرحله) توانسته، فعالیت های تکرارشونده را مشخص کند و به دلیل تکرار شدن فرآیند و تغییر تمرکز در فاز ها، می توان گفت که قابل مدیریت تر از Fusion می باشد.

مقایسه با OMT: توجه داریم که OMT فرآیند خود را به ۵ فاز مجزا تقسیم کرده و کل پیچیده ی خود را با این قسمت های تشکیل دهنده(طراحی، معماری سطح بالای سیستم، معماری تفصیلی سیستم، برنامه نویسی و آزمون) قابل مدیریت می سازد. سطح بسیار خوبی از لایه بندی به سطح جزئی را شاهد هستیم که نمونه ی آن در توضیح ذیل فاز سوم، قابل درک می باشد. در کل OMT نیز به مانند Fusion به خوبی توانسته است که فرآیند خود را از نظر پیچیدگی مدیریت کند.

۱۰.۱.۲ قابل گسترش، مقیاس پذیر، قابل پیکربندی، انعطاف پذیر

- **گسترش پذیری**

مقایسه با RDD: از آنجایی که تعریف یک فرآیند به صورت هسته ی قابل گسترش در Fusion تصریح نشده است و نمی توان قابلیت هایی به آن برای پروژه های خاص متصور شد، پس از نظر گسترش پذیری، Fusion ضعیف عمل می کند. همچنین RDD دقیقا از این نظر به Fusion شباهت دارد.

مقایسه با Booch: متدولوژی Booch نیز به مانند Fusion نمی تواند برای گسترش پذیری راه حلی ارائه بدهد. مقایسه با OMT: دقیقا OMT نیز از نظر گسترش پذیر نبودن به Fusion شباهت دارد.

- **مقیاس پذیری**

مقایسه با RDD: متدولوژی RDD به حد Fusion مقیاس پذیر نیست. زیرا بدلیل نداشتن تکنیک های صوری، نمی تواند سطح بحرانیت بالا را به خوبی پشتیبانی کند. از طرفی اما به دلیل کم بودن فاز ها، پروژه های با اندازه ی بزرگ برای آن کمتر موضوعیت دارد و در تعریف امروزه، گستره مقیاس پذیری کمی دارد.

مقایسه با Booch: در این منظر Booch از وضعیت خوبی برخورددار است، چرا که هم فازی برای نگهداری داشته و هم رویکرد افزایشی-تکرار شونده ی خود، به خوبی می تواند تعداد افراد زیاد در پروژه در هر لحظه را مدیریت کند. در مقام مقایسه می توان گفت که Fusion ضعیف تر عمل کرده و تنها از formalism بهره برده که مزیتی برای آن است تا برای سطوح بحرانیت بالا، مقیاس پذیر باشد.

مقایسه با OMT: متدولوژی OMT توانایی اجرا بر روی پروژه های با اندازه بزرگ را به نسبت دارد، اما پروژه ای که سطح بالایی از بحرانیت را دارد، برای این متدولوژی مناسب نیست؛ چرا که نه به روش های صوری در آن تصریح شده

است و نه نگهداری در آن تاکید شده است. از آنجا که تعدد مدل در آن وجود ندارد و پیچیدگی فرآیند در آن کم می باشد، پروژه های با اندازه بزرگ در حوزه OMT می توانند قابل اجرا باشند. در این مقایسه، می توان گفت که Fusion چون نه نگهداری دارد و نه آزمون، مقیاس پذیری کمتری نسبت به OMT دارد ولی به دلیل استفاده از روش های صوری در operation ها، از خود برای مدیریت پروژه های بحرانی، توانایی نشان می دهد.

- **قابلیت پیکربندی**

مقایسه با RDD: در RDD قابلیت پیکربندی یافت نمی شود. بدین منظور، فرآیند آن یکپارچه بوده و نمی توان وضعیت پروژه ای خاص، فرآیند متدولوژی را پیکربندی کرد. متدولوژی Fusion دقیقا به همین زاویه با RDD شباهت دارد.

مقایسه با Booch: متدولوژی Booch نیز مانند Fusion توانایی خاصی برای پیکربندی از خودش نشان نمی دهد. مقایسه با OMT: همچنین فرآیند OMT را نمی توان پیکربندی کرد که از این لحاظ شبیه به Fusion می باشد.

- **انعطاف پذیری**

مقایسه با RDD: اینکه بتوان فرآیند را در حین اجرای متدولوژی مورد بازنگری قرار داد در RDD تصریحی ندارد. هیچ گونه جلسات مرور فرآیند دوره ای و یا بازخوردی از افراد ذکر نشده است. در Fusion نیز چنین است.

مقایسه با Booch: متدولوژی Booch می تواند با رویکرد افزایشی-تکراری خود، تمرکز روی فرآیند ها را با بازبینی در جلسات، تغییر بدهد و امکان بازگشت به عقب می تواند مهیا باشد. بدین ترتیب بسیار بهتر از Fusion عمل می کند.

مقایسه با OMT: در OMT نیز قابلیت انعطاف پذیری مهیا نیست و از این نظر به Fusion شباهت دارد.

۱۱.۱.۲ حوزه کاربرد

مقایسه با RDD: متدولوژی RDD از نظر این که کاوش و تحلیل را به خوبی با کشف مسئولیت همراه کرده است می توان از آن برای سیستم های اطلاعاتی data-intensive استفاده کرده است. متدولوژی Fusion نیز می تواند با تفکر بی درز شیءگرای خود و تحلیل و طراحی شیءگرایی که مرز سیستم را نیز به خوبی مدل می کند در تامین حداقل حوزه کاربرد، موفق عمل کند. از طرفی بهره مندی از روش صوری به کمک متدلوژی Fusion می آید و حوزه کاربرد آن را برای پروژه های نیازمند دقت، وسیع تر می کند.

مقایسه با Booch: از طرفی Booch روش صوری را صریحا ندارد و Fusion از روش صوری پشتیبانی می کند؛ از طرفی دیگر Booch فاز نگهداری را معرفی می کند و Fusion کاری برای نگهداری نمی کند. به نظر می رسید هر دو در تامین حداقل سیستم اطلاعاتی حساس به داده مناسب باشند ولی پروژه نیازمند نگهداری را Booch بهتر مدیریت می کند و پروژه های نیازمند دقت و صحت بالا را Fusion بهتر می تواند جواب دهد.

مقایسه با OMT: در نظر داریم که از این نظر، حوزه کاربرد OMT می تواند وسیع بوده و حداقل ها را تامین کند(کما اینکه NASA از آن بهره برده است). می توان از متدولوژی این نتیجه را گرفت که حداقل برای سیستم های اطلاعاتی مفید بوده، چرا که حدی از مدلسازی را پشتیبانی می کند. از این نظر به Fusion شباهت دارد(فقط OMT به دلیل نداشتن فاز نگهداری و عدم استفاده از روش صوری، برای حوزه کاربرد هایی که با جان انسان سر و کار دارد و یا باعث هدررفت منابع زیادی می شود، لطمه زننده خواهد بود در حالی که Fusion نیز نگهداری ندارد ولی پیش شرط ها و پسا شرط ها به کمک آن می آیند).