هفتمین همایش سالانه بانکداری الکترونیک و نظام‌های پرداخت
نوآوری، بازیگران جدید و کارآیی در کسب وکار مالی
تهران، مرکز همایش‌های بین‌المللی برج میلاد - ۲و۳ بهمن ۱۳۹۶
7th Annual Conference on Electronic Banking and Payment Systems

انبارداده در بانک‌ها بر مبنای متدولوژی چابک

Data warehousing in banks based on agile methodology

پرویز وکیلی صادقی، تحلیلگر گروه هوش تجاری و داده‌کاوی شرکت داده‌ورزی سداد، p.vakili@sadad.co.ir
ایمان اسلامیان، معاون مدیرعامل موسسه همیاری غدیر بانک صادرات ایران، i.eslamian@bsi.ir

چکیده (فارسی)

در نظام بانکداری ایران، مدیران سطوح مختلف بانکی برای تصمیم‌گیری به اطلاعات مرتبط به عملیات فعلی بانک، روندها و تغییرها نیاز دارند. در چند سال اخیر پیشنهاد متخصصین فنآوری اطلاعات، همیشه راهکارهای مبتنی بر هوش تجاری بوده است. در سطح شاخص و حوزه‌های داده‌ای مراجع معتبری مانند BIAN در بیشتر موارد گره‌گشا بوده، حال آنکه طراحی انبارداده فارغ از تکنولوژی مورد استفاده در سطح متدولوژی طراحی و پیاده‌سازی در مواجهه با تغییرات دچار چالش اساسی می‌باشد. برای پاسخ به این نیاز متدولوژی چابک در طراحی انبارداده بررسی و رویه اجرایی آن ارایه می‌شود (متدولوژی چابک در انبار داده با چابک در نرم افزار متفاوت است). این متدولوژی منشعب شده از اسکرامبان (Scrumban) است. در این مقاله روش اجرایی چابک و فرآیندهای مناسب برای انبارهای داده در صنعت بانکداری ارایه و تشریح می‌شود. مجموعه مطالب ارایه شده در این مقاله، برای تیم های اجرایی این امکان را فراهم می‌آورد که در مقابل تغییرات عکس‌العملهای مناسب و اجرایی داشته‌باشند.

واژگان کلیدی :انبارداده، طراحی چابک انبارداده، معماری انبار داده

چکیده (انگلیسی)

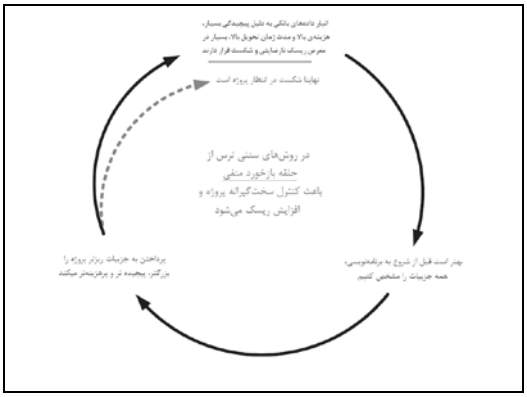
In the banking system of Iran, managers require decisions and information related to the current operations of the bank, trends and changes. Lately, the suggestion of IT professionals has always been based on BI. At the level of subject domains of credential references such as BIAN is accepted, while Data warehouse design is in difficulty when faced with changes. To meet this need, an agile methodology is presented in the design of the warehouse and its implementation is presented. This methodology is Inherited from Scrumban.

In this paper, in addition to introducing an agile executive approach, an appropriate architecture for data warehousing in the banking industry is presented and explained. The subjects in this article allows teams to respond to changes in appropriate and effective responses.

Keywords: Data Warehouse, Agile Data Warehousing, Data Warehouse Architect

مقدمه

در تئوری، انبارداده‌ی بانک به‌عنوان سرویس‌دهنده اطلاعاتی برای بانک می‌تواند بسیار ارزشمند باشد، اما در عمل مشاهده می‌شود این انبارداده در زمان مورد نظر مدیران و با هزینه معقول، غالبا اجرا نمی‌شود و در مواجهه با تغییرات به حد کافی سریع عمل نمی‌کند. یک انبار داده‌ی بانکی مخزن مشترک از اطلاعات استاندارد شده و قابل اطمینان در مورد رویدادها، فرآیندها و تراکنش‌های بانک در سراسر شعب بانک و همچنین ستاد است. مورد ناخوشآیندی که در این حوزه مشاهده می‌شود آن است که برای برآورده شدن برخی نیازهای اطلاعاتی باید زمان طولانی صبر و ده‌ها میلیون ریال سرمایه‌گذاری کرد و با روش‌های مهندسی نرم افزار سنتی معمولا نتیجه حاصل‌شده یک زیر مجموعه کوچکِ ناامیدکننده از قابلیت‌هایی که انتظار می‌رود، خواهد بود ضمن آن‌که استرس وارد شده به تیم توسعه دهندگان قابل توجه است. در بررسی های صورت گرفته نکته قابل توجهی به‌عنوان دلیل این چالش‌ها نمایان شد. با توجه به ماهیت پروژه‌های انبارداده (پیچیدگی ذاتی، هزینه بالا، طولانی بودن زمان تحویل و کمبود مدیران پروژه باتجربه در این حوزه) ترس مخاطره‌آمیزی از شکست، ذینفعان پروژه را احاطه می‌کند که باعث اتخاذ روش‌های مهندسی و مدیریت پروژه بسیار صُلب و خطرناک می‌شود تا نهایتا لیست کارهای توسعه دهندگان پروژه را به طور قابل توجهی طولانی‌تر و پروژه را پیچیده‌تر می‌کند. حال وقتی بازخوردهای منفی پس از تحویل، ارایه می‌شود این ترس شدیدتر و فرآیند توسعه با کنترل و حسابرسی بسیار پیچیده‌تر مواجه می‌شود. در تصویر ۱ می‌توان حلقه بازخورد منفی[1] را مشاهده کرد.



تصویر ۱ – حلقه بازخورد منفی

برای غلبه بر این چالشِ اساسی بسیاری از تیم‌های توسعه دهنده سراغ استفاده از رویکردهای تحویل تدریجی[2] که در پروژه‌های نرم افزاری استفاده می‌شدند، در پروژه‌های انبارداده رفتند؛ ولی فرآیندها در پروژه‌های انبار داده ذاتا با پروژه‌های نرم‌افزاری متفاوت است که باعث مشکلات در اجرا شد. به‌عنوان مهمترین این مشکلات می‌توان به دو موضوع اشاره کرد؛ اول این‌که در معماری انبارهای داده تعداد لایه‌های موجود حداقل دو تا چهار برابر سیستم‌های نرم‌افزاری است و هر لایه نیازمند استراتژی مدل داده ویژه خود، روش‌های تبدیل داده متفاوت و حتی مجموعه ابزار توسعه متفاوت است که این مانند آن است

[1] Negative feedback loop
[2] Incremental delivery approach

که در یک پروژه انبارداده در آنِ واحد، چندین پروژه نرم افزاری را بخواهیم در قالب یک پروژه مرتبط پیش ببریم. دوم اینکه در یک پروژه انبارداده در سطح بانک اگر نخواهیم بگوییم میلیاردها، حداقل با میلیون‌ها رکورد سر و کار داریم که می‌بایست در انبارداده تزریق شوند که اگر بخواهیم فرآیندهای تزریق را در نظر بگیریم ممکن است روزها زمان از پروژه طلب کند. بنابراین در الگوهای تکرارپذیر در شرایط تغییرات بار زیادی به زمان وارد می شود که عملا پروژه را با خطر مواجه می‌کند و در حالت بدبینانه اگر منبع این اطلاعات دیگر در دسترس نباشد، تیم باید پس از گذراندن صدها ساعت نوشتن، اجرا و اعتبار اسکریپتهای تبدیل که میلیون‌ها سوابق داده را به منظور انطباق با طراحی جدید انبار تأمین می کنند، گام بعد را آغاز کند.

هدف این مقاله نشان دادن استراتژی‌ها و تکنیک‌های جایگزین است تا تیم‌های انبارداده برای ساخت برنامه‌های بزرگ و کاربردی داده محور استفاده کنند. روش ارایه شده از چهار روش مختلف تکرارپذیر Scrum، XP، Kanban و RUP و همچنین تعدادی از روش‌های مهندسی نظیر مهندسی نیازمندی‌ها و تضمین کیفیت استفاده می‌کند. این روش بر چهار تاثیر میگذارد که عبارتند از:

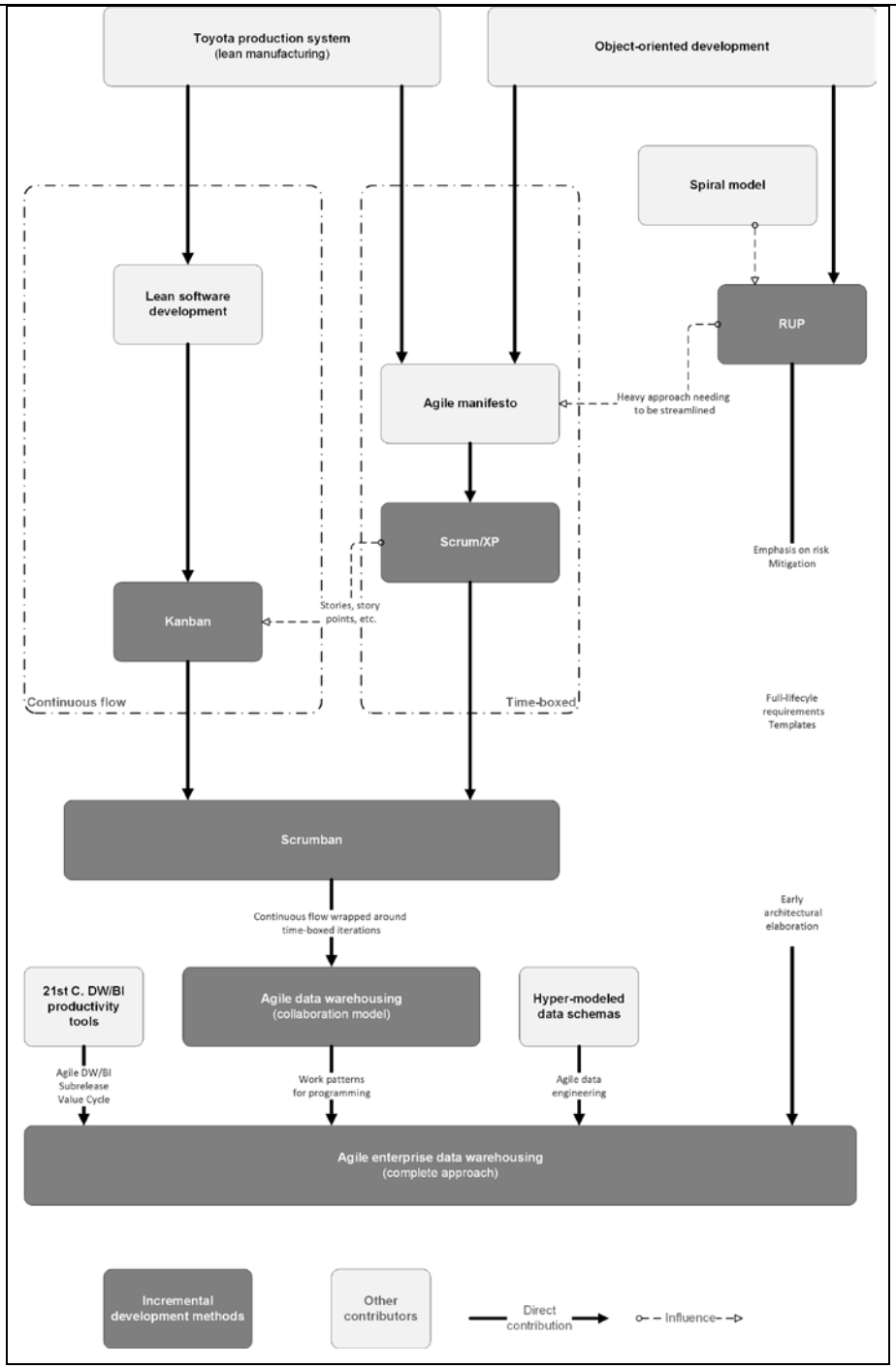
۱. تکنیک‌های کدنویسی افزایشی و تکرارپذیر[۳]، که نه تنها سرعت تحویل سریعتری را فراهم می کنند، بلکه باعث کاهش قابل توجه ریسک می شود.
۲. مدیریت نیازمندی‌های ساده و موثر[۴] که کار تعریف پروژه را سریع و متمرکز می کند.
۳. مهارت‌های مهندسی داده[۵] که اجازه می دهد انبارداده به صورت گام به گام ایجاد شود و سپس حتی پس از تزریق داده ها تغییرات لازم را به صورت به صرفه اصلاح کنیم.
۴. تضمین کیفیت متعادل[۶] که سعی می‌کند تست تدریجی را در تمامی سطوح پروژه انجام می دهند.

در ادامه مقاله ضمن تشریح ادبیات موضوع، در بخش یافته‌ها و نتایج پنج پایه تشریح می‌شود.

ادبیات موضوع

رویکرد مهندس نرم‌افزار برای ایجاد انبار داده بانکی به صورت چابک از تکنیک‌های سنتی و جدید بسیاری استفاده می کند که در تصویر ۲ قابل مشاهده است. این رویکرد در هسته خود، تکنیک‌های چابک را برای برنامه نویسی تکرارشونده و افزایشی استفاده می‌کند. در این بخش جهت آشنایی با تکنیک‌هایی که از آنها اقتباس صورت گرفته، مرور کلی از خواهیم نمود. در ابتدا شرحی بر دو روش اسکرام[۷] و XP[۸] از بیانیه چابک می‌پردازیم. سپس یک نگاه گذرا به تولید نرم افزار بهینه[۹] و کانبان[۱۰] خواهیم داشت و در خاتمه به مرور RUP[۱۱] می‌پردازیم.

[3] Iterative, incremental application coding
[4] Streamlined requirements management
[5] Adaptive data engineering
[6] Balanced quality assurance
[7] Scrum
[8] Extreme Programming
[9] lean software development
[10] Kanban
[11] Rational Unified Process



تصویر ۲- سلسله مراتب اقتباسی طراحی انبارداده چابک

مروری بر چابکی

با توجه به روش تولید نرم‌افزار در اواسط قرن بیستم، مدیران پروژه اصل را براین می‌گذاشتند که کار برای هر مرحله باید به طور کامل به پایان برسد تا قابل ارایه باشد، این رویکرد سنتی به وضوح در سال ۱۹۷۰ در مقاله ای از رویس[12] با عنوان "مدیریت توسعه سیستم های نرم افزاری بزرگ" به طور واضح بیان شد که اغلب به عنوان روش آبشاری[13] نامیده می شود. با توجه به نقاط ضعفی که این روش داشت، به مرور زمان بعضی از توسعه دهندگان تحویل‌های افزایشی و تکراری سیستم های نرم افزاری را آزمایش کردند. با ترکیب یک رویکرد تکراری با تکنیک‌های جدید در آن دوران، مهندسان نرم افزار در طیف گسترده ای از صنایع، شناسایی راه‌های جدیدی را برای ساخت برنامه های کاربردی آغاز کردند. در سال ۲۰۰۱ تعدادی از پیشروهای این رویکرد، بیانیه‌ای برای چابکی ارایه کردند که عبارت بودند از:

- تکیه بر دانش افراد و توانایی و تعاملات آنها به جای تکیه بر فرآیندها و ابزار
- هدف اصلی تولید نرم افزار قابل اجرا است و مستندات سنگین هدف اصلی نیست
- تعامل با مشتریان و مشارکت آنان برتر از چانه زنی در قرارداد است
- پاسخ به تغییرات برتر از دنبال کردن یک طرح ثابت است

اگر موارد مطرح شده در سمت راست تامین شوند، موارد سمت چپ نیز تامین خواهند شد. البته باید توجه کرد که به طور کلی هر دو رویکرد سنتی و چابک در مراحل عمده و توالی فعالیت‌هایی که منتج به نظم مهندسی نرم افزار می‌شود با گام‌های الزامات سیستم، الزامات برنامه، تجزیه و تحلیل، طراحی، برنامه نویسی، تست، عملیات، و نگهداری موافق هستند. امروزه بسیاری از نویسندگان چابکی را به‌عنوان هر روشی که به بیانیه چابک وفادار بماند، تعریف می‌کنند. در نگاه نخست این تعریف خوب است، اما وقتی به دقت به آن نگاه می‌شود این تعریف توضیح نمی دهد چه چیزی از چابکی متفاوت از روش‌های سنتی است. برای رفع این چالش، نویسندگان بیانیه ۱۲ اصل پیشنهاد شده را ارائه می دهند که توسعه دهندگان باید در طی یک پروژه دنبال کنند. این اصول عبارتند از:

۱. بالاترین اولویت، جلب رضایت مشتری با ارائه به موقع و متداوم محصول با ارزش است
۲. استقبال از تغییر نیازمندیها، حتی در مراحل آخر توسعه (فرآیندهای چابک تغییرات را به نفع مشتری مدیریت می‌کنند)
۳. تحویل زود به زود نرم افزار کاربردی هر چند هفته تا چند ماه یک بار و با اولویت ارائه در زمان کوتاهتر
۴. همکاری بخش تجاری با تیم توسعه به طور روزانه در تمام طول پروژه
۵. تکیه بر توانایی های افراد در ساخت پروژه(ارائه محیط و پشتیبانی مناسب و اعتماد به افراد برای انجام کار)
۶. بهترین وموثرترین روش تبادل اطلاعات در یک تیم توسعه تعامل چهره به چهره است
۷. معیار اصلی پیشرفت، نرم افزار کاربری ارائه شده است
۸. فرایند چابک توسعه پایدار را تضمین می کند. پشتیبانان، توسعه‌دهندگان و کاربران باید قادر به مشاهده دائمی پیشرفت باشند
۹. توجه دائمی به طراحی خوب و تکنیک های عالی، چابک سازی را افزایش می دهد
۱۰. سادگی نیاز اصلی است
۱۱. بهترین معماری ها، نیازمندی ها و طرح ها توسط تیم های خود سازمانده بوجود می آیند

[12] Winston Royce
[13] Waterfall method

۱۲. در فاصله های منظم، تیم سعی می کند که موثرتر باشد و سعی می‌کند در همین راستا رفتار خودش را تغییر داده یا تطبیق دهد

طبق نظرسنجی‌های صورت گرفته از تیم‌های توسعه‌ی برنامه‌های کاربردی، سه چهارم تیم‌های مرتبط با انبارداده در فرآیند توسعه، سعی می‌کنند به بیانیه چابکی وفادار باشند و این بیانیه را در تکنیک های تکراری و تکاملی به کار گیرند [Hughes & Stodder 2012]

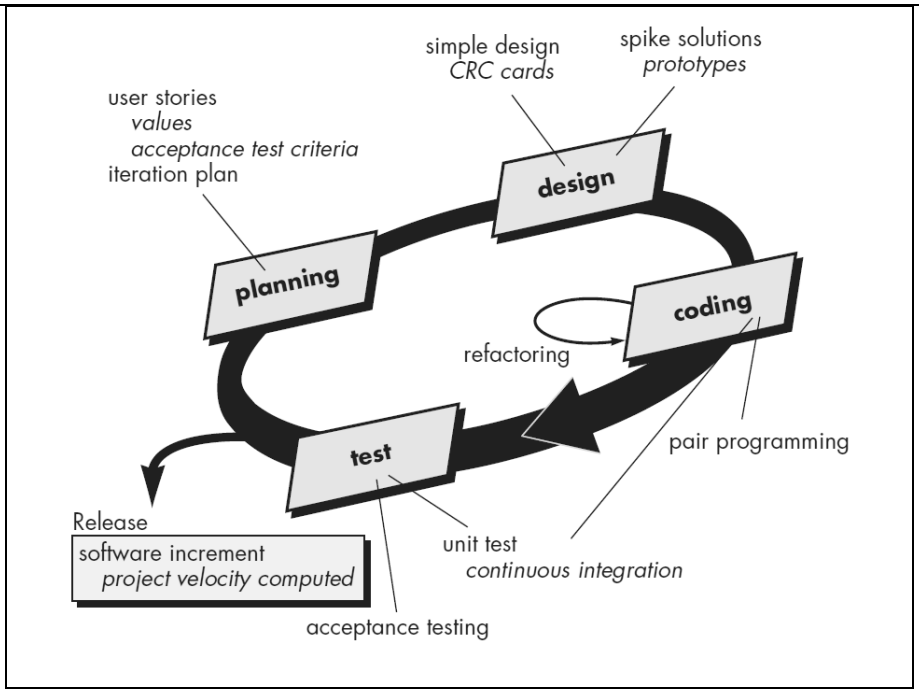
از آنجاییکه در فرآیند انبارداده چابک که از روش تکرارپذیر استفاده می‌کنند، ایجاد تقاضاهای تغییر بیش از حد، پروژه را مستعدِ انحراف از استاندارهای چابک می‌کند، ارزش ها و اصول برای رهبران پروژه‌های انبارداده چابک بسیار مهم است تا تیم های خود را در پیاده سازی به طور واضع در مسیر روش‌های چابک باقی نگهدارند؛ اما تجربه نشان می‌دهد، پیروی مطلق از اصلی و ارزش‌های چابک که در کتاب‌ها یافت می‌شود، پروژهای انبارداده را با نقصان مواجه می‌کند. برای مثال در کتاب‌های اسکرام اشاره می‌شود در پایان هر تکرار قابلیت‌های جدید به کاربران ارایه شود. کسانی که سابقه کار در پروژه‌های انبارداده را دارند به وضوح می‌دانند این توصیه عملا در پروژه‌هایی که اهداف کلان یکپارچگی دارند غیرممکن است. فرض کنید بخواهید یک قلم اطلاعاتی به داشبورد گزارشی اضافه کنید، برای این هدف به سادگی با حجم زیادی از کار ETL[14] مواجه خواهید بود، بررسی تاثیرات جانبی و طراحی در داشبورد هم جزو فرآیندهای زمان بر خواهد بود. بنابراین تیم های چابک انبار داده علی‌رغم فعالیت به صورت چرخه های تکرارپذیر، از ارایه قابلیت جدید در پایان هر چرخه چشم‌پوشی می‌کنند و این چشم پوشی باعث می‌شود مشتری حسی از پیشرفت پروژه نداشته باشند و نتواند نظرات خود را ارایه نماید. تیم‌های موفق انبارداده چابک، اصل ۱ بیانیه را در ذهن نگه می دارند و برای رسیدن به آن از روش‌های ابتکاری استفاده می‌کنند.

برنامه‌نویسی مفرط (XP)

کنت بنت[15] در حین انجام پروژه ای روی سیستم برآورد هزینه های شرکت کرایسلر در سال ۱۹۹۶ روش مفرط را ارائه کرد و در کتاب خود "Extreme Programming explained" آن را اینگونه تعریف کرد: "متدی سبک برای تیم های کوچک تا متوسط که با در نظر گرفتن نیازهای مدام در حال تغییر، نرم افزار را تولید می کنند."

IEEE تفاوت روش مفرط با روش‌های سنگین را اینگونه توضیح می دهد:"متد مفرط روشهای مرسوم توسعه نرم افزار را کنار میزند؛ یعنی به جای تعریف، آنالیز و توسعه برای تولید در آینده دور، برنامه نویسان روش مفرط تمام این مراحل را به مرور در حین توسعه انجام می دهند". در نتیجه کار تدریجی، در روش مفرط هزینه اعمال تغییر در نرم افزار عموما ثابت است. برنامه نویسی افراطی یا مفرط پراستفاده ترین روش فرآیندهای چابک است. برنامه ریزی برنامه نویسی افراطی با ایجاد داستان های کاربری شروع می شود، تیم چابک این داستان ها را ارزیابی کرده و هزینه آنها را مشخص می کند دسته بندی داستان ها برای یک انتشار مرحله ای و یک تعهد بر تاریخ تحویل صورت می گیرد پس از ارائه اولین مرحله از محصول، سرعت پروژه برای تعریف زمان ارائه ویرایش های بعدی محصول استفاده می شود. تصویر ۳ این فرآیند را نمایش می‌هد.

[14] Extract, Transform, and Load
[15] Kent Beck

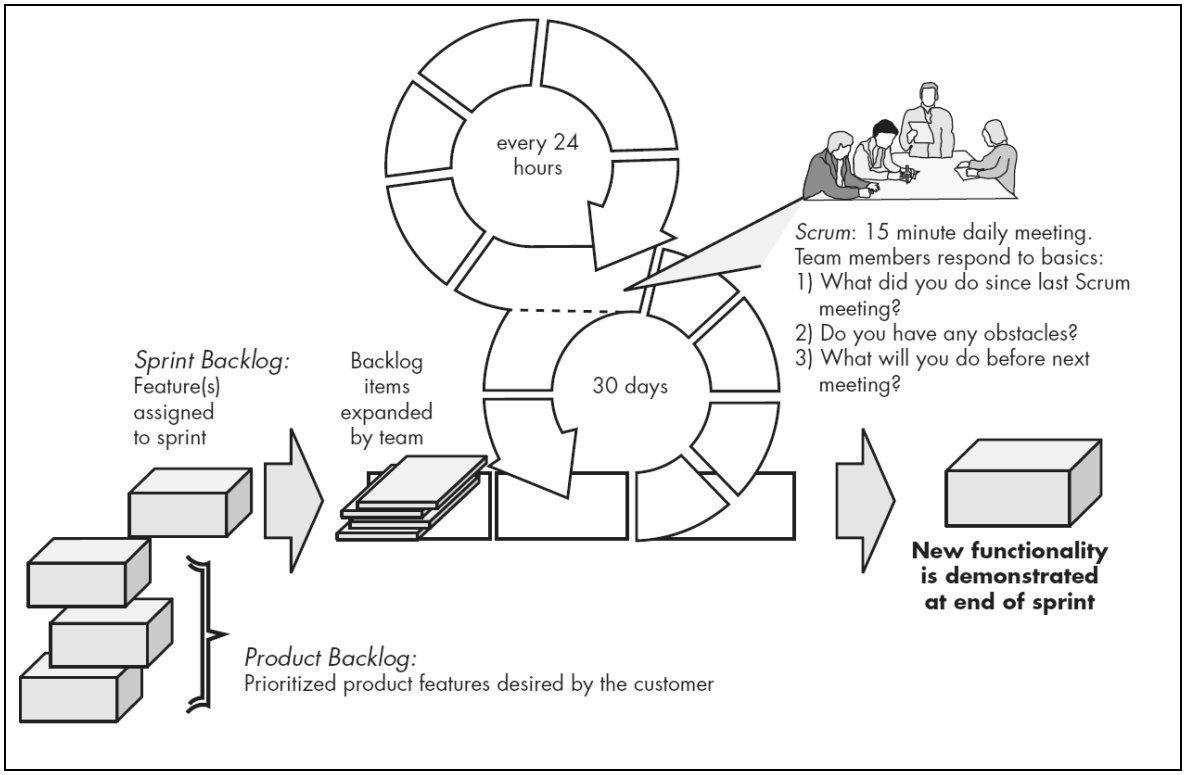


تصویر ۳ – چرخه برنامه‌نویسی مفرط

اسکرام

متدولوژی اسکرام در سال ۱۹۸۶ در کشور ژاپن توسط تاکیچی[۱۶] و نوناکا[۱۷] برای اولین بار مطرح شد. اسکرام در دهه ۹۰ میلادی توسط شوبر[۱۸] و استرلند[۱۹] توسعه داده شد و به عنوان یک متدولوژی رسمی جهت تولید محصولات نرم افزاری شناخته و به کار گرفته شد. در یک پروژه بر مبنای اسکرام، به تعدادی پروژه کوچک به نام بسته[۲۰] تقسیم می شود، تست و مستند سازی در طول ساخت انجام می شود، کار به صورت دوری[۲۱] انجام شده و در هر دور از روی لیست کارهای[۲۲] لازم انجام می شود. جلسات با فواصل کوتاه برگزار شده و بعضی از مواقع به طور ایستاده برگزار می شود. ارائه دمو (نمونه برنامه) به مشتری در زمانبندی های مشخص شده انجام می‌شود. چرخه فعالیت اسکرام را در تصویر ۴ می‌توان مشاهده کرد.

[16] Hirotaka Takeuchi
[17] Ikujiro Nonaka
[18] Ken Schwober
[19] Jeff Stherland
[20] Packet
[21] Sprint
[22] backlog)



تصویر ۴ – چرخه اسکرام

تولید نرم افزار بهینه[۲۳]

در حالی که اسکرام با نفوذ تولید ژاپنی و نوآوری‌ها در برنامه نویسی شی گرا ناشی شد، ، تولید نرم افزار بهینه از فرآیند معکوسی مشتق شده است که حاصل مطالعه دقیق تولید به سبک ژاپنی به ویژه سیستم تولید تویوتا و تطبیق آن با مهندسی نرم افزار شکل گرفت.

تفسیر و تبدیل سیستم تولید تویوتا به اصول توسعه نرم افزار بهینه در اوائل سال ۲۰۰۰ توسط پاپندیک‌ها [۲۴] آغاز شد. [Poppendieck 2004] نوآوری پایدار در نگرش‌های مدیریتی و تکنیک‌های ساخت و ساز در اواسط دهه ۱۹۷۰ در سیستم تولید تویوتا ارایه شد، که توسط چندین مهندس تویوتا از جمله تایچی اوون[۲۵] تعریف شده بود. رویکردی که اوون و همکارانش اتخاذ کردند، به عنوان پایه ای برای "تولید بهینه[۲۶]" به کار گرفته شد، که توسط جیمز واماک[۲۷] و دانیل جونز [۲۸]در مجموعه

[23] lean software development
[24] Mary and Tom Poppendieck
[25] Taiichi Ohno
[26] lean manufacturing
[27] James Womack
[28] Daniel Jones

کتاب های در ایالت متحده، از جمله Thinking Lean [Womack & Jones 2003] گسترش یافت. این کتابها به طور گسترده ای توسط طرفداران توسعه نرم افزار بهینه از جمله پابندیک‌ها و دیوید اندرسون[۲۹]، خالق روش کانبان[۳۰] اشاره شده است. اصول ارایه شده توسط پابدیک برای روش بهینه عبارتند از:

- کاهش ضایعات
- تقویت یادگیری
- تصمیم‌گیری تا بیشترین حد امکان
- ارایه در سریعترین زمان ممکن
- توانمند سازی تیم
- یکپارچگی ذاتی و درونی
- نگاه کل گرایانه

ابزار اجرای به این اصول را در جدول ۱ می‌توان مشاهده کرد.

Principles / tools	
1 Eliminate waste	5 Empower the team
Tool 1: Seeing waste	Tool 13: Self-determination
Tool 2: Value stream mapping	Tool 14: Motivation
2 Amplify learning	Tool 15: Leadership
Tool 3: Feedback	Tool 16: Expertise
Tool 4: Iterations	6 Build integrity in
Tool 5: Synchronization	Tool 17: Perceived integrity
Tool 6: Set-based development	Tool 18: Conceptual integrity
3 Decide as late as possible	Tool 19: Refactoring
Tool 7: Options thinking	Tool 20: Testing
Tool 8: The last responsible moment	7 See the whole
Tool 9: Making decisions	Tool 21: Measurements
4 Deliver as fast as possible	Tool 22: Contracts
Tool 10: Pull systems	
Tool 11: Queuing theory	
Tool 12: Cost of delay	

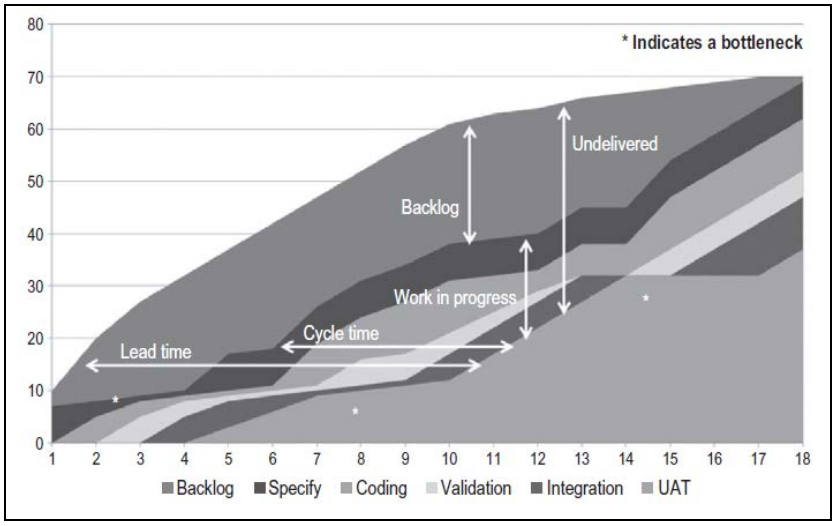
جدول ۱ – اصول و ابزار توسعه نرم افزار بهینه

کانبان

اصول مطرح شده در تولید نرم افزار بهینه، تنها مجموعه ای از مفاهیم و ترجیحات برای تیمهای نرم افزاری را به همراه دارد. می‌توان به نوعی گفت موارد اشاره شده بسیار سطح بالا هستند و هنگامی که اولین بار، تحویل در انتهای چرخه شروع می‌شود، اکثر تیم ها متوجه می‌شوند به مراتب بیش از راهنمایی‌های سطح بالای مطرح شده نیاز دارند تا بتوانند در یک فرآیند دقیق برای ارائه سریع نرم افزار جدید اقدام کنند. خوشبختانه، دیوید اندرسون در اواسط سال ۲۰۰۰، در حین مدیریت

[29] David Anderson
[30] Kanban

پروژه‌ها در مایکروسافت، یک روش خاص را بر پایه مفاهیم تولید نرم‌افزار بهینه پیش گرفت. نتیجه‌ی این تجربه، کانبان نامیده شد، کانبان بعد از یک دهه یا بیشتر پس از اسکرام ظاهر شد و علی‌رغم اینکه جمعیت رو به رشدی از پیروی‌کنندگان را دارد، تیم های چابک آنچنان که شایسته آن است توجه زیادی به آن نکرده‌اند. طرفداران کانبان، تمایز بسیاری میان آن و اسکرام را مشخص می کنند، اما در واقع این دو به تا این حد از هم تمایز ندارند. در تیم‌های موفق انبارداده مشاهده شده راه‌هایی برای ترکیب این دو روش یافته‌اند، به همین دلیل، آشنایی با کانبان برای تیم هایی که مایل به ساخت انبار داده‌ها با استفاده از تکنیک‌های تحویل افزایشی هستند ضروری است. در تصویر ۵ نمودار انباشت سبک کانبان نشان داده‌شده که در آن فرآیندها مشخص است.



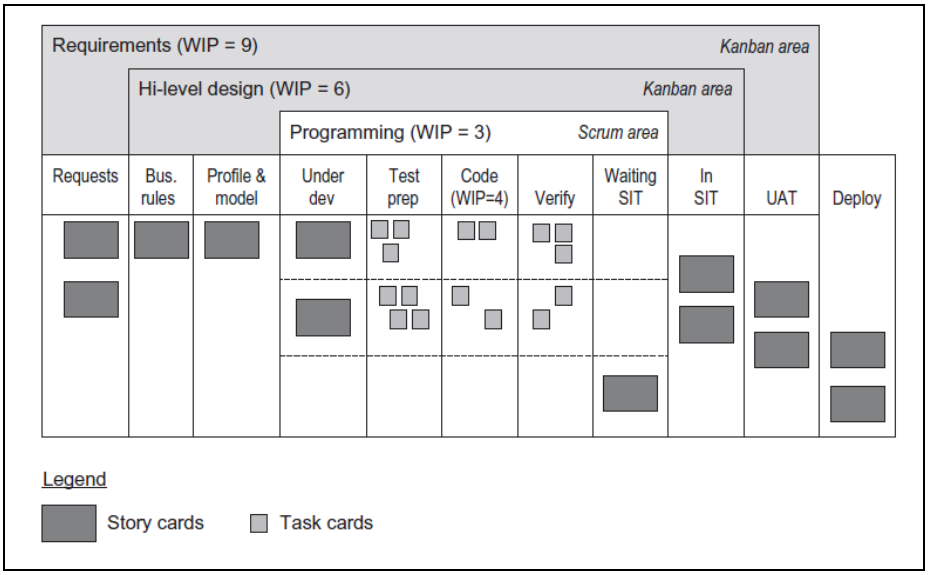
تصویر ۵ – نمودار انباشت شیوه کانبان

رویکرد تلفیقی اسکرامبان[31]

به طرز جالب توجهی، زمانی که بحث میان پیروان اسکرام و کانبان به بالاترین درجه حرارت خود رسید، یک روش ترکیبی به آرامی معرفی شد. اسکرامبان توسط کوری لادس[32] در سال ۲۰۰۹، به عنوان راه‌حل میانجی برای تیم های چابک ارایه شد [Ladas 2009] . در واقع، بسیاری از تمرینکنندگان اسکرام را پیدا می کنند تا ساده تر توضیح دهند، آموزش، نظارت و تنظیم در سازمان هایی که کاملا جدید به تفکر چابک هستند، در نتیجه آن را نقطه شروع بهتر است. با این حال، آنها می خواهند مراسم گران قیمت و زمان بندی را در اسرع وقت هنگامی که تیم هایشان بالغ می شوند، رها کنند. آنها به سادگی به مجموعه ای از ایستگاههای راه دور نیاز دارند تا در سفر خود از Scrum به سمت توسعه نرم افزاری ناب استفاده شوند.

لادس اسکرامبان را به جای آنکه به عنوان روش معرفی کند آنرا به عنوان یک فرآیند گذار توصیف نمود. در این فرآیند، تیم‌ها کار خود را با اسکرام شروع کرده و در مراحل بعدی به سوی کانبان حرکت می کنند. تصویر ۶ نمونه‌ای برد اسکرامبان را نشان می‌دهد.

[31] SCRUMBAN
[32] Corey Ladas



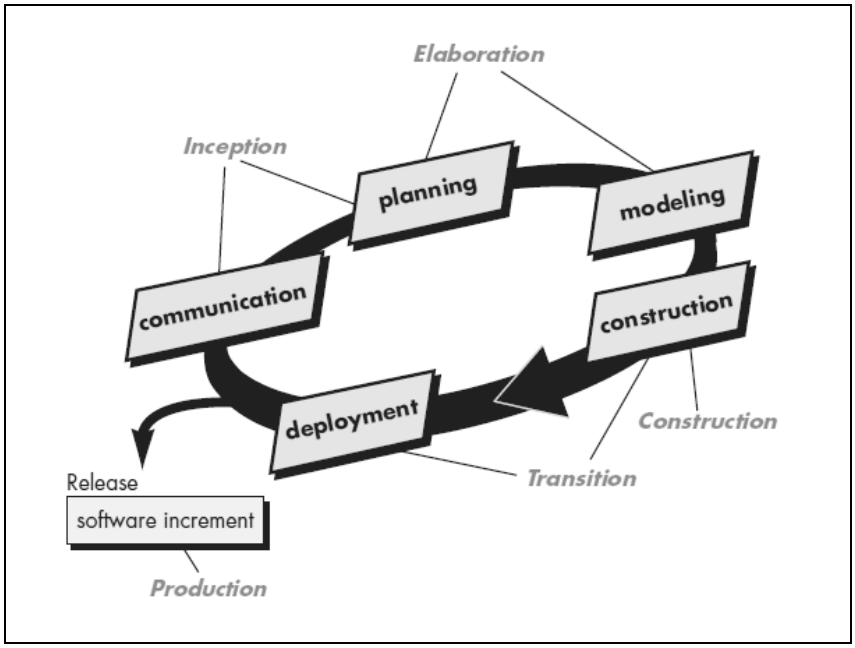
تصویر ۶ – نمونه‌از برد اسکرامبان

RUP[۳۳]

حدود نیم دهه یا بیشتر قبل از اعلامیه چابک، فرایند یکپارچه منطقی به عنوان یکی از اولین روشهای تکراری حضور پیدا کرد و به طور گسترده ای تبلیغ می شد. RUP از ادغام تکنیک های نوین مهندسی و تکنیک های مدل سازی سیستم توسط چندین طراح و برنامه نویسی شی گرا برجسته به نامهای بوچ[۳۴]، جاکوبسن[۳۵] و رومبوا[۳۶] [Jacobson & Booch 1999] آغاز شد.

فرآیند یکپارچه یک روش نظام‌مند برای تخصیص کارها و مسئولیت ها است، استفاده از روش تکرار و توسعه تدریجی، معماری مبتنی بر مولفه ها، راهبری بر مبنای موارد کاربری، مدلسازی بصری نرم افزار، کنترل تغییرات و بررسی کیفیت نرم افزار از اصول این روش است. در این فرآیند همه اعضای تیم به پایگاه دانش پروژه دسترسی دارند. هدف از این روش تولید نرم افزار به صورت بهینه و با کیفیت بالا و در زمانبندی مشخص شده و با بودجه مناسب است. در این مدل یک چرخه حیات نرم افزار شامل فازهای آغازین[۳۷]، تشریح[۳۸]، ساختن[۳۹] و انتقال[۴۰] که در نهایت به تولید[۴۱] و انتقال به محیط واقعی منجر می شود. تصویر ۷ چرخه مربوط به آن‌را نمایش می‌دهد.

[33] Rational Unified Process
[34] Booch
[35] Jacobson
[36] Rumbaugh
[37] Inception
[38] Elaboration
[39] Construction
[40] Transition
[41] Production



تصویر ۷ – چرخه RUP

روش تحقیق

تحقیق حاضر از منظر هدف یک تحقیق توسعه‌ای کاربردی است، چون منجر به توسعه دیدگاه در خصوص روش توسعه نرم افزار چابک می‌شود، با استفاده از دستآوردهای آن می توان یک رویکرد جدید در طراحی انبارداده بانک‌ها در پیش گرفت. همچنین رویکرد تحقیق کیفی بوده و در زمره مطالعات اکتشافی با استراتژی پیمایش قرار می‌گیرد.

یافته‌ها و نتایج

در مقدمه مقاله به چالش‌های موجود در طراحی انبارهای داده بانکی اشاره شد و هدف از ارایه این مقاله پیشنهادی اقتباس شده از روش های تکراپذیر و افزایشی برای حل مشکلات مطرح شده است. همانطور که بیان شد مشکل پایه‌ای، صرف زمان طولانی و هزینه‌ی بالا برای ساخت انبارداده در مقابل برآورده شدن حداقل نیازهای مورد انتظار است. در روش‌های سنتی حتی یک اشتباه ساده در شناخت نیازمندی‌ها یا طراحی به راحتی می‌تواند ماه‌ها انرژی صرف شده برای تبدیل داده‌ها و فعالیت فنی صورت گرفته را بی‌اثر کند. برای رفع این مشکل پایه‌ای به کار بردن روش‌های چابک در چهار حوزه زیر پیشنهاد می‌شود:

- تکنیک‌های برنامه‌ریزی و فنی
- مهندسی نیازمندی‌ها
- مهندسی داده
- تضمین کیفیت

تکنیک‌های برنامه‌ریزی و فنی

اگرچه روشهای عمومی چابک بدون توجه به بخش‌های زیادی از چرخه کامل نرم افزار، آنرا پیش می‌برند، اما هنوز هم می‌توانند به طور چشمگیری یک پروژه انبارداده را به تنهایی در جهت بهبود تغییر دهند. اسکرام در تکرار اول خیلی خوب عمل می‌کند و پنج گام آن به خوبی تشریفات، ترمینولوژی و اهداف مورد نیاز را پشتیبانی و هدایت می‌کند. با این‌حال چالش‌های مطرح شده در یکپارچگی داده هنوز بر جای خود باقی است. اگرچه Scrum به طور قابل توجهی پروژه‌های انبارهای داده و هوش تجاری (DW / BI) را بهبود می بخشد اما تجربه‌های Front-End و Back-End را پوشش نمی‌دهد. برای آشنایی با تقسیم بندی Front-End و Back-End جدول ۲ که طبقه‌بندی لازم را ارایه می‌کند را مشاهده کنید.

Front-End Applications
Business intelligence
Business analytics
Data visualization
Information delivery
Reporting
Dashboarding
User experience (UX)
Back-End Applications
Data warehousing
Data integration
Data transformation
Data management
DW/BI (front-end and back-ends combined)
Data analytics
Decision support systems
Executive information systems
Enterprise data warehousing*
Online analytical processing (OLAP)

جدول ۲ – Front-End و Back-End در انبارداده

در حالی که توسعه دهندگان BI در یک تیم معمولا می‌توانند ویژگی‌های جدید را در بازه‌های دو هفته‌ای ارایه کنند، توسعه دهندگان ETL در یک بازه چهارهفته‌ای ممکن است تنها موفق به انجام تعداد محدودی از داستان کاربری شوند. توسعه دهندگان BI فقط دارای یک یا دو لایه معماری هستند و نگرانی آن‌ها تنها در مورد معانی و داشبوردها هست. داده های ارائه شده توسط لایه معنایی کاملا تمیز، سازمان یافته و ذخیره شده برای ابزارهای BI هستند که نگرانی‌ها را برطرف می‌کند. توسعه دهندگان ETL، از سوی دیگر، باید چندین لایه معماری انبار داده را برای ماژول‌ها ایجاد کنند. اهداف آنها معادل تلاش برای ساخت چهار یا پنج برنامه پیچیده در یک زمان است. جدول ۳ چالش های اساسی شناسایی و راه‌حل رفع آن را پیشنهاد کرده است.

ردیف	چالش	راه حل تطبیق
۱	هر داستان کاربری برای یکپارچه‌سازی داده، بیش از دو هفته زمان می‌برد.	داستان‌های کاربران به "داستان‌های توسعه دهنده" تقسیم می‌شوند طوری که هر یک به یک لایه از معماری مرجع انبارداده مربوط می‌شود
۲	بسیاری از مهارت‌های مورد نیاز در تیم، به طور مطلوبی در بین اعضا برای رفع ازدحام مسائل موجود، به اشتراک گذاشته نشده	تعریف تعدادی نقش خاص برای DW / BI خاص در تیم برای پشتیبانی از توسعه‌دهنگان ستون مبوط به نقش‌های جدید به برد وظایف اضافه شود تا به وضوع فعالیت این نقش‌ها مشخص شود.
۳	مشتریان نیازمند دانستند این موضوع هستند که چه موقع ویژگی‌های وعده شده به دستشان می‌رسد و هزینه برآوری آن چقدر خواهد بود	به جای کار کردن بر روی یک Backlog کامل، قسمتی از آن را با توجه به سرعت جاری تیم انتخاب کرده و یک به روز رسانی بر روی تخمین جاری[۴۲] با توجه به قابلیت‌های همه‌ی تکرارهای آینده ارایه می‌شود.
۴	هیچ شریک تجاری به حد کافی طولانی در پروژه باقی نمی‌ماند تا به عنوان صاحب محصول معرفی شود.	یک نقش جدید به عنوان معمار پروژه از بین اعضای توسعه دهنده انتخاب می‌شود تا نیازمندی‌ها و طراحی را هدایت کند تا تیم فردی را به‌عنوان کسی که دید کلنگرانه نسبت به پروژه دارد در بلند مدت کنار خود داشته باشد.
۵	توسعه دهندگان انبارداده برای شروع برنامه‌نویسی یک ماژول نیازمند جزییات بیشتری از جزییات مطرح شده در برنامه‌ریزی هر تکرار هستند.	یک نقش تحلیل‌گر سیستم ایجاد می‌شود و بر عهده یکی از توسعه دهندگان تیم قرار می‌گیرد تا با ۸۰ درصد شناخت از نیازمندی‌ها پیش از آغاز کدنویسی، نگاشت بین نیازمندی‌ها و اهداف شکل دهد.
۶	تحلیل و طراحی برای هر داستان ممکن بیش از یک تکرار را نیازمند باشد.	یک کانال ارتباطی[۴۳] ایجاد می‌شود، طوری‌که متخصصان تحلیل و طراحی در هر تکرار، پیش از توسعه‌دهندگان کار انجام دهند.
۷	توسعه دهندگان گرایش به انجام تعداد زیادی از کارها در یک مرتبه هستند که باعث پیچیدگی تجزیه و تحلیل وابستگی روزانه می‌شود	برای برد فعالیت‌ها، مشابه روش کانبان در ستون کارهای در جریان محدودیت تعداد فعالیت قرار می‌گیرد به طوری که تیم جریان مداوم کار را از طریق کل فرایند توسعه حفظ می‌کند

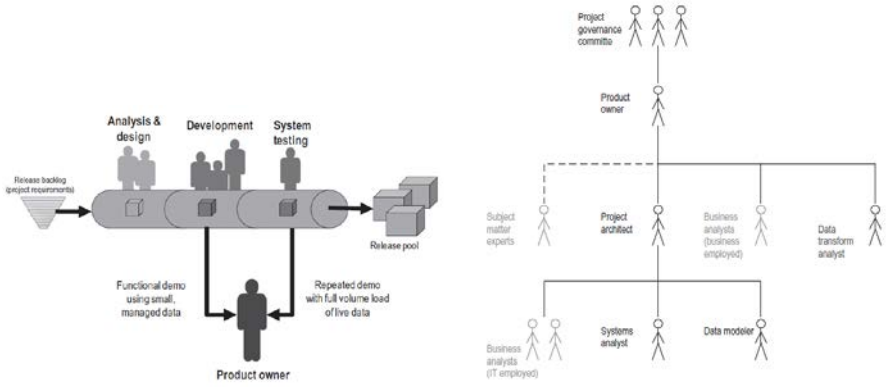
جدول ۳ - چالش‌ها و راه‌حل‌های برنامه‌نویسان ETL در Scrum

[42] Current Estimate
[43] pipeline

اسکرام به طور کلی نقش‌های بسیار کمی را در تیم تعریف می‌کند. در نسخه اصلی اسکرام یک تیم به یک شریک تجاری به عنوان صاحب محصول، یک استاد اسکرام و توسعه دهندگان نیاز دارد. وقتی برنامه نویسی شروع می‌شود، اسکرام به صورت خوش‌بینانه توصیه به خودسازماندهی تیم‌ها و نقش‌ها می‌کند. متاسفانه در پروژه‌های انبار داده و یکپارچگی داده، این خودسازماندهی می‌تواند فاجعه بیافریند. بنابراین در این مقاله قسمتی از وظیفه خودسازماندهی از عهده تیم‌ها برداشته شده و در مجموعه‌ای از نقش‌ها ثابت شده است که یک تیم خوب برای پروژه DW / BI نیاز دارد توصیه شده‌است. از آنجا ییکه این نقش‌های اضافی ابعاد مختلف تصمیم‌گیری های کلیدی را ارائه می دهند، می‌توان آنها را به صورت کلی به عنوان تیم به پروژه ارجاع داد. این نقش‌ها عبارتند از:

معمار پروژه[۴۴]، مدل کننده داده[۴۵]، تحلیل‌گر سیستم[۴۶]، تست کننده سیستم[۴۷]، متولی محصول واسطه[۴۸]، استاد اسکرام[۴۹]

دیگر نوآوری های ساده مربوط ۲۰/۸۰ نیازمندی‌ها و تولید داستان‌های توسعه‌دهندگان در تکرارها که منجر به برآورد فعلی پروژه می‌شود، است. همچنین در شیوه کانبان ایجاد دو تکرار جدی منفی یک و صفر به ابتدای پروژه به عنوان تکرار نیازمندی‌ها و تکرار تعریف آزمون صحت‌پذیر، از خلاقیت‌های شیوه ارایه شده است. فرآیند اصلاح شده در تصویر ۸ قابل مشاهده است.



Iteration	Project architect Solution reqts	Data modeler / sys analyst Technical reqts	Coders Potentially shippable	System test Shippable code
–1	A+			
0	B	A+		
1	C	B	A	
2	D	C	B	A
3		D	C	B
4			D	C

تصویر ۷ – روش‌های ابتکاری ارایه شده

[44] Project Architect
[45] Data Modeler
[46] Systems Analyst
[47] System Tester
[48] Proxy Product Owner
[49] Scrum Master

مهندسی نیازمندی‌ها

پروژه‌های چابک انبارده وقتی به بخش نیازمندی‌ها می‌رسند، مانند یم شمشیر دو دَم است همواره یک سوی تیز آن به سوی اجرا کنندگان قرار دارد. بدون یک تصور دقیق و صحیح از مشکلاتی که مشتریان با آن مواجه هستند و درگ قوی از مفاهیم اصلی که سیستم باید قادر به اجرای آن باشد، انرژی تیم برای تولید محصولات ناکارآمد و خروجی نامناسب صرف می‌شود. برای دستیابی به موفقیت، تیم‌های چابک باید رویکردی برای به دست آورن نیازمندی‌ها داشته باشند که معروف به "فقط به اندازه کافی خوب[۵۰]" است که به صورت تجزیه و تحلیل تا حدی است که تیم بتواند کار خود را پیرامون یک Backlog شروع نماید. تیم‌های انبار داده چابک قبل از شروع برنامه‌نویسی استراتژی‌های منحصر بفردی را جهت ایجاد تعادل بین حداقل و حداکثر نیازمندی‌های پروژه بکار می‌بندند. این استراتژی یک روش سبک وزن (روش ۲۰/۸۰)، افزایش‌پذیر و ریزبینانه (مشاهده از نقاط کرنری[۵۱]) برای تعریف موثر پروژه است. جدول ۴ گروه‌بندی نیازمندی‌ها را نمایش می‌دهد که در بررسی‌ها می‌توان به‌کار برد.

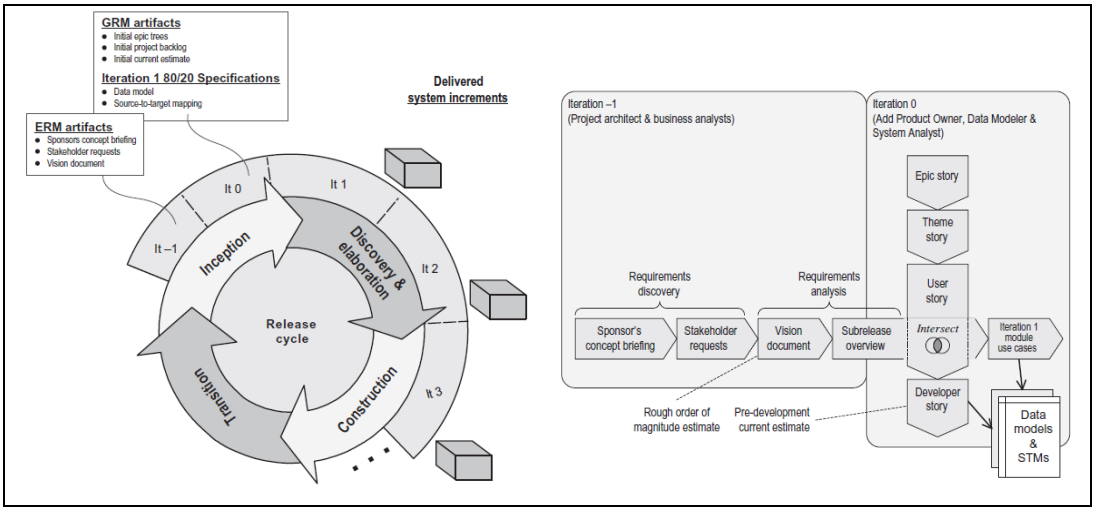
	Category	Subcategory
Functional	Functionality	Accuracy (ISO)
		Security (ISO)
		Suitability (ISO)
		Compliance (ISO)
	Usability	Accessibility
		Aesthetics
		UI Consistency
		Ergonomics
		Ease of Use
	Interface Requirements (+)	
Non-Functional	Reliability	Maturity (ISO)
		Recoverability (ISO)
		Availability
		Robustness
		Accuracy
		Fault Tolerance
		Safety
		Security
		Correctness
	Performance	Throughput
		Response Time
		Recovery Time
		Stop/Start Cycle Time
		Capacity
		Resource Utilization
Non-Functional (cont.)	Supportability	Testability
		Adaptability
		Maintainability
		Compatibility
		Configurability
		Upgradeability
		Installability
		Scalability
		Portability
		Reusability
		Interoperability
		Compliance
		Replaceability
		Changeability
		Analyzability
		Localizability
	Design Constraints (+)	
	Implementation Requirements (+)	
	Physical Requirements (+)	
	Documentation Requirements (RUP)	
	Licensing and Legal Requirements (RUP)	

All standards from FURPS unless otherwise marked [Grady 1992].
(+) Added later to FURPS, making it FURPS+
(ISO): Borrowed from [ISO 2011]
(RUP) Borrowed from Rational Unified Process. See [Zielczynski 2008] for details on elements listed.
Designations between functional and non-functional are generalizations—exceptional circumstances will occur

جدول ۴ – گروه‌بندی نیازمندی‌ها

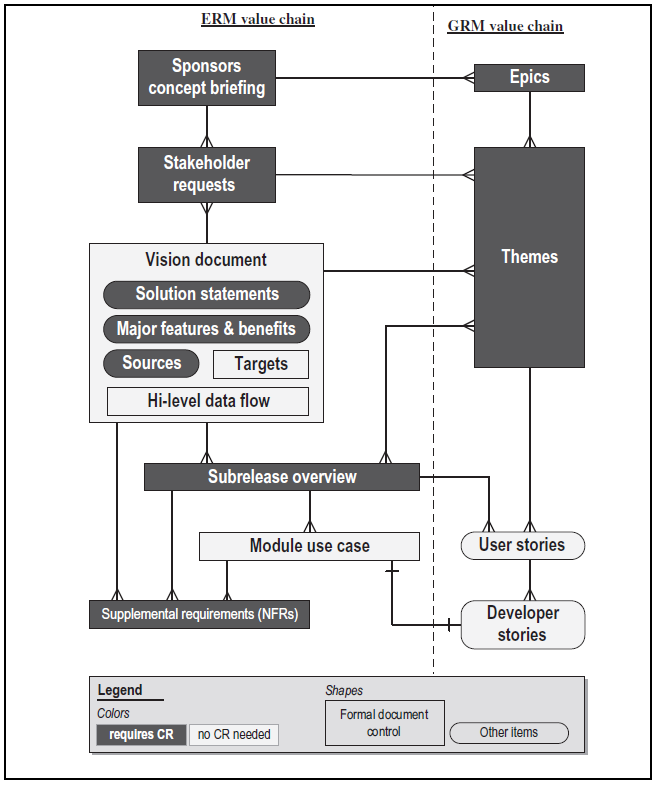
فرآیند نیازمندی‌ها و تغییراتی که در چرخه شناسایی نیازمندی‌ها به وجود می آید به خوبی در تصویر ۸ نمایش داده شده است.

[50] Just Good Enough
[51] See Around the Corners



تصویر ۸ – تغییرات ایجاد شده با وجود چرخه صفر و منفی یک با روش مهندس نیازمندی‌های چابک

روش و مدل کارکرد مهندسی نیازمندی‌های چابک با مفاهیم چابک مظیر اپیک، داستان کاربری و ... در تصویر ۹ به نمایش درآمده است.



تصویر ۹ – مهندسی نیازمندی‌های چابک با مفاهیم چابک

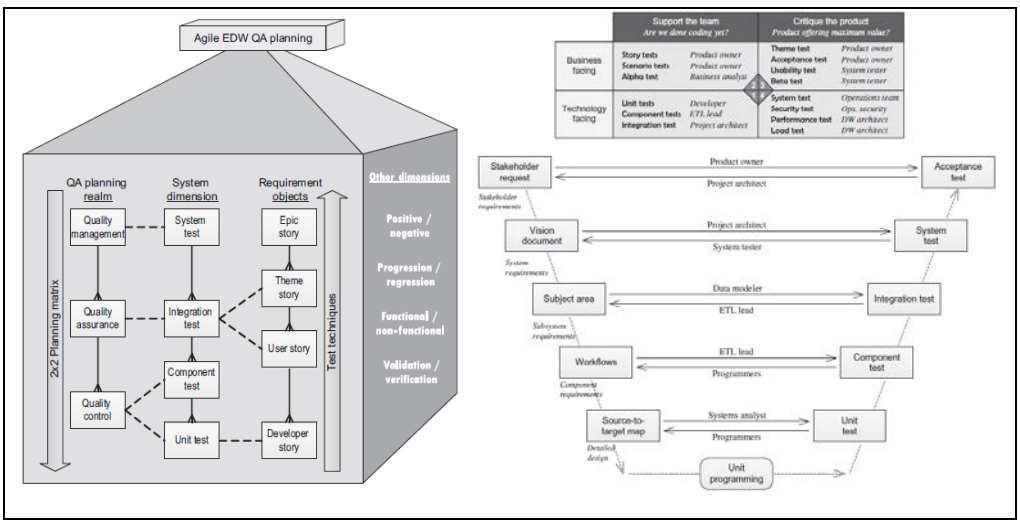
مهندسی داده

رویکرد چابک در مهندسی نیازمندی‌ها، به رهبران پروژه انبار داده بانکی اجازه می دهد تا با اهداف و تمایلات مورد نظر ذینفعان برای ذخیره‌سازی داده ها در پروژه DW / BI آشنا شوند. همان‌قدر که این تکنیک‌ها قدرتمند هستند، باید روش مهندسی پیاده‌سازی آنها در نظر گرفته شود تا نیازهای شناسایی شده تا به امروز را برآورده سازد و آنهایی که بعدا در طول روند تکامل تکراری ظاهر می شوند نیز پوشش داده شوند. در پروژه‌های ذخیره‌سازی داده‌های بانکی، چالش اصلی الگوی طراحی اولیه مدل داده است. یک مدل داده ضعیف طراحی می‌تواند بازدارنده روش چابک باشد و پروژه را با شکست مواجه کند. علاوه بر این، یک مدل داده ای که به خوبی طراحی شده است و جامع باشد، می تواند به طور غیرمستقیم هزینه‌ی بالایی داشته باشد. چنین تأثیرات می تواند مشتریان را به شدت ناامید کند و توسعه دهندگان را به یک دوره طولانی برنامه‌نویسی تحت استرس بکشاند تا برنامه‌های داده را تغییر داده و ماژول ها را برای پشتیبانی از نیازهای کسب و کار تغییردهند.

یک رویکرد هوشمندانه برای طراحی DW/BI، تیم را قادر می سازد تا انبارداده را به صورت بخش به بخش افزایش دهد و به این ترتیب هم تیم فنی و هم شرکای تجاری بتوانند طرح‌های پیاده‌سازی شده و باقی‌مانده را در طول پروژه بررسی کنند اهدافشان بدون تحمیل هزینه به مرور همسو شود. البته بیشتر متخصصان داده سنتی، با توجه به محدودیت‌های تکنولوژی پایگاه داده DW / BI که در حال حاضر در اختیار دارند، می‌گویند چنین هدفی غیر ممکن است. خوشبختانه، امروزه تکنیک‌های مدل سازی داده سازگار با ابزارهایی که آنها را پشتیبانی می کنند وجود دارند طوری که امروز امکان دارد یک بخش کوچک انبارداده در یک زمان مشخص ارایه شود و همچنین به طور سیال و سازگار با طراحی و داده بارگذاری شده پاسخ به بازخورد کاربر نهایی نیز تامین شود. این تکنیک‌ها و ابزارها که به تیم های انبار داده اجازه می دهند سریع و ارزان به صورت افزایشی کار کنند نیاز به پیش‌نیازهایی دارند تا بتوان در قالب یک معماری مرجع آن‌ها را بیان کرد.

تضمین کیفیت

با توجه به تعاریف انجام شده تضمین کیفیت به صورت چابک بسیار مهم خواهد بود. رویکرد اصلی در تصویر ۱۰ قابل مشاهده است:



تصویر ۱۰ – تضمین کیفیت چابک

تست های مورد نیاز و متولی آنها جدول ۵ به خوبی نشان داده شده‌است.

Ln	Type / Who	Product Owner	Project Architect	Business Analyst	Data Modeler	Systems Analyst	Programmer	System Tester	IT Support Team
1	Iteration Integration Test Data Prep		✓			✓		✓	
2	Unit Tests					✓	✓		
3	Component Tests					✓	✓		
4	Integration Tests		✓	✓	✓				
5	Story Tests	✓	✓	✓					
6	Scenario Tests	✓		✓					
7	Alpha Tests	✓		✓					
8	Theme Tests	✓		✓	✓				
9	Acceptance Tests	✓	✓	✓					
10	Usability Tests	✓		✓					
11	Beta Tests	✓		✓					
12	System Tests		✓		✓	✓			✓
13	Performance Tests		✓						✓
14	Load Tests		✓				✓		✓
15	Subrelease Candidate Presentations	✓	✓						
16	Operational Readiness Reviews							✓	✓

جدول ۵ – تست‌ها و متولیان آن‌ها در تضمین کیفیت چابک

جمع بندی

در ارایه راه‌کار پیاده‌سازی انبارهای داده به روش چابک، روشی ابتکاری ارایه و مرور شد. با توجه به نوع مطالب ارایه شده، هر کدام از چهار گام را می توان در قالب مقاله‌ای مجزا تا سطح جزییات شکافت و تبیین نمود. همچنین در جمع‌بندی چهار گام، گام پنجمی وجود دارد که به ارتباط و تعامل چهار گام مطرح شده می پردازد. این موضوع در مقاله‌ای مجزا توسط نگارندگان در دست تهیه می‌باشد.

منابع

[1] Ralph Hughes. (2013). *Agile Data Warehousing Project Management Business Intelligence Systems Using Scrum* (First edition). Elsevier Inc.
[2] Ralph Hughes, MA, PMP, CSM (2016). *Agile Data Warehousing for the Enterprise* (First edition). Elsevier Inc.
[3] Roger S. Pressman (2015). *Software Engineering: a practitioner's approach* (Eighth edition). McGraw-Hill.
[4] Elizabeth Hull, Ken Jackson, Jeremy Dick (2011). *Requirements Engineering* (First edition). Springer.
[5] IIBA (2015). *A Guide to the Business Analysis Body of Knowledge* (Third edition). International Institute of Business Analysis.