دانشکده فنی و مهندسی
کارشناسی ارشد مهندسی کامپیوتر نرم‌افزار
گروه مهندسی کامپیوتر و فناوری اطلاعات

گزارش درس سمینار

موضوع:

متدولوژی‌های ترکیبی در توسعه نرم‌افزار در سازمان‌ها و صنایع بزرگ

(مطالعه موردی فولاد سبا)

نگارش:

محمد کدخدایی الیادرانی

استاد راهنما:

دکتر سید علی رضوی ابراهیمی

بهمن ۱۳۹۹


بسم الله الرحمن الرحيم

چکیده

وقتی یک پروژه نرم‌افزاری را شروع می‌کنیم، انتخاب یک مدل توسعه مناسب باعث افزایش میزان موفقیت انجام آن پروژه می‌شود. مدل مناسب کمک می‌کند پروژه در زمان مشخص شده با بودجه مقرر مطابق با نیازهای سازمان به پایان برسد. با این حال نسخه ثابتی برای این کار وجود ندارد. در طول این سال‌ها چرخه حیات نرم‌افزار، به دلیل فراگیر بودن تغییرات از مدل‌های تجویزی و سنتی به سمت الگوهای تطبیقی تکرارشونده و افزایشی در قالب تیم‌های خودسازمانده و فرا وظیفه‌ای حرکت کرده است. تیم چابک قادر است به تغییرات پاسخ مناسب و به‌موقع بدهد و رضایت مشتری را جلب نماید.

در این پژوهش، در گام نخست به کمک چارچوب تصمیم‌گیری کانه‌وین ویژگی‌های محیط توسعه شناسایی می‌شود، سپس مزایای استفاده از چارچوب اسکرام و روش کانبان برای محیط بررسی می‌شود، بر اساس فرایند معماری سازمانی و حرکت سازمان به سمت چابکی، متدولوژی ترکیبی اسکرامبان‌فال پس از نتایج موفق به‌کارگیری متدولوژی اسکرام‌بان به‌عنوان مدل توسعه نرم‌افزار انتخاب شده است. دلایل چنین انتخابی و مراحل انجام آن در این تحقیق مورد بررسی قرار گرفته است. از نتایج حاصل در پروژه‌های اتوماسیون سطح دو فولاد سبا بهره گرفته شده است.

کلمات کلیدی: چابک، متدولوژی ترکیبی، اسکرامبان‌فال، توسعه نرم‌افزار، معماری سازمانی

فهرست مطالب

عنوان ..... صفحه

فصل اول: مقدمه

فهرست اشکال

عنوان ..... صفحه

فهرست جداول

عنوان ..... صفحه

فهرست علائم اختصاری

ASD: Agile software development	توسعه نرم‌افزاری چابک
BDD: Behavior-Driven Development	توسعه رفتار محور
DDD: Domain-driven design	طراحی دامنه محور
EPF: Eclipse Process Framework	چارچوب فرایند اکلیپس
KPI: Key Performance Indicator	شاخص کلیدی عملکرد
PMI: Project Management Institute	مؤسسه مدیریت پروژه
SPL: Software Product Line	خط تولید نرم‌افزار
SPrL: Software Process Line	خط فرایند نرم‌افزار
TDD: Test-Driven Development	توسعه آزمون محور
TOC: Theory of Constraint	تئوری محدودیت‌ها
TOGAF: The Open Group Architecture Framework	چارچوب معماری سازمانی اُپن گروپ (توگف)
WIP: Work in Progress	کارهای در حال انجام

فصل اول

مقدمه

مقدمه

اکنون که جنبش چابک به سازمانهای بزرگتر در صنایع بیشتری گسترش یافته است، شاهد تغییرات زیادی هستیم. مسلم است که ما از چارچوب ها، تکنیک ها و روش های مختلفی استفاده می کنیم، از برنامه‌نویسی مفرط گرفته تا اسکرام، کانبان و تحویل مداوم را بکار می‌گیریم. با این حال، اخیرا بیشتر و بیشتر در مورد استفاده از رویکردهای ترکیبی میشنویم.
در این تحقیق ضمن معرفی روش‌های سنتی توسعه نرم‌افزار و رویکردهای مطرح دنیای چابک به سراغ متدولوژی‌های ترکیبی رفته و مزایا و معایب استفاده از آنها در سازمان‌ها و صنایع بزرگ را مرور می‌کنیم. ناگفته پیداست خیلی از این روش‌ها هنوز در مرحله آزمون‌وخطا قرار دارد و معرفی آنها به منزله تأیید بکارگیری چنین روش‌هایی نیست.

۱-۱ تعریف مسئله و بیان سؤال‌های اصلی تحقیق

نرم‌افزار[1] محصولی است که مهندس نرم‌افزار طراحی کرده و می‌سازد. در حقیقت اشخاص در دنیای صنعت و زندگی روزمره چه مستقیم و چه غیرمستقیم از آن استفاده می‌کنند. وقتی کار می‌کنیم تا یک سیستم یا یک محصول بسازیم، حتماً باید یک سری مراحل قابل پیش‌بینی را بررسی کنیم: نقشه راهی که در ایجاد نتیجه‌ای با کیفیت بالا و به‌موقع ما را یاری می‌کند را فرایند توسعه نرم‌افزار[2]می‌نامیم (Pressman, 2005). مهندسان نرم‌افزار و مدیران آنها، فرایند را با نیازهای خود مطابقت داده سپس آن را دنبال می‌کنند. بعلاوه، کسانی که نرم‌افزار را درخواست کرده‌اند و عموماً با نام ذی‌نفعان[3]شناخته می‌شوند، در فرایند نرم‌افزار نقش دارند. اهمیت این موضوع از آنجا ناشی می‌شود که باعث ثبات، کنترل و سازماندهی فعالیتی می‌شود که اگر به حال خود گذاشته شود ممکن است باعث آشوب شود. یک رویکرد نوین در مهندسی نرم‌افزار باید چابک[4]باشد. تنها باید آن دسته از فعالیت‌ها، کنترل‌ها و محصولات کاری را طلب کند که مناسب تیم پروژه و محصولی باشد که قرار است تولید شود.

از دیدگاه مهندس نرم‌افزار، حاصل کار، برنامه‌ها، داده‌ها و مستنداتی است که به‌عنوان نتیجه‌ای از فعالیت‌های مهندسی نرم‌افزار مشخص شده توسط فرایند، تولید می‌شوند. برای اطمینان از اینکه درست از عهده کار برآمده‌ایم، چند راهکار ارزیابی نرم‌افزار وجود دارد که سازمان‌ها را قادر به تعیین بلوغ فرایند نرم‌افزار می‌سازد. ولی کیفیت، به‌موقع بودن و کارایی در درازمدت بهترین ملاک‌ها برای بازدهی فرایند مورداستفاده شما هستند.
در حقیقت فرایند، مجموعه‌ای از فعالیت‌ها، کنش‌ها و وظایف است که هنگام ایجاد یک محصول کاری اجرا می‌شوند. در حیطه مهندسی نرم‌افزار، فرایند، دستورالعمل نهایی برای چگونگی ساخت نرم‌افزارهای کامپیوتری نیست بلکه یک روش انطباق‌پذیر است که افراد تیم نرم‌افزار به کمک آن می‌توانند مجموعهٔ مناسبی از کنش‌ها و وظایف کاری را برگزینند. در سطح مشروح، فرایندی که بر می‌گزینیم به نرم‌افزاری که می‌خواهیم بسازیم، بستگی دارد. چارچوب فرایند با تعیین تعداد کوچکی از فعالیت‌های چارچوبی که برای کلیه پروژه‌های نرم‌افزاری قابل استفاده باشند، صرف نظر از اندازه و پیچیدگی آنها، شالوده‌ای برای یک فرایند مهندسی

[1] Software
[2] Software development process
[3] Stakeholders
[4] Agile

نرم‌افزار کامل پی‌ریزی می‌کند. به علاوه، چارچوب فرایند شامل مجموعه‌ای از فعالیت‌های چتری می‌شود که در سرتاسر فرایند نرم‌افزار قابل اعمال هستند.

امروزه صنعت عظیم نرم‌افزار به عاملی تعیین‌کننده در اقتصاد جهان صنعتی تبدیل شده است. تیم‌های متخصصان نرم‌افزار که هر یکروی بخشی از فناوری لازم برای تحویل یک برنامهٔ کاربردی پیچیده کار می‌کنند، جایگزین برنامه‌نویسان تنهای گذشته شده‌اند و هنوز همان سؤال‌هایی که از آن برنامه‌نویسان تنها پرسیده می‌شدند، همان‌هایی هستند که هنگام ساخته‌شدن سیستم‌های کامپیوتری مدرن و به‌روز پرسیده می‌شوند.

- چرا به پایان رساندن یک نرم‌افزار این‌قدر وقت می‌گیرد؟
- چرا ساخت نرم‌افزار هزینهٔ بالایی دارد؟
- چرا نمی‌توانیم همهٔ خطاها را پیش از تحویل برنامه به مشتری بیابیم؟
- چرا برای نگهداری یک برنامه موجود، این میزان وقت و هزینه صرف می‌کنیم؟
- چرا در اندازه‌گیری پیشرفت ساخت و نگهداری نرم‌افزار، دچار مشکل می‌شویم؟

نرم‌افزارهای موردنیاز برای سیستم‌هایی با فناوری‌های به‌روز، هرسال از سال قبل پیچیده و پیچیده‌تر می‌شوند و اندازه برنامه‌های نوشته شده نیز به همان نسبت رشد می‌کند. رشد سریع اندازه یک برنامه با سایز متوسط ممکن است مشکل چندانی ایجاد نکند اما برای پروژه‌های بزرگ با افزایش اندازه برنامه، تعداد افرادی که باید روی توسعه و نگهداری آن کار کنند افزایش می‌یابد. درعین‌حال با افزوده شدن بر تعداد نفرات یک تیم، بهره‌وری گروه ممکن است دچار خدشه شود. یک راه‌حل برای غلبه بر این مشکل، ایجاد چند تیم مهندسی نرم‌افزار است. به‌موازات رشد تعداد تیم‌های نرم‌افزاری، برقراری ارتباط بین این تیم‌ها نیز دشوار می‌شود و زمان بیشتری صرف این ارتباطات خواهد شد. برای غلبه بر این مسئله باید شاهد تغییرات بنیادی در شیوه برقراری ارتباط میان افراد و تیم‌ها با یکدیگر باشیم.

درعین‌حال با چالش‌هایی چون نحوه انتخاب متدولوژی مؤثر و کارا برای توسعه برنامه‌های خود مواجه شده‌ایم. متدولوژی سیستم‌های اطلاعاتی را این‌گونه تعریف کرده‌اند: مجموعه پیشنهادی فلاسفه، فازها، رویه‌ها، قوانین، تکنیک‌ها، ابزارها، مستندات، مدیریت و آموزش توسعه‌دهندگان (مولاناپور و همکاران، ۱۳۹۱). با استفاده از این تعریف، پیشنهاد شده که متدولوژی، دارای اجزایی به شرح ذیل باشد:

- چه کارهایی باید در هر مرحله انجام شود.
- چه نتایجی باید حاصل شود.
- امور مربوطه باید در چه زمانی و تحت چه شرایطی انجام شوند.
- کدام محدودیت‌ها باید اعلام شوند.
- چه افرادی در این کار باید دخیل باشند.
- نحوه مدیریت و کنترل پروژه چگونه باید باشد.
- از کدام ابزارهای پشتیبان باید استفاده شود.
- تأمین نیازهای آموزشی کاربران و نحوه تعیین آن مشخص شود.

برای تولید یک محصول نرم‌افزاری همواره مهم‌ترین نیاز در اختیار داشتن یک متدولوژی یا چارچوب برای توسعه نرم‌افزار است (گلشاهی و جاودانی گندمانی، ۱۳۹۷). مهندسی نرم‌افزار چابک، تلفیقی از یک فلسفه و مجموعه‌ای از دستورالعمل‌های توسعه است. این فلسفه مشوق جلب رضایت مشتری و تحویل افزایشی نرم‌افزار از همان ابتدای پروژه است. توسعه چابک نرم‌افزار یا توسعه نرم‌افزاری چابک گروهی از متدهای توسعهٔ نرم‌افزار مبتنی بر تکرار و به شکل تدریجی است که در آنها، راه‌حل‌ها از طریق خودسازمان‌دهی و همکاری بین تیم‌های مختلف کاری، انجام می‌شوند. این روش برنامه‌ریزی تطبیقی، توسعه و تحویل تکاملی و رویکرد زمان بسته‌بندیِ تکرارشونده را ارتقا می‌بخشد و پاسخ‌های سریع و انعطاف‌پذیر برای انجام تغییرات را تقویت می‌کند. در واقع چابک‌سازی یک چارچوب مفهومی است که پیش‌بینی تعاملات در سراسر چرخهٔ توسعه را بهبود می‌بخشد. توسعه چابک را شاید به بهترین وجه بتوان مهندسی نرم‌افزار نامید. فعالیت‌های چارچوبی پایه – ارتباطات، برنامه‌ریزی، مدل‌سازی، ساخت و استقرار – همچنان به قوت خود باقی خواهد ماند، ولی به یک مجموعه وظایف کمینه تغییر شکل می‌دهند که تیم نرم‌افزاری را به سمت ساخت و تحویل هدایت می‌کنند (Pressman, 2005).

اکنون سازمان‌ها و صنایع بزرگ تصمیماتی برای حرکت به سمت چابک شدن دارند اما ساختارهای پیچیده درون سازمانی و مقاومت برخی از مدیران و نفرات از موانع اصلی چنین حرکتی به شما می‌رود. بااین‌حال می‌توان راه میانه را در پیش گرفت و از یک روش ترکیبی بهره برد. در روش‌های ترکیبی ساختار سازمان بلافاصله دچار تغییر نخواهد شد و ضمن حفظ روش‌های سنتی به‌تدریج پیوندهایی با روش‌های چابک برقرار خواهد شد.

سؤالاتی که در این تحقیق در پی پاسخ به آن هستیم:

سؤال ۱: تفاوت روش‌های سنتی توسعه نرم‌افزار با رویکرد چابک و تفکر ناب چیست؟
سؤال ۲: در سازمان‌ها و صنایع بزرگ چگونه از مزایای چابک به کمک متدولوژی‌های ترکیبی بهره ببریم؟
سؤال ۳: متدولوژی ترکیبی اسکرام‌بان‌فال تا چه اندازه در حرکت سازمان‌ها به سمت چابکی مؤثر بوده است؟

سؤال اول در پایان فصل دوم و با برشمردن معایب روش‌های سنتی پاسخ داده می‌شود. سؤال دوم در فصل سوم با برشمردن روش‌های چابک و بررسی مزایا و معایب هرکدام پاسخ داده خواهد شد، در آنجا خواهیم دید چگونه شرکت‌های بزرگ به‌خوبی از مزایای این روش‌ها برای موفقیت در فضای رقابتی بهره برده‌اند. سؤال سوم به فصل چهارم تحقیق مرتبط می‌شود و به‌تفصیل به آن خواهیم پرداخت.

۱-۲ سابقه و ضرورت انجام تحقیق

در فصل آینده نگاهی به روش‌های توسعه سنتی نرم‌افزار خواهیم انداخت و معایب آن در دنیای مدرن را تشریح خواهیم کرد. متدهای توسعه افزایشی به سال ۱۹۵۷ برمی‌گردند. در سال ۱۹۷۴ مقاله‌ای در ارتباط با فرایند توسعه تطبیقی نرم‌افزار انتشار یافت. نهایتاً در اواسط دهه ۱۹۹۰ متدهای توسعه چالاک یا بعدها چابک به‌صورت یک عکس‌العمل در مقابل متدهای سنگین آبشاری

مطرح شدند. نتیجه‌ی ضعف‌هایی که در طول سال‌ها شناسایی و تجربه شده بود باعث شد در سال ۲۰۰۱، کنت بک[1] و شانزده نفر دیگر از سازندگان نرم‌افزار، نویسندگان و مشاوران (از آنها به عنوان اتحادیه چابک یاد می‌شود) بیانیه توسعه‌ی نرم‌افزاری چابک را امضا کنند. مفاد این بیانیه به قرار زیر بود:

- افراد و تعامل‌ها را بر فرایندها و ابزارها
- نرم‌افزار عملیاتی را بر مستندات جامع
- همکاری با مشتری را بر مذاکره قرارداد
- پاسخ به تغییر را بر دنبال‌کردن یک برنامه برتری دهیم.

یعنی درحالی‌که در آیتم‌های طرف چپ هم‌ارزش وجود دارد، به موارد طرف راست ارزش بیشتری خواهیم داد. بیانیه‌ها معمولاً به جنبش‌های سیاسی مربوط می‌شوند – چیزی که به سیستم قدیمی حمله می‌کند و تغییری انقلابی را پیشنهاد می‌کند. از جهاتی، این دقیقاً چیزی است که توسعه‌ی چابک با آن سروکار دارد (Pressman, 2015).

بیانیه چابک[2] بر ۱۲ اصل استوار است، در پیمان چابک[3] دوازده اصل برای کسانی که می‌خواهند به چابکی دست پیدا کنند، تعریف شده است:

- جلب رضایت مشتری از طریق تحویل زودهنگام و مستمر نرم‌افزارهای ارزشمند
- استقبال و پذیرا بودن تغییرات در خواسته‌ها حتی در اواخر فرایند توسعه
- تحویل زودبه‌زود نرم‌افزارهای کاری از ۲ هفته تا دو ماه که بازه زمانی کوتاه‌تر باید در اولویت قرار گیرد.
- دست‌اندرکاران و ذی‌نفعان باید در سرتاسر پروژه هر روز با همدیگر کار کنند.
- سپردن کار به افراد با انگیزه، فراهم‌کردن محیط و پشتیبانی موردنیاز آنها
- گفتگوی رودررو مؤثرترین روش انتقال اطلاعات به درون و بیرون تیم توسعه است.
- میزان اصلی در سنجش پیشرفت، نرم‌افزار کاری است.
- توسعه پایدار که قادر به حفظ سرعت ثابت در پیشرفت کار باشد.
- توجه مستمر به برتری فنی و طراحی خوب
- سادگی – هنر به حداکثر رساندن کارهایی که انجام نشده – ضروری است.
- بهترین معماری‌ها، خواسته‌ها و طراحی‌ها از تیم‌های خودسازماندهی شده ظهور می‌کنند.
- در فواصل منظم تیم بر چگونگی مؤثرتر شدن فکر می‌کند و رفتار را بر اساس بازتاب این تفکر تنظیم و همسو می‌کند.

البته این ۱۲ اصل در همه فرایندهای چابک با وزن برابر به کار برده نمی‌شوند، و در برخی مدل‌ها از اهمیت یک یا چند اصل چشم‌پوشی می‌شود یا دست‌کم نقش آنها کم‌رنگ‌تر می‌شود. نتیجه پیدایش چارچوبی چون اسکرام و یا متدولوژی‌ای مانند

[1] Kent Beck
[2] The Agile Manifesto
[3] Agile Alliance

اسکرام‌بان از دل همین تفکر است. اسکرام، در سال ۱۹۸۶ توسط هیروتاکا تاکوچی[1] و ایکوجیرو نوناکا[2] به‌عنوان یک خط‌مشی جدید برای تولید نرم‌افزارهای تجاری که باید قابلیت سرعت در تولید و انعطاف‌پذیری را داشته باشند، عرضه گردید. در سال ۱۹۹۵ کن شوئیبر[3] و جف سادرلند[4] از آن به‌عنوان یک روش تولید نرم‌افزار استفاده کردند. نقطه عطف مهم در این میان به سال ۲۰۰۱ و انتشار بیانیه چابک توسط ۱۷ توسعه‌دهنده مطرح دنیای نرم‌افزار شروع شد. در سال‌های پس از انتشار بیانیه چابک میل به ترکیب تفکر ناب و چابک بیشتر شد. در این میان، گام‌ها برای سازمان‌ها و صنایع بزرگ و سنتی کمی پیچیده‌تر است و به متدولوژی‌هایی برای حرکت به سمت چابکی نیاز است که ساختار غیرمنعطف و خشک چنین سازمان‌هایی را به‌خوبی شناسایی کرده باشد. متدولوژی اسکرامبان‌فال برآمده از نیازهایی از همین نوع است که در این تحقیق بیشتر به بررسی آن خواهیم پرداخت.

۱ –۳ هدف‌ها

در طول این سال‌ها شرکت‌های زیادی به سمت استفاده از اسکرام[5] حرکت کرده‌اند. برای حرکت به سمت این چارچوب نیازمند تغییرات رادیکالی زیادی هستیم و نمی‌توان آن را با حفظ ساختار قبلی به کار گرفت و باید ساختار و نقش‌های جدیدی در سیستم تعریف شوند. اما حقیقتاً اسکرام درمانی بر هر نوع مشکلی نیست و نمی‌توان از این چارچوب به‌تنهایی برای رفع هر مشکلی استفاده کرد. گزینه چابک دیگری که همیشه پای آن در محافل چابک در میان است، کانبان[6] است. کانبان که ذات آن به متدهای ناب[7] بازمی‌گردد از جمله متدهای چابک به‌حساب می‌آید.

کانبان برعکس اسکرام تأکیدی بر روی تغییرات رادیکالی ندارد و بیشتر سعی می‌کند به ساختار و نقش‌های فعلی سازمان احترام بگذارد و تحول این سازمان به‌صورت تدریجی و با استفاده از همان ساختار صورت بگیرد. بعلاوه این متد به‌هیچ‌وجه تجویزی نیست و حتی یک‌مرتبه بالاتر از اسکرام برای مشکلات سازمان و تیم‌ها راه‌حلی ندارد جز نشان‌دادن آن‌ها با ایجاد یک محیط شفاف.

هیچ ابزاری کامل نیست و هیچ ابزاری هم بی‌عیب نیست. مانند هر ابزار دیگری، اسکرام و کانبان هم هیچ‌کدام کامل و بی‌عیب نیستند. آن‌ها به شما همهٔ کارهایی که باید انجام دهید را نمی‌گویند بلکه فقط راهنمایی کرده و حدودی را برایتان فراهم می‌کنند. مثلاً اسکرام شما را محدود می‌کند تکرارهایی با مدت ثابت و تیم‌های فرا وظیفه‌ای داشته باشید و کانبان شما را مقید می‌کند از تابلوهای قابل‌مشاهده استفاده کنید و طول صف‌هایتان را محدود سازید. نکته جالب اینجاست که ارزش هر ابزاری در محدودکردن گزینه‌های شماست. یک ابزار فرایند که به شما اجازه دهد هر کاری انجام دهید، به‌اندازه کافی مناسب نیست. استفاده از ابزار درست به شما کمک می‌کند موفق شوید، هرچند موفقیت را تضمین نمی‌کند. لازم است موفقیت پروژه و موفقیت ابزار با هم اشتباه گرفته نشوند. از ترکیب این متدولوژی‌ها به روش‌های جدیدی دست می‌یابیم. امروزه، اسکرام‌بان یک چارچوب مدیریتی است که وقتی تیم‌های مختلف، اسکرام را به‌عنوان روش انتخابی خود برمی‌گزینند، از روش کانبان به‌عنوان یک دریچه برای مشاهده، درک و بهبود مداوم چگونگی انجام کار استفاده می‌کنند. بااین‌حال نیاز به مستندسازی در سطحی بالاتر از روش‌های چابک، درک کلی و

[1] Hirotaka Takeuchi
[2] Ikujiro Nonaka
[3] Ken Schwober
[4] Jeff Sutherland
[5] Scrum
[6] Kanban
[7] Lean

اولیه از مجموعه نیازمندی‌های پروژه در سازمان‌ها و صنایع بزرگ لزوم بهره‌گیری از متدولوژی‌ای که این موارد را در نظر بگیرد به‌عنوان هدف این پژوهش شناخته می‌شود.

۱-۴ چه کاربردهایی از انجام این تحقیق متصور است؟

ابزارهای فرایند در طول این سال‌ها مدام شاهد توسعه و تغییروتحول بوده‌اند، در یک مقایسه بر اساس مقیاس تجویزی - تطبیقی موارد ذیل به‌روشنی نمایان می‌شود:

فرایند یکپارچه گویا[1] کاملاً تجویزی است به این معنی که بیش از ۳۰ نقش، ۲۰ فعالیت و افزون بر ۷۰ دستاورد در آن وجود دارد که این یعنی چیزهای بسیاری برای یادگرفتن و تجربه‌کردن وجود دارد. البته قرار نیست در توسعه یک پروژه از همه آن‌ها استفاده شود و باید زیرمجموعه مناسبی برای پروژه انتخاب شود. این کار در عمل بسیار سخت است. برنامه‌نویسی مفرط[2] هم نسبت به اسکرام تجویزی‌تر است. این روش علاوه بر تجویزات اسکرام شامل مقادیری تجربیات مهندسی مانند توسعه تست محور و برنامه‌نویسی جفتی نیز می‌باشد.

اسکرام کمتر از برنامه‌نویسی مفرط تجویزی است، چون هیچ تجربیات مهندسی خاصی را تجویز نمی‌کند. ولی درعین‌حال اسکرام تجویزی‌تر از کانبان است چون چیزهایی مثل تکرار و تیم‌های فرا وظیفه‌ای را تجویز می‌نماید. کانبان تقریباً همه چیز را باز می‌گذارد. تنها محدودیت‌ها این‌ها هستند: گردش کارتان را ویژوال کنید و تعداد کارهای در جریان را محدود کنید. در واقع فقط مقدار کمی با عبارت هر کاری می‌خواهی انجام بده فاصله دارد، ولی هنوز هم بسیار قدرتمند است. اسکرام‌بان از مزایای موجود در اسکرام و کانبان به‌خوبی بهره می‌برد.

شاید کاربرد عمده همه این بحث‌ها این جمله کلیدی است: خودتان را به یک ابزار محدود نکنید. ابزارها را به هر صورت که لازم دارید با هم یکی کنید. باید دید چه چیزی به درد شما و پروژه‌ای که درحال‌توسعه آن هستید می‌خورد. براین‌اساس برای سازمان‌هایی که تغییر چارت سازمانی و سایر تغییرات در کوتاه‌مدت ممکن نیست، ترکیب این روش‌ها به بهبود کارایی در فرایند توسعه نرم‌افزار می‌انجامد. نمونه مشخص این موضوع در مورد تعداد زیادی از شرکت‌های ایرانی نیز متصور است که می‌توان با شیب ملایمی تغییرات را در سازمان و تیم‌های توسعه ایجاد کرد.

در نهایت این تحقیق می‌تواند به‌عنوان راهنمایی برای انتخاب چارچوب کاری تیم‌های توسعه در شرکت‌ها با شرح ذیل به کار رود:

کانبان برای تیم‌ها و محیط‌هایی مناسب است که به‌صورت مرتب باتوجه‌به تعدد ورودی‌ها (از سمت مشتری، مدیران و ...) تمرکزشان از کارهای فعلی در حال انجام قطع می‌شود و کارهای جدید اولویت اجرا می‌گیرند. اسکرام برای تیم‌ها و محیط‌هایی مناسب است که ثبات نسبی در طی زمان انجام کار تعهد شده (اسپرینت[3]) داشته باشند و یا این ثبات توسط اسکرام‌مستر[4] با همکاری تیم و شرکت برایشان ایجاد شود. نهایتاً در محیط‌های با تغییرات زیاد پیش‌بینی نشده که مدام اولویت‌ها تغییر می‌کنند و جهت تمرکز بر تعداد کارهای در حال انجام استفاده از اسکرام‌بان مفید خواهد بود. اگر نیاز به ایجاد تصویری اولیه از نیازمندی‌ها و

[1] RUP
[2] XP
[3] Sprint
[4] Scrum Master

مستندسازی و تعریف فازهای نگهداری اهمیت دارد و سازمان همچنان ویژگی‌های سنتی خود را حفظ کرده اما در حال حرکت آرامی به سمت چابک است، متدولوژی ترکیبی اسکرامبان‌فال پیشنهاد میگردد.

۱-۵ روش انجام تحقیق

روش انجام تحقیق به‌صورت کتابخانه‌ای می‌باشد. منابع مورداستفاده شامل مقالات، تحقیقات علمی و پژوهشی، کتب و جستجوهای اینترنتی در زمینهٔ توسعه آبشاری، نرم‌افزاری چابک، چارچوب اسکرام، روش کانبان، تولید ناب و عوامل مؤثر در مدیریت و افزایش کیفیت پروژه‌های بزرگ نرم‌افزاری در سازمان‌هایی با نیازهای سیستمی اطلاعاتی درحال‌رشد بوده است. پروژه توسعه خط ۲ فولاد سبا از نواحی مجتمع فولاد مبارکه نیز فرصتی برای آزمودن مفاهیم مطالعه شده بود. ضعف‌های متدولوژی اسکرام‌بان در یک سازمان که حرکت آهسته‌ای از دیدگاه سنتی به سمت رویکرد چابک دارد باعث بکارگیری نسخه سفارشی شده از متدولوژی ترکیبی اسکرامبان‌فال گردید.

۱-۶ مراحل انجام تحقیق

در ابتدا مطالعه در زمینه توسعه نرم‌افزاری چابک و مفاهیم و کاربردها و مسائل مرتبط با آن پرداخته شد و مقایسه‌ای با روش‌های سنتی توسعه نرم‌افزار صورت‌گرفته است، در ادامه چارچوب اسکرام به‌عنوان یک چارچوب تکرارپذیر و افزایشی برای کنترل پروژه و مدیریت نرم‌افزار مورد بررسی قرار گرفت. سپس کانبان به‌عنوان یک سیستم زمان‌بندی برای رویکردهای تولید ناب و ساخت هم‌زمان مطالعه شد. در ادامه مزایا و معایب این روش‌ها در سازمان‌ها و پروژه‌های بزرگ در قالب مطالعه منابع، مقالات و کتب مختلف مورد بررسی قرار گرفت. مطالعه ترکیب این روش‌ها که دامنه وسیع روش‌های سنتی و برنامه محور تا رویکرد چابک را در برمی‌گرفت، نهایتاً به متدولوژی‌های اسکرام‌بان و اسکرامبان‌فال منتهی گردید و بر اساس محیط توسعه فولاد سبا یک متدولوژی سفارشی شده برای سازمان مربوطه تهیه گردید که لازم است نتایج حاصل از بکارگیری آن به کمک سنجه‌های نرم‌افزاری مناسب در پروژه‌های آتی اندازه‌گیری گردد.

فصـــل دوم

مفاهیم پایه توسعه نرم‌افزار

مقدمه

هنگامی‌که یک محصول نرم‌افزاری را توسعه می‌دهید، لازم است یک سری مراحل را در نظر داشته باشید، به نقشه راهی که در ایجاد نتیجه‌ای با کیفیت و به‌موقع شما را یاری می‌کند، «فرایند توسعه نرم‌افزار» گفته می‌شود. بحران نرم‌افزار[1] عبارتی است که در روزهای آغازین علوم رایانه، برای نمایش میزان پیچیدگی توسعه برنامه‌های مفید و بهینهٔ کامپیوتری در مدت‌زمان مشخص به کار می‌رفت. پیشرفت‌های سریع در حوزه سخت‌افزار و نیاز به برنامه‌های کاربردی پیچیده‌تر باعث به‌وجودآمدن بحران نرم‌افزار شد. فرایندهای توسعه نرم‌افزار پاسخی به این مشکلات بودند. پس از گذشت سال‌ها و مشاهده تحولات صنعت نرم‌افزار آنچه اکنون مشخص است این است که مدل‌های فرایند تجویزی در ابتدا برای نظم بخشیدن به آشوب و بی‌نظمی موجود در توسعهٔ نرم‌افزار پیشنهاد و تدوین گردیدند. درعین‌حال تاریخ نشان داده است که این مدل‌های سنتی به میزان معینی به کار مهندسی نرم‌افزار ساختار بخشیده‌اند و راهنمایی مؤثر برای تیم‌های نرم‌افزاری در طول این سال‌ها بوده‌اند.

در ضمن تجربه این سال‌ها این نکته را نیز مشخص کرده است که کار مهندسی نرم‌افزار و محصولی که از آن به دست می‌آید در این حالت در «آستانه آشوب» قرار می‌گیرد. بااین‌حال در این قسمت به بیان مفاهیم مدل‌های فرایند سنتی که حالتی تجویزی دارند می‌پردازیم تا نحوه تکامل این مدل‌ها به‌ویژه وقتی وارد مفاهیم چابک می‌شویم ترسیم گردد.

۲ – ۱ مدل فرایند کلی

همان گونه که در فصل قبل عنوان شد، فرایند به‌عنوان مجموعه‌ای از فعالیت‌های کاری، کنش‌ها، و وظایف تعریف می‌شود که در هنگام توسعه یک محصول نرم‌افزاری باید اجرا شوند.

در مدل فرایند کلی هر فعالیت چارچوبی حاوی مجموعه‌ای از کنش‌های مهندسی نرم‌افزار است. هر کنش مهندسی نرم‌افزار به‌وسیله مجموعه‌ای از وظایف تعیین می‌شود که مشخص می‌کند به چه وظایفی باید عمل شود، چه محصولاتی باید تولید شوند، به چه نقاط تضمین کیفیتی نیاز است و چه نقاط عطفی برای نشان‌دادن پیشرفت فرایند به کار گرفته خواهد شد (Pressman, 2005).

چارچوب فرایند کلی برای مهندسی نرم‌افزار، پنج فعالیت چارچوبی را تعریف می‌کند:

۱. ارتباطات

۲. برنامه‌ریزی

۳. مدل‌سازی

۴. ساخت

۵. استقرار

[1] Software Crisis

علاوه بر این فعالیت‌ها، مجموعه‌ای از فعالیت‌های چتری را در سرتاسر پروژه به کار می‌گیرد:

۱. کنترل و پیگیری پروژه
۲. مدیریت ریسک
۳. تضمین کیفیت
۴. مدیریت پیکربندی
۵. بازبینی‌های فنی

مسئله مهم دیگر مفهومی است به نام جریان فرایند، این جنبه در حالت کلی شرح می‌دهد که فعالیت‌های چتری و کنش‌ها و وظایفی که در داخل هر فعالیت چارچوبی رخ می‌دهند از نظر ترتیب زمانی چگونه مدیریت و سازمان‌دهی می‌شوند. بر اساس همین جریان فرایند دست‌کم ۴ حالت مختلف از ترتیب پنج فعالیت چارچوبی ایجاد می‌شود:

۱. جریان فرایند خطی: در این حالت هرکدام از پنج فعالیت چارچوبی به ترتیب اجرا می‌شود، به‌طوری‌که با ارتباطات شروع و با استقرار به پایان می‌رسد.

۲. جریان فرایند مبتنی بر تکرار: در این حالت پیش از رفتن به دور تکرار بعدی، ممکن است یک یا چند فعالیت تکرار شود.

۳. جریان فرایند تکاملی: در این حالت فعالیت‌ها به شیوه‌ای حلقوی اجرا می‌شوند. هر مدار از پنج فعالیت عبور می‌کند که نهایتاً به نسخه کامل‌تری از برنامه منتهی می‌شود.

۴. جریان فرایند موازی: در این حالت یک یا چند فعالیت به‌موازات سایر فعالیت‌ها انجام می‌شوند.

اگر پروژه نرم‌افزاری به‌واسطه داشتن ذی‌نفعان زیاد پیچیدگی بالاتری داشته باشد، فعالیت ارتباطات ممکن است کنش‌های متمایز بیشتری داشته باشد: آغاز، استخراج، شناخت، مذاکره، تعیین مشخصات و اعتبارسنجی. هرکدام از این کنش‌ها چندین وظیفه کاری و تعدادی محصولات کاری دارند.

۲ – ۲ مدل آبشاری و مدل وی

مدل آبشاری[1] که گاه از آن به‌عنوان چرخه حیات کلاسیک یاد می‌شود، روشی سیستماتیک و ترتیبی برای توسعه نرم‌افزار پیشنهاد می‌دهد. وقتی خواسته‌های مربوط به یک مسئله به‌خوبی شناخته شده‌اند و از پایداری مناسبی برخوردار باشند این روش به‌خوبی قابلیت استفاده دارد.

[1] Waterfall Model

فرایند توسعه نرم‌افزار آبشاری توسط دانشمند علوم کامپیوتر، وینستون رویس[1] در سال ۱۹۷۰ میلادی معرفی شد. اگرچه به طور مستقیم برای نام‌گذاری مدل خود از واژه آبشاری استفاده نکرد ولی مقاله وی در واقع در زمینه فرایند توسعه نرم‌افزار ارائه شد و در مدلی که او معرفی کرد اجازه تکرارهای بیشتری بین مراحل مختلف مدیر فراهم می‌گردید مدل واقعی روی قابلیت تکرارپذیری بیشتری داشت و فضای وسیع‌تری جهت انجام مانور بین مراحل مختلف مدل فراهم می‌آورد (خاتمی‌نژاد، ۱۳۹۵).

فرایند توسعه آبشاری به مراحل جداگانه تقسیم شده است که در آن نتیجه و خروجی هر فاز به‌عنوان ورودی فاز بعدی تلقی می‌شود. در فاز اول، نیازمندی‌ها و خواسته‌های مشتری و به نحوی تمام نیازهای ممکن برای سیستمی که قرار است توسعه داده شود ثبت و ضبط شده و در سندی با عنوان نیازمندی‌ها مستندسازی می‌شود. به طور معمول لازم است این سند به امضای ذی‌نفعان کلیدی پروژه و کسب‌وکار برسد این بخش از مدل آبشاری معمولاً توسط تحلیلگر سیستم سازماندهی می‌شود اما بسته به‌اندازه پروژه تیم و یا سازمان ممکن است دیگر اعضای تیم توسعه نیز درگیر شوند. در این فاز باید اطلاعات لازم درباره الزامات سیستم از ذی‌نفعان کسر شود این امر شامل قابلیت‌های موردنیاز، مستندات قوانین و فرایندهای کسب‌وکار و نیز الزامات قانونی و نظارتی است که بر سیستم کلی تأثیر می‌گذارد.

فاز بعدی طراحی سیستم است. در این مرحله، مشخصات موردنیاز تهیه شده در فاز اول، مورد بررسی قرار گرفته و براین‌اساس، سیستم طراحی می‌گردد. این طراحی کمک می‌کند تا الزامات موردنیاز در طراحی سیستم تعیین شود و معماری کلی سیستم نیز شکل بگیرد. در این فاز است که معماران، طراحان و توسعه‌دهندگان در کنار هم تصمیم می‌گیرند که چگونه سیستم کلی ساخته شود. ساخت چنین سیستمی از دیدگاه کدنویسی و انتخاب نوع تکنولوژی و نیز انتخاب نوع زیرساخت صورت می‌پذیرد.

فاز بعدی پیاده‌سازی است. این فازی است که در آن توسعه دهندگان طراحی را دریافت کرده و شروع به تولید کد می‌کنند تا طراحی رنگ واقعیت به خود بگیرد. البته توسعه‌دهندگان ممکن است در این فاز قطعات خودکار و تست یکپارچه‌سازی را نیز بنویسند. پس از فاز پیاده‌سازی نوبت به فاز یکپارچه‌سازی و تست می‌رسد. این همان جایی است که همه اقلام قابل‌قبول و قابل تحویل فاز پیاده‌سازی در کنار یکدیگر قرار داده می‌شوند و سیستم به‌صورت یکجا مورد تست قرار می‌گیرد. تیم تست باید بر اساس برنامه تست تعریف شده کار کند. هنگامی‌که سیستم آزمایش شد و صحت عملکرد آن توسط تیم تست تأیید گردید، فاز بعدی این است که سیستم مستقر شده و در اختیار کاربران نهایی قرار داده شود. این کاربران نهایی می‌توانند کاربرانی از داخل سازمان و یا مشتریان واقعی باشند. پس از آنکه سیستم استقرار یافت به فاز نگهداری می‌رسیم که در آن هرگونه نقص و اشکالی که گزارش داده شود رفع شده و سیستم دوباره استقرار می‌یابد که این کار معمولاً در قالب انتشار اصلاح وصله[2] انجام می‌شود. ممکن است در این فاز بهبودهای جزئی نیز در سیستم ایجاد شود. علاوه بر آن اگر دامنه بهبودی کاملاً وسیع باشد آنگاه ممکن است روند آبشاری از نو شروع شده و نیاز به کسب اطلاعات بیشتری درباره نیازمندی‌های مشتری باشد همه این مراحل به شکل یک جریان مداوم متمایل به سمت پایین و همانند یک آبشار به نظر می‌رسد. فاز بعدی تنها زمانی می‌تواند آغاز شود که مجموعه‌ای از اهداف از پیش تعریف شده در فاز قبلی تحقق یافته باشد. لازم به ذکر است که در این مدل مراحل با یکدیگر هم‌پوشانی ندارند.

شکل دیگری در نمایش مدل آبشاری به‌عنوان مدل وی[3] شناخته می‌شود. این مدل رابطهٔ تضمین کیفیت با کنش‌های مرتبط با ارتباط، مدل‌سازی و فعالیت‌های ساخت اولیه را تصویر می‌کند.

---

[1] Winston Royce
[2] Patch
[3] V Model

با حرکت تیم نرم‌افزاری به‌طرف پائین و سمت چپ نمودار، خواسته‌های اساسی مسئله رفته‌رفته پالایش می‌شوند و جزئیات بیشتری از آن‌ها تعیین می‌شوند و مسئله و راهکار آن بهتر نمایش داده می‌شود. هنگامی‌که کدها نوشته شد، تیم در طرف راست نمودار به‌طرف بالا حرکت می‌کند و اساساً یک سری آزمون اجرا می‌کند تا هرکدام از مدل‌های ایجاد شده در مدت حرکت تیم به‌طرف پائین را وارسی کند. (Pressman, 2005).

امروزه کار نرم‌افزاری با گام‌هایی سریع انجام می‌شود و در معرض جریانی بی‌پایان از تغییرات قرار دارد. مدل آبشاری غالباً برای چنین کاری نامناسب است، ولی در شرایطی که خواسته‌ها ثابت هستند و قرار است کار تا پایان به شیوه‌ای خطی پیش برود، می‌تواند به‌عنوان مدلی مفید عمل کند.
اما چه عواملی به طور عمده باعث ایجاد مشکل در چنین مدلی می‌شود؟ از جمله مشکلاتی که به هنگام اجرای مدل ترتیبی خطی پیش می‌آید، می‌توان به موارد زیر اشاره کرد:

۱. پروژه‌های واقعی بندرت جریان ترتیبی پیشنهاد شده توسط این مدل را دنبال می‌کنند.
۲. برای مشتری بیان واضح همه نیازهای خود کاری دشوار است.
۳. ازآنجایی‌که یک نسخه کاری[1] از برنامه‌ها تا آخرین روزهای پروژه در دسترس او قرار نمی‌گیرد باید صبور باشد.

ماهیت تکراری و افزایشی پیشنهاد شده در مفاهیم چابک از دل همین مشکلات در طول سال‌ها بروز کرد و به موفقیت‌های چشمگیری در توسعه سیستم‌های اطلاعاتی در سال‌های اخیر دست‌یافته است.

۲ – ۳ مدل‌های فرایند افزایشی

مدل افزایشی اجزاء مدل ترتیبی خطی را با جریان‌های فرایند خطی و موازی در هم می‌آمیزد. مثلاً در مواردی که به فراهم‌کردن سریع مجموعهٔ محدودی از عملکردهای نرم‌افزار برای کاربران و سپس پالایش و گسترش بر اساس آن عملکردها در نسخه‌های آتی نیاز باشد می‌توان از مدل فرایند افزایشی بهره برد. مدل افزایشی یک سری نرم‌افزار تحویل می‌دهد که هرکدام یک گام نامیده می‌شود و این گام‌ها هر یک نسبت به سلف خود عملکرد بیشتری در اختیار مشتری قرار می‌دهند.

هنگامی‌که از یک مدل افزایشی استفاده شود افزایش نخست غالباً محصول هسته‌ای است، یعنی به خواسته‌های پایه می‌پردازد، ولی بسیاری از ویژگی‌های مکمل تحویل داده نمی‌شوند. محصول هسته‌ای توسط مشتری مورداستفاده یا بازبینی مفصل قرار می‌گیرد. در نتیجه‌ی استفاده و یا ارزیابی، طرحی برای افزایش بعدی توسعه می‌یابد. این طرح حاوی اصلاحاتی است که نیازهای مشتری و تحویل قابلیت‌ها و ویژگی‌های اضافی را بهبود می‌بخشد این فرایند به دنبال تحویل هر قطعه تکرار می‌شود تا اینکه محصول کامل تولید شود.

[1] Working Software

مدل فرایند افزایشی، همانند مدل ساخت نمونه اولیه و روش‌های تکاملی دیگر ماهیتی تکراری دارد. ولی برخلاف مدل ساخت نمونه اولیه، مدل افزایشی بر تحویل قطعه‌ای در هر افزایش تأکید می‌ورزد. قطعات اولیه، نسخه‌های دست‌وپا شکسته‌ای از محصول نهایی هستند، ولی قابلیت ارائه خدمات به کاربر را داشته در ضمن به‌عنوان محیطی برای ارزیابی توسط کاربر نیز عمل می‌کنند. فلسفه فرایند افزایشی در تمامی مدل‌های فرایند چابک دیده می‌شود.

توسعه افزایشی به‌ویژه هنگامی مفید واقع می‌شود که تعداد کارمندان لازم برای تکمیل پیاده‌سازی پروژه در مهلت کاری مقرر، در دسترس نباشد. افزایش‌های اولیه را با تعداد کمتری از افراد می‌توان پیاده‌سازی نمود. اگر محصول هسته‌ای به‌خوبی دریافت شود، کارمندان دیگری را در صورت نیاز می‌توان اضافه کرد و افزایش بعدی را پیاده‌سازی کرد به‌علاوه می‌توان افزایش‌ها را طوری برنامه‌ریزی کرد که خطرات تکنیکی قابل مدیریت باشند (Pressman, 2005).

باتوجه‌به اینکه فلسفه فرایند افزایشی در تفکر چابک دیده می‌شود شایسته است ویژگی‌ها و مزایای این روش را با جزئیات بیشتری شرح دهیم. در کل می‌توان ویژگی‌های مدل افزایشی را به‌صورت زیر برشمرد:

۱. سیستم به تعداد زیادی ریز - پروژه شکسته می‌شود.
۲. بخش‌های سیستم ساخته می‌شوند تا سیستم نهایی را تولید کنند.
۳. نیازهای اساسی در اولویت اول قرار می‌گیرند و اول به آن‌ها رسیدگی می‌شود.
۴. نیازهای یک بخش تا زمانی که افزایش آن بخش توسعه نیابد غیرفعال است.

مزایای این مدل فرایندی:
۱. تحویل محصول اولیه سریع‌تر و کم‌هزینه‌تر است.
۲. مشتری می‌تواند به ویژگی‌ها پاسخ دهد و محصول را برای هر نیاز یا تغییرات مفید بازبینی کند.
۳. عموماً تست و رفع خطا در این مدل راحت‌تر از سایر مدل‌های توسعه نرم‌افزار سنتی است زیرا تغییرات ایجاد شده در هر تکرار نسبتاً اندک است.
۴. بعد از هر تکرار تست رگرسیون باید انجام شود. در طول این تست بخش‌های مشکل‌دار سیستم سریعاً تشخیص داده می‌شوند زیرا در هر تکرار تغییرات ایجاد شده اندک و مشخص است.

معایب این مدل فرایند را به شرح ذیل می‌توان برشمرد:
۱. هزینه نهایی ممکن است از بودجه سازمان بیشتر باشد.
۲. با ایجاد یک افزایش و عملکرد به محصول، مشکل مربوط به معماری سیستم ممکن است به وجود بیاید چیزی که در نسخه قبلی وجود نداشت.

۲ – ۴ مدل‌های فرایند تکاملی

نرم‌افزارها نیز همانند همه سیستم‌های پیچیدهٔ دیگر، با گذشت زمان تکامل می‌یابند. خواست‌ها و نیازهای تجارتی محصول غالباً به‌موازات توسعه، تغییر می‌یابند و منجر به ساخت محصول نهایی غیرواقعی می‌شوند. مهلت‌های زمانی محدودِ بازار، کامل کردن یک محصول نرم‌افزاری مفهومی را غیرممکن می‌سازند ولی یک نسخه محدود را باید وارد بازار کرد تا فشارهای رقابتی و کاری را مرتفع سازد، مجموعه‌ای از خواسته‌های اصلی و محوری سیستم و یا محصول به‌خوبی درک می‌شود ولی جزئیات محصول یا سیستم هنوز باید مشخص شود. در این وضعیت‌ها مهندسان نرم‌افزار به مدل فرایندی نیاز دارند که به طور مشخص برای محصول طراحی شده باشد و با گذشت زمان تکامل می‌یابد.

۲ – ۴ – ۱ ساخت نمونه اولیه

نمونه اولیه[1] پیش نمونه، الگوی اولیه یا پروتوتایپ یک نمونه ابتدایی، مدل یا محصول است که با هدف آزمایش یک محصول یا فرایند ساخته می‌شود. اگرچه از تهیه نمونه اولیه می‌توان به‌عنوان یک مدل فرایند مستقل استفاده کرد، از آن بیشتر به‌عنوان تکنیکی استفاده می‌شود که می‌توان در حیطه هرکدام از مدل‌های فرایند ذکر شده استفاده کرد. شیوه به کار گرفته شده هر چه که باشد تهیه نمونه اولیه به توسعه دهندگان و سایر طرف‌های ذی‌نفع کمک می‌کند که مسائل را بهتر درک کرده و در هنگام مبهم بودن خواسته‌ها مشخص شود چه چیزی قرار است ساخته شود. الگوی ساخت نمونه اولیه با جمع‌آوری خواسته‌ها آغاز می‌شود. مشتری و سازنده با هم ملاقات می‌کنند و اهداف کلی نرم‌افزار را تعیین می‌کنند همه خواسته‌های معلوم را شناسایی می‌کنند و سپس یک طراحی سریع صورت می‌پذیرد و نهایتاً طراحی سریع منجر به ساخت یک نمونه اولیه می‌شود. در اکثر پروژه‌ها نخستین سیستمی که ساخته می‌شود چندان قابل‌استفاده نیست ممکن است بیش از حد گند باشد، بیش از حد بزرگ باشد و استفاده از آن دشوار باشد؛ بنابراین چاره‌ای جز شروع دوباره وجود ندارد باید نسخه دیگری ساخت که این مشکلات در آن حل شده باشد. نمونه اولیه می‌تواند به‌عنوان «نخستین سیستم» عمل کند یعنی همان‌طور که بروکز توصیه می‌کند دور انداخته شود. دلیل این امر این است که افراد ذی‌نفع چیزی را می‌بینند که ظاهراً یک نسخه کاری از نرم‌افزار است ولی نمی‌دانند که این نمونه اولیه با «موم» سرهم‌بندی شده است.

۲ – ۴ – ۲ مدل مارپیچی (حلزونی)

مدل مارپیچی که نخستین‌بار بری بوهم[2] آن را پیشنهاد کرد یک مدل فرایند نرم‌افزاری تکاملی است که ماهیت تکراری مدل ساخت نمونه اولیه را با جنبه‌های کنترلی و سیستماتیک مدل ترتیبی خطی تلفیق می‌کند. طی نخستین دوره‌های تکرار ممکن است یک مدل کاغذی یا نمونه اولیه تهیه شود، تکرارهای بعدی هر بار نسخه کامل‌تری از سیستم مهندسی شده و تولید می‌گردد. برخلاف سایر مدل‌های فرایند کلاسیک که با تحویل نرم‌افزار پایان می‌یابند، مدل مارپیچی را می‌توان طوری تطبیق داد که در

[1] Prototype
[2] Barry W. Boehm

سرتاسر عمر نرم‌افزار کامپیوتری قابل به‌کارگیری باشد. مدل مارپیچی یک روش واقع‌گرا برای توسعه نرم‌افزارها و سیستم‌هایی در مقیاس انبوه است. ازآنجاکه نرم‌افزار به موازات پیشرفت فرایند تکامل می‌یابد سازنده و مشتری در هر سطح تکامل ریسک‌ها را بهتر درک کرده و به آن واکنش نشان می‌دهند. مدل مارپیچی از ساخت نمونه اولیه به عنوان راهکاری برای کاهش ریسک استفاده می‌کند.

۲ – ۵ مدل فرایند یکپارچه

ایوار جیکابسون[1] و گرادی بوچ[2] در کتاب خود با عنوان «فرایند یکپارچه[3]»، نیاز به یک فرایند نرم‌افزاری مبتنی بر موارد کاربرد،[4] معماری، تکرار و افزایش را موردبحث و بررسی قرار می‌دهند.

امروزه در نرم‌افزار، سیستم‌های پیچیده‌تر و بزرگ‌تر بیشتر مورد نظرند. این امر تا حدی از این واقعیت ناشی می‌شود که هرساله بر قدرت کامپیوترها افزوده می‌شود و در نتیجه انتظارات کاربران نیز از آنها بیشتر می‌شود. وقتی که در می‌یابیم یک محصول از نسخه‌ای به نسخه‌ی دیگر چقدر قابلیت بهبود دارد اشتهای ما برای نرم‌افزارهایی با پیچیدگی بیش از پیش رشد می‌کند. ما نرم‌افزارهایی می‌خواهیم که بهتر بر نیازهای ما منطبق باشد ولی این به نوبه خود فقط باعث پیچیدگی بیشتر می‌شود. فرایند یکپارچه از جهاتی تلاش برای گرد هم آوردن بهترین ویژگی‌ها و خصوصیات مدل‌های فرایند سنتی است. اوایل دهه ۱۹۹۰، ایوار جیکابسون، گرادی بوچ و جیمز رومباف کار روی یک روش یکپارچه را آغاز کردند که بهترین ویژگی‌های هر کدام از روش‌های طراحی و تحلیل شیءگرا را تلفیق می‌کرد. نتیجه، «زبان مدل‌سازی یکپارچه[5]» است که حاوی یک نمادگذاری قدرتمند برای مدل‌سازی و توسعه‌ی سیستم‌های شیءگراست.

بااین‌وجود جیکابسون به این نتیجه رسید که در اختیار داشتن یک زبان مدل‌سازی استاندارد به‌تنهایی کافی نیست. شما باید بدانید که چطور از آن استفاده کنید و این امر منجر به پیدایش نرم‌افزارهای مرتبط با فرایند توسعه و یا به عبارت دیگر متدولوژی شد. جیکابسون استفاده از واژه روش یا متدولوژی[6] را برای توصیف نمی‌پسندید. او اظهار می‌کند یک روش معمول شامل مجموعه‌ای از ایده‌های جالب و تشریح گام به گام و جامع است. همچنین تعدادی از نواقص این روش‌ها را به صورت زیر برشمرد:

۱. این روش‌ها تنها روی کاغذ و در کتاب وجود دارد.

۲. این روش‌ها به‌ندرت در پروژه‌های واقعی امتحان شده‌اند.

۳. این روش‌ها روی توسعه سیستم‌های جدید تمرکز می‌کنند و در ارتباط با توسعه تکاملی و یا نگهداری حرف زیادی برای گفتن ندارند.

۴. از لحاظ وجود علایم غنی بوده ولی از نظر بار معنایی ضعیف هستند.

---

[1] Ivar Jacobson
[2] Grady Booch
[3] UP – Unified Process
[4] Use case
[5] UML
[6] Methodology

به‌این‌ترتیب، RUP دارای تعدادی از چرخه‌هاست که با یکدیگر توسعه پروژه را شکل می‌دهند و در طول حیات آن‌ها جریان دارند. هر چرخه چهار فاز آغاز، رشد، ساخت و گذار دارد. نه فرایند اصلی گردشکار در اینجا وجود دارد. گردشکار جریانی از فعالیت‌هاست که نتیجه ارزش قابل‌مشاهده را تولید می‌کند. یکی از جنبه‌های RUP مفهوم کارگر است. کارگر در واقع فرد خاصی نیست بلکه کسی است که یک نقش را ایفا می‌کند و یا همان گونه که کراچتن عقیده دارد کسی است که «کلاه» مخصوص را در زمان مشخصی به سر دارد. تعاریف متفاوت آر.یو.پی در تعدادی از گردش‌های کاری شناسایی شده تفاوت‌هایی دارد. نه گردشکار توصیه شده توسط کراچتن شامل این موارد است: گردشکار مدل‌سازی کسب‌وکار، گردشکار نیازمندی‌ها، گردشکار تحلیل و طراحی، گردشکار پیاده‌سازی، گردشکار آزمایشی، گردشکار استقرار، گردشکار پیکربندی و مدیریت تغییر، گردشکار مدیریت پروژه، گردشکار محیطی. فرایند یکپارچه تلاش‌هایی برای گرد هم آوردن بهترین ویژگی‌ها و خصوصیات مدل‌های فرایند سنتی است و درعین‌حال آن‌ها را به شیوه‌ای مشخص می‌کند که بسیاری از بهترین اصول توسعهٔ نرم‌افزار چابک را پیاده‌سازی می‌کند. بااین‌حال در فصل بعدی نگاه کاملی بر توسعه نرم‌افزار چابک خواهیم انداخت و چارچوب‌ها و متدولوژی‌های مطرح آن را مرور کرده و کارهای انجام شده در این سال‌ها در این حوزه را با بسط بیشتری شرح می‌دهیم.

۲ – ۶ اهداف مطالعه

بررسی و مرور مفاهیم چابک و چارچوب‌های مطرح این روزها مانند اسکرام و اسکرام‌بان بدون مطالعه پیشینه آن‌ها، تصویر روشنی از تحول این روش‌ها در گذر زمان را نشان نمی‌دهد، باید در زمینه این‌که آیا برای یک جهان نرم‌افزاری که با تغییرات پیشرفت می‌کند، صرفاً مدل‌های سنتی و تجویزی کارایی دارند یا نه، مروری صورت می‌گرفت و مشکلات روش‌های سنتی و سیر تحول آن‌ها قدم‌به‌قدم معرفی می‌شد. به‌علاوه باید بدانیم گذار از روش‌های سنتی به‌سوی مفاهیم مدرن به معنای جایگزینی آنها با بی‌نظمی و یک فعالیت بدون برنامه‌ریزی نیست.

۲ – ۷ جمع‌بندی

در این فصل مفاهیم مدل‌های فرایند سنتی (تجویزی) مورد بررسی قرار گفت، نحوه شکل‌گیری تفکر چابک از دل این مدل‌ها و روش‌ها مورد بررسی قرار گرفت. مزایا و نقشی که این مفاهیم در توسعه چابک داشته‌اند بحث شد و درعین‌حال نیاز به روش‌هایی که نظم و درعین‌حال آزادی عمل را برای توسعه موفق نرم‌افزار لازم می‌داند مطرح کردیم. اکنون در پایان فصل دوم با مرور معایب روش‌های سنتی دلایل حرکت از روش‌های تجویزی به سمت روش‌های نوین و چابک روشن شده است و سؤال اول تحقیق پاسخ داده شده است.

فصـــل ســـوم

توسعه نرم‌افزاری چابک

مقدمه

در این فصل به بررسی متدولوژی‌ها، چارچوب‌ها، رویکردها، ابزارها و تکنیک‌ها در حوزه تفکر چابک خواهیم پرداخت. مزایا و معایب هرکدام از این روش‌ها بررسی می‌شود و تأکید می‌گردد نوع پروژه و ساختار سازمان در انتخاب یکی از این رویکردها تأثیر بسزایی دارد. بااین‌حال در گذر زمان تیم‌ها تجربه‌هایی کسب کرده‌اند و نتایج حاصل می‌تواند چراغ راهی برای انتخاب یک متدولوژی مناسب برای پروژه‌های تیم‌های دیگر باشد. برای اولین‌بار، صنعت ما یک راه پایدار و واقعی برای حل مشکلاتی پیدا کرده که نسل‌های متوالی از تیم‌های توسعه نرم‌افزار با آن سروکله زده‌اند. پروژه‌های چابک به‌موقع تمام می‌شوند که این برای تیم‌ها مسئله بزرگ و مهمی است. مخصوصاً وقتی با تحویل دیرهنگام و دورتر از بودجه درگیر هستند. پروژه‌های چابک یک نرم‌افزار با کیفیت بالا تحویل می‌دهند. این یک تغییر بزرگ برای تیم‌هایی است که نرم‌افزارهای پر از باگ و ناکارآمد تحویل می‌دهند. کدی که توسط یک تیم چابک ساخته می‌شود به شکل مؤثری خوش‌ساخت و نگهداشت‌پذیر است. این یک روش نویدبخش برای تیم‌هایی است که کدهایی با پیچیدگی کدهای اسپاگتی درست می‌کنند. تیم‌های چابک، کاربر نهایی را راضی می‌کنند که این تغییر بزرگی نسبت به نرم‌افزارهایی است که توانایی انتقال ارزش‌ها به کاربران را ندارند. پس از آشنایی با این روش‌ها در فصل آینده روش‌های ترکیبی را بررسی خواهیم کرد. جایی که روش‌های سنتی فصل دو و شیوه‌های نوین این فصل درهم‌آمیخته می‌شوند تا نیازهای صنایع و سازمان‌های بزرگ را بهتر برطرف کند.

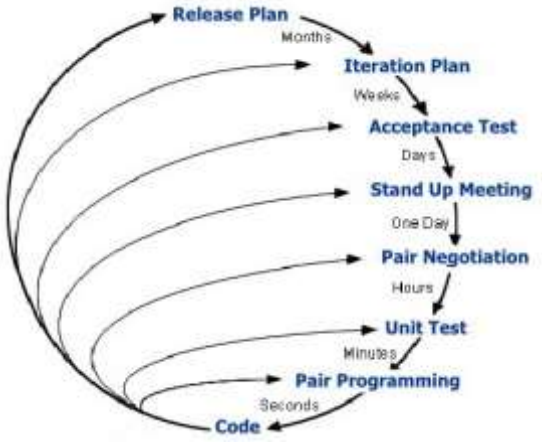
۳- ۱ برنامه‌نویسی مفرط (حدی)

برنامه‌نویسی مفرط[1] حاصل تلاش‌های کنت بک[2] در اواخر دهه ۱۹۸۰ در شرکت کرایسلر است. در کل به‌عنوان یک رویکرد مهندسی نرم‌افزار شناخته نمی‌شود و بیشتر به‌عنوان یک رویکرد چابک مطرح می‌شود. این رویکرد از توسعه هر چه سریع‌تر نرم‌افزار به‌خصوص در سازمان‌ها و برنامه‌های کاربردی کوچک و متوسط پشتیبانی کرده و بیشترین بازدهی را زمانی فراهم می‌آورد که پروژه به سه تا ده برنامه‌نویس نیاز داشته باشد. گاهی به‌عنوان یک متدولوژی ساده در نظر گرفته می‌شود و گاهی به‌جای یک متدولوژی، به‌عنوان مجموعه‌ای از اصول برای توسعه سریع برنامه کاربردی شناخته می‌شود. بر ارزش‌هایی چون سادگی، ارتباطات، بازخورد و شجاعت تأکید می‌کند. در حوزه ارتباطات بر کار تیمی و جلسات غیررسمی روزانه تأکید می‌کند که با عنوان «جلسه ایستاده» نام‌برده می‌شود. بر انجام آزمایش‌های مداوم از اولین روز و برقراری ارتباط با مشتریان توجه ویژه دارد. در این رویکرد به اعمال تغییرات موردنظر مشتری حتی در مراحل پایانی چرخه حیات نیز توجه شده و آن را لازم می‌داند.

برنامه‌نویسی دونفره یا زوجی[3] به طور مشخص برای کاهش ریسک و افزایش کیفیت برنامه کاربردی در این رویکرد، یک مسئله مهم و حیاتی است. در این روش دو برنامه‌نویس در کنار هم و روی یک ایستگاه کاری کار می‌کنند. در هر لحظه یکی از این دو کدنویسی کرده و دیگری کد او را بررسی و نقد می‌کند و به فراخور نیاز راهنمایی‌اش می‌نماید. این دو به‌صورت دوره‌ای جای خود را عوض کرده و کسی که قبلاً نقش نقاد داشته، دست به کدنویسی برده و کدنویس مرحله قبل کد او را نقد و بررسی می‌کند. نفر

[1] Extreme programming
[2] Kent Beck
[3] Pair programming

دوم که کار کدنویسی را مشاهده می‌کند، مشاهده‌گر می‌نامند. به این ترتیب از پیاده‌سازی‌های غلط و راه‌حل‌های مشکل‌دار جلوگیری شده و تبادل دانش بین دو نفرِ برنامه‌نویس هم به خوبی صورت می‌پذیرد.



تصویر ۳-۱ : چرخه برنامه‌ریزی و بازخورد در برنامه‌نویسی مفرط (Beck, 1999).

برای استفاده از این رویکرد در سازمان‌های بزرگ شکل دیگری از XP با نام IXP یا همان XP صنعتی در سال ۲۰۰۴ پیشنهاد شده است. از تفاوت‌های عمده آن می‌توان به اعمال مدیریت بیشتر، گسترش نقش مشتریان و ارتقا روش‌های فنی اشاره کرد. در این رویکرد مشتری باید نیازمندی‌های خود را در قالب داستان‌های کاربر[1] مشخص نماید. این کار تااندازه‌ای مشابه Use Case های استفاده شده در روش‌های سنتی است اما به شکل موثری بر سادگی و کوتاهی و تمرکز بر نیاز اصلی کاربر تاکید دارد. این داستان‌ها جایگزین سند نیازمندی‌ها می‌شود. روشی که در برنامه‌سازی مفرط برای جمع‌آوری نیازمندی‌ها وجود دارد شنیدن نام دارد. فعالیت برنامه‌ریزی در برنامه‌سازی مفرط نیز با شنیدن آغاز می‌شود. طی این فعالیت شنیدن، چندین داستان کاربری ایجاد می‌شود. برای پی بردن به راه‌حل‌های بالقوه گاهی از راه‌حل ضربتی یا Spike استفاده می‌شود. در مواردی که داستان‌های کاربری حساس و مهم باشند، از راه‌حل‌های ضربتی استفاده می‌شود. راه‌حل ضربتی برنامه ساده‌ای است که به‌وسیله آن می‌توان راه‌حل‌های بالقوه را کشف کرد. طراحی در برنامه‌سازی مفرط از اصل سادگی[2] تبعیت می‌کند و یک طراحی ساده همواره بر یک طراحی پیچیده اولویت دارد. این اصل بیان می‌کند که اکثر سیستم‌ها چنانچه ساده و به‌دور از پیچیدگی بمانند، عملکرد بهتری خواهند داشت. بنابراین، سادگی باید هدف اصلی طراحی سیستم‌ها باشد و از پیچیدگی‌های بیهوده اجتناب کرد. پس از نوشته شدن داستان‌های کاربری و طراحی اولیه، برنامه‌نویسی آغاز نمی‌شود، بلکه آزمون‌های واحدی[3] برای نرم‌افزار نوشته می‌شود که با توجه به داستان‌های کاربری و طراحی ساخته شده‌اند. پس از این مرحله برنامه‌نویسی با تمرکز بر کد مورد نیاز برای گذراندن آزمون‌های واحد نوشته می‌شود و هیچ چیز دیگری به کد اضافه نمی‌گردد. برنامه‌نویسی دونفره در همین مرحله انجام می‌شود. این روش باعث کم شدن خطاهای برنامه‌نویسی تا ۱۵ درصد نسبت به روش‌های متداول برنامه‌نویسی می‌شود و در عین حال برنامه‌ها با این روش به طور میانگین تا ۲۰ درصد کوتاه‌تر می‌شوند. آزمون‌های نرم‌افزار باید قبل از شروع پیاده‌سازی توسط چارچوبی که از آزمون

[1] User Stories
[2] KISS principle
[3] Unit Test

خودکار پشتیبانی می‌کند نوشته شوند. کنت بک از پایه‌گذاران این متدولوژی، بیان می‌کند که برنامه‌سازی مفرط نیازمند شجاعت و نظم است. پارادایم برنامه‌نویسی متداول در این رویکرد همان شیءگرایی است. بازآرایی کد[1] و توسعه آزمون محور[2] هم جز فعالیت‌های اساسی در این رویکرد وجود دارند. امروزه همچنان در تمام تیم‌های توسعه از مفاهیم ایکس‌پی به خصوص در آزمون و کدنویسی استفاده می‌شود، اما مدیریت پروژه نرم‌افزاری با روش‌هایی چون کانبان و اسکرام بیشتر صورت می‌پذیرد.

۳- ۲ روش توسعه پویای سیستم‌ها

روش توسعه پویای سیستم‌ها[3] یا DSDM، پس از مشخص‌شدن مشکلات RAD (توسعه سریع برنامه کاربردی[4]) آغاز شد. در سال ۱۹۹۴ گروهی از توسعه‌دهندگان سیستم‌هایی که با RAD کار می‌کردند گرد هم آمدند تا بی‌نظمی این روش را بهبود ببخشند. توسعه سیستم‌های پویا یک روش تکرارشونده و افزایشی است که قوانین موجود در روش چابک، یعنی توسعه پروژه با دخالت مشتری را در بر می‌گیرید. اولین تعریف از این روش در سال ۱۹۹۵ منتشر شد. پس از نظارت و بازبینی عملی، نسخه بعدی در همان سال و نسخه سوم آن در سال ۱۹۹۷ منتشر شد. این روش به‌خصوص در انگلستان و تا حد کمتری در اروپا شناخته‌شده‌تر و کاربردی است. در سال ۲۰۰۰، نسخه‌ای از این روش باتوجه‌به کسب‌وکارهای الکترونیکی منتشر شد و با نام eDSDM شناخته شد. این چارچوب دائماً در حال تکامل بوده و نسخه ۴ آن در سال ۲۰۰۱ منتشر شد. آخرین نسخه آن، نسخه ۴,۲ است که دربرگیرنده خط‌مشی برای آن دسته از افراد و سازمان‌هایی است که به دنبال استفاده از XP در راستای این روش هستند. اختلاف نظرهایی در مورد متدولوژی یا چارچوب بودن این روش وجود دارد (مولاناپور و همکاران، ۱۳۹۱). بااین‌حال به نظر بیشتر به‌عنوان یک چارچوب عمل می‌کند. نه اصل این رویکرد به شرح ذیل بیان می‌شوند:

- حضور و مشارکت فعالانه کاربر ضروری است.
- تیم‌ها باید بر اساس چهار متغیر توانمندسازی برای تصمیم‌گیری تقویت شوند.
- توزیع مکرر و پیوسته محصولات امری ضروری است.
- اقلام تحویل داده شده باید برای هدف آن کسب‌وکار مناسب باشند.
- توسعه مکرر و افزایشی یک امر ضروری است.
- اگر در حین توسعه با مشکلی مواجه شدید به آخرین نقطه امن قبلی برگردید.
- نیازمندی‌ها در سطوح بالایی تعریف شده‌اند.
- انجام آزمایش مداوم در حین کار به‌جای آزمایش در انتهای کار توصیه می‌گردد.
- یک رویکرد مشارکتی بین تمامی ذی‌نفعان لازم است.

در این روش پنج فاز اصلی در توسعه چرخه حیات وجود دارد. به مجموع این فازها و محصولات اصلی هر فاز گاهی نمودار «سه پیتزا و یک پنیر» نیز گفته شده است. این فازها به ترتیب: مطالعه امکان‌سنجی، مطالعه کسب‌وکار، تکرار مدل عملکردی، طراحی

[1] Refactoring
[2] TDD
[3] Dynamic systems development method
[4] Rapid application development

سیستم و ساخت تکرار و پیاده‌سازی هستند. این روش روی نقش کلیدی افراد در فرایند تأکید می‌کند و بر همین اساس از آن تحت عنوان «کاربرمحور» نام می‌برد. در طرف کاربران، دو نوع کاربر کلیدی وجود دارد. کاربر مأمور یا فرستاده که فردی از جامعه کاربران است که نیازهای کاربران را درک می‌کند و آن‌ها را ارائه می‌کند. کاربر نوع دوم، کاربر دوراندیش است. مسئولیت این کاربر توجه به این نکته است که به چشم‌اندازها در خلال پروژه توجه شود و انحراف‌های مسیر را گوشزد نماید. این روش تیم‌های توسعه کوچک شامل کاربران و تیم توسعه را پیشنهاد می‌کند. توسعه سیستم‌های پویا هزینه، کیفیت و زمان را با استفاده از اولویت‌بندی روش ماسکو[1] به طرز مناسبی ثابت نگاه می‌دارد تا در محدودیت زمانی اعلام شده از سوی مشتری پروژه در زمان مناسبی تحویل گردد. روش ماسکو تکنیکی است که برای رسیدن به یک فهم مشترک از اهمیت نیازمندی‌ها برای ذی‌نفعان پروژه در تحلیل کسب‌وکار، مدیریت پروژه و ایجاد نرم‌افزار به کار می‌رود. همچنین با نام اولویت‌بندی ماسکو یا تحلیل ماسکو نیز شناخته می‌شود.

۳-۳ اسکرام

اسکرام در حقیقت روشی برای شروع مجدد بازی در راگبی است. وقتی خطایی در بازی رخ می‌دهد و یا توپ از زمین خارج می‌شود از اسکرام برای شروع دوباره استفاده می‌شود. این روش در سال ۱۹۸۶ توسط هیروتاکا تاکوچی[2] و ایکوجیرو نوناکا[3] به‌عنوان یک خط‌مشی جدید برای تولید نرم‌افزارهای تجاری که باید قابلیت سرعت در تولید و انعطاف‌پذیری را داشته باشند، عرضه گردید. چارچوب یا فرایند مدل اسکرام یک چارچوب تکرارپذیر و افزایشی برای کنترل پروژه (مدیریت نرم‌افزار) است. کن شوئبر[4] و جف سادرلند[5] در سال ۱۹۹۵ اسکرام را به‌عنوان یک چارچوب توسعه نرم‌افزاری معرفی کردند. اکنون مشخص شده که ترکیب تفکر چابک و چارچوب اسکرام نسبت به روش فرماندهی و کنترل پروژه‌های سنتی به شکل مؤثرتری کاراست. البته این مورد به این معنا نیست که روش سنتی مبتنی بر برنامه‌ریزی، دیگر در هیچ شرایطی کاربرد ندارد. منظور این است که امکان برگشت به چند مرحله قبل و یادگرفتن و بهبود فرایند و تکنیک‌ها با ایده‌ها و مفاهیم جدید چیز خوبی است. اسکرام سبک‌وزن و ساده است اما تسلط بر آن در عمل دشوار است. اسکرام درجه سودمندی مدیریت محصول و روش‌های کاری را واضح‌تر می‌کند، بنابراین می‌توان محصول، تیم و محیط کاری را به طور مستمر بهبود بخشید. چارچوب اسکرام از تیم‌های اسکرام به همراه نقش‌ها، رویدادها، مصنوعات و قوانین مرتبط با آن‌ها تشکیل شده است. اسکرام چهار رویداد رسمی را برای بازرسی و سازگاری تجویز کرده است:

- برنامه‌ریزی اسپرینت[6]
- اسکرام روزانه
- بازبینی اسپرینت[7]
- بازاندیشی اسپرینت[8]

[1] MoSCoW method
[2] Hirotaka Takeuchi
[3] Ikujiro Nonaka
[4] Ken Schwaber
[5] Jeff Sutherland
[6] Sprint planning
[7] Sprint review
[8] Sprint retrospective

تیم اسکرام از یک مالک محصول،[1] تیم توسعه و یک اسکرام مستر[2] تشکیل شده است. تیم‌های اسکرام خودسازمانده و فراوظیفه‌ای هستند. مالک محصول، مسئول به حداکثر رساندن ارزش محصولی است که از کار تیم توسعه حاصل می‌شود. مالک محصول تنها فرد مسئول برای مدیریت بک‌لاگ محصول[3] است. اسکرام‌مستر مسئولیت ترویج و حمایت از اسکرام به‌گونه‌ای که در راهنمای اسکرام تعریف شده است را بر عهده دارد. اسپرینت[4] قلب اسکرام است، یک بازه‌ی زمان ثابت یکماهه یا کمتر که طی آن، یک فرآورده تکمیل‌شده و قابل استفاده تکمیل می‌شود (بی‌بالان و همکاران، ۱۳۹۷).

تصویر ۳-۲ : فرایند اسکرام

اصطلاح بک‌لاگ محصول، نامی است که به مخزن نیازمندهای عملیاتی و غیرعملیاتی کل یک پروژه اطلاق می‌شود و در حقیقت مجموعه‌ای اولویت‌بندی شده از نیازمندی‌های سطح بالای سیستمی است که در نهایت بایستی تحویل داده شود. مواردی از بک‌لاگ محصول که در طی یک اسپرینت بایستی انجام شود در طول جلسه طراحی اسپرینت مشخص می‌شود. در طول این جلسه، مالک محصول اعضاء تیم را دربارهٔ مواردی از بک‌لاگ محصول آگاه می‌کند. سپس اعضاء تیم مشخص می‌کنند که چه مقدار از موارد مشخص شده را می‌توانند در این اسپرینت انجام دهند و چه میزان از آن را در اسپرینت‌های بعدی تکمیل خواهند کرد. مواردی از بک‌لاگ محصول که قرار است در یک اسپرینت انجام شود را اصطلاحاً بک‌لاگ اسپرینت می‌نامند. وقتی ارزش‌هایی مانند تعهد، شجاعت، تمرکز، باز بودن و احترام، توسط تیم اسکرام مجسم و محقق شود، ارکان شفافیت، بازرسی و سازگاری اسکرام، پدیدار شده و موجب ایجاد اعتماد برای همه خواهد شد. اسکرام یک چارچوب چابک است که با آن می‌توانیم مسئله‌های پیچیده را حل نماییم. مسئله پیچیده به مواردی گفته می‌شود که دانش ما نسبت به مسئله ناقص است و به‌مرور این دانش پدیدار خواهد شد (Zayat, et. al., 2020).

۳-۴ کانبان

برای آشنایی با مفاهیم کانبان[5] لازم است نگاهی به تاریخچه طرز تفکر تولید ناب[6] بیندازیم. تولید ناب از روش خط تولید شرکت تویوتا[7] در حدود سال ۱۹۳۰ مشتق شده است. فرایند ساخت‌وساز ناب و مدل‌سازی دیجیتال، صنایع تولیدی و هوافضا را متحول کرده‌اند. اولین به‌کارگیری این ابزارها و فرایندهای تولیدی، نظیر تویوتا، بوئینگ، به بهره‌وری در تولید و موفقیت‌های تجاری دست

[1] Product Owner
[2] Scrum Master
[3] Product Backlog
[4] Sprint
[5] Kanban
[6] Lean Production
[7] Toyota Production System (TPS)

یافته‌اند. می‌توان حوزه‌های اصلی ساخت ناب را به چهار حوزه جریان فرایندها، فرایند تولید ارزش، حل مسئله و توسعه ذی‌نفعان تقسیم کرد. کانبان یک سیستم زمان‌بندی برای رویکردهای تولید ناب و ساخت هم‌زمان است. سیستم کانبان توسط تای‌ایچی اونو[1] یک مهندس صنایع ژاپنی شاغل در شرکت خودروسازی تویوتا، با هدف بهبود کارایی در تولید، توسعه داده شد. کانبان یکی از روش‌های دستیابی به نگرش تولید به‌موقع[2] است. این سیستم در واقع نام خود را از کارت‌هایی گرفته است که در کارخانه‌ها تویوتا، برای رهگیری فرایند تولید مورداستفاده قرار می‌گیرند. امروزه سیستم کانبان به‌عنوان ابزاری برای پشتیبانی در زمان اجرای سیستم‌های تولید، تبدیل شده است. کلمه کانبان یک واژه ژاپنی به معنی تخته اعلان است. آیتم‌ها یا نمونه‌های کار به‌صورت تصویری ارائه می‌شوند تا به شرکت‌کنندگان نمایی از پیشرفت و روند کار از ابتدا تا پایان ارائه دهند، معمولاً این نما از طریق تابلوی کانبان نمایش داده می‌شود. تفکر ناب توسط تام و مری پاپندیک[3] وارد حوزه مهندسی نرم‌افزار شد. اما کانبان و ورود آن به دنیای نرم‌افزار نتیجه کارهای دیوید اندرسون[4] می‌باشد که نتیجه تجربیات او و همکارانش در مایکروسافت را در فاصله سال‌های ۲۰۰۴ تا ۲۰۱۰ در برمی‌گیرد.

اصول کانبان به شرح ذیل می‌باشد:

- کارها را به شکل تصویری مجسم کنید.
- کارهای در حال انجام را محدود کنید.
- جریان کار را مدیریت و ارتقا ببخشید.
- سیاست‌ها را به‌صراحت بیان کنید.
- حلقه‌های بازخورد دهنده را اجرا کنید.
- به طور مشارکتی پیشرفت کنید، به‌صورت تجربی تکامل پیدا کنید.

کانبان پیشرفت کار را مستقیماً روی تابلو کانبان مدیریت می‌کند. کارهای در حال انجام،[5] از مشکلات رایج در پیشرفت کار بازخورد فوری به تیم توسعه ارائه می‌دهد. به شکلی می‌توان کانبان را روح مکتب مدیریت در ژاپن نامید و اسکرام با جزئیات فراوان را حاصل مدیریت آمریکایی در نظر گرفت. بااین‌حال می‌دانیم که اسکرام در حدود سال ۱۹۹۵ توسط کن شوئبر و جف سادرلند به‌عنوان یک متدولوژی رسمی برای تولید نرم‌افزار ایجاد شد اما ریشه در کارهای هیروتاکا تاکوچی و ایکوجیرو نوناکا در سال ۱۹۸۶ دارد.

[1] Taiichi Ohno
[2] Just-in-time manufacturing (JIT)
[3] Tom and Mary Poppendieck
[4] David Anderson
[5] Work in Process (WIP)

تصویر ۳-۳ : تابلوی کانبان

به‌این‌ترتیب با یک نگاه به تابلوی کانبان می‌توان به موارد ذیل به‌راحتی پی برد:

- کارهای در حال انجام و متوقف شده کدام‌ها هستند.
- هر فرد در تیم مشغول انجام چه فعالیت‌هایی است.
- انواع کار و حتی پروژه‌های موجود مشخص می‌شود و درصد آن‌ها از کل کار قابل تشخیص خواهد بود.

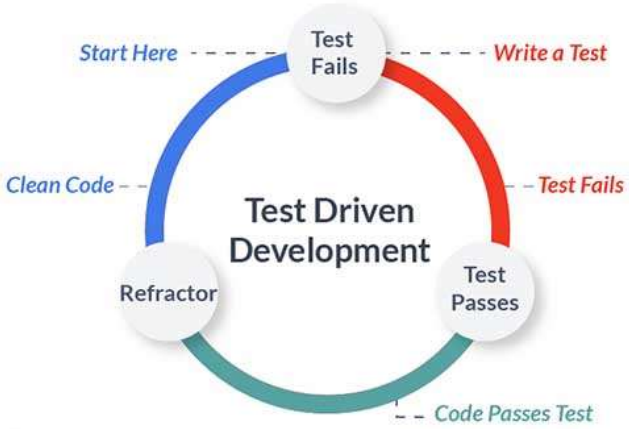
آقای هنریک نیبرگ[1] معتقد است کانبان بهتر از اسکرام نیست، بلکه تنها از آن کوچک‌تر است. به‌راحتی به کمک تصویرسازی می‌توان از وضعیت پروژه مطلع شد و گلوگاه‌ها را تشخیص داد و اگر کارها از حالت عادی خارج شده‌اند شاید لازم باشد تنها چند برگه را ازروی تابلو جابه‌جا کنیم. به این ترتیب یک حلقه تاثیرگذاری متقابل شکل می‌گیرد. بیان صریح سیاست‌ها کمک می‌کند تا برخی از جنبه‌های سیستم که با زبان تصویری رنگ‌ها، ستون‌ها و برچسب‌ها قابل بیان نیست به خوبی بیان شوند. این سیاست‌ها دستوراتی از بالا و رسمی نیستند بلکه تنها کمک می‌کند تا افرادی که در توسعه سیستم مشارکت دارند به درک مشترکی از چگونگی عملکرد سیستم و نحوه تعاملات دست یابند. نظر به اهمیت حلقه‌های بهبود در توسعه نرم‌افزار، کانبان نیز از این اصل مستثنی نیست و «چرخه دمینگ[2]» را بکار می‌برد. این چرخه با نام پی‌دی‌سی‌ای[3] نیز شناخته می‌شود که به مراحل آن اشاره دارد:

- مرحله اول: برنامه‌ریزی که باید فرایند و طرحی برای بهبود و رفع مشکل فراهم شود.
- مرحله دوم: انجام که طرح خود را به‌صورت آزمایشی اجرا می‌کنیم.
- مرحله سوم: بررسی، اقدامات انجام شده را بررسی و نتایج را با چیزی که در طرح و برنامه بوده مطابقت می‌دهیم.
- مرحله چهارم: اقدام، با بررسی دست به انتخاب از بین روش‌های موجود خواهیم زد.

[1] Henrik Kniberg
[2] Deming circle
[3] PDCA: plan–do–check–act

۳- ۵ توسعه آزمون محور

توسعه آزمون محور نیز توسط کنت بک در سال ۲۰۰۳ معرفی شد. در روش توسعه آزمون محور هر ویژگی جدید با افزودن یک تست شروع می‌شود. این تست باید طوری طراحی شود که تنها با افزوده شدن صحیح و کامل ویژگی جدید مطلوب پاس شود. پس در ابتدا یک تست جدید می‌نویسیم. در مرحله بعد باید مطمئن شویم با اجرای تست حتماً خطا دریافت می‌کنیم و اصطلاحاً تست پاس نشود. در گام بعدی تنها کد اضافه‌ای که نیاز هست تا تست جدید پاس شود را می‌نویسیم و به زیاده‌روی در توسعه نمی‌پردازیم. سپس تمام تست‌ها را دوباره اجرا می‌کنیم و از پاس شدن تست جدید و سایر تست‌های قبلی اطمینان حاصل می‌کنیم. یک گام بازآرایی و بازسازی کد نیز وجود دارد. در این گام به طور ویژه لازم است به ملاحظات توسعه همانند رعایت وراثت و خوانایی و امکان نگهداری مناسب کد و الگوهای طراحی برنامه توجه داشته باشیم. برای افزودن هر ویژگی جدید گام‌های قبل را تکرار می‌کنیم.



تصویر ۳-۴ : چرخه توسعه آزمون محور

به‌این‌ترتیب مزایای این متدولوژی را می‌توان به شرح ذیل موردتوجه قرارداد:

- بررسی و توجه به جنبه‌های مختلف توسعه برنامه از همان ابتدا.
- کاهش هزینه‌های توسعه برنامه به کمک این روش قابل‌توجه است.
- با بررسی نتایج آزمون هر ویژگی مستندسازی بهتری انجام می‌پذیرد.
- اقلام تحویل داده شده باید برای هدف آن کسب‌وکار مناسب باشند.
- بازسازی کد به کدهای مرتب‌تر، خواناتر و با نگهداشت‌پذیری بالاتری منجر می‌شود.

۳-۶ توسعه ویژگی محور

توسعه ویژگی محور[1] اولین‌بار در سال ۱۹۹۹ توسط پیتر کود، اریک لوفور و جف دلوکا معرفی شد. سپس در سال ۲۰۰۲ توسط استیفن پالمر و مک فلشینگ بهبودهایی در آن ایجاد شد و به‌عنوان فرایندی چابک و انطباق‌پذیر برای پروژه‌های متوسط و بزرگ معرفی گردید. فلسفه اقتباسی این روش بر همکاری میان اعضاء تیم تأکید دارد. ویژگی[2] در اینجا یک عملکرد است که برای متقاضی دارای ارزش بوده و در کمتر از دوهفته قابل پیاده‌سازی است. در اینجا بیش از هر روش چابک دیگری بر تکنیک‌ها و دستورالعمل‌های مدیریت پروژه تأکید می‌شود.

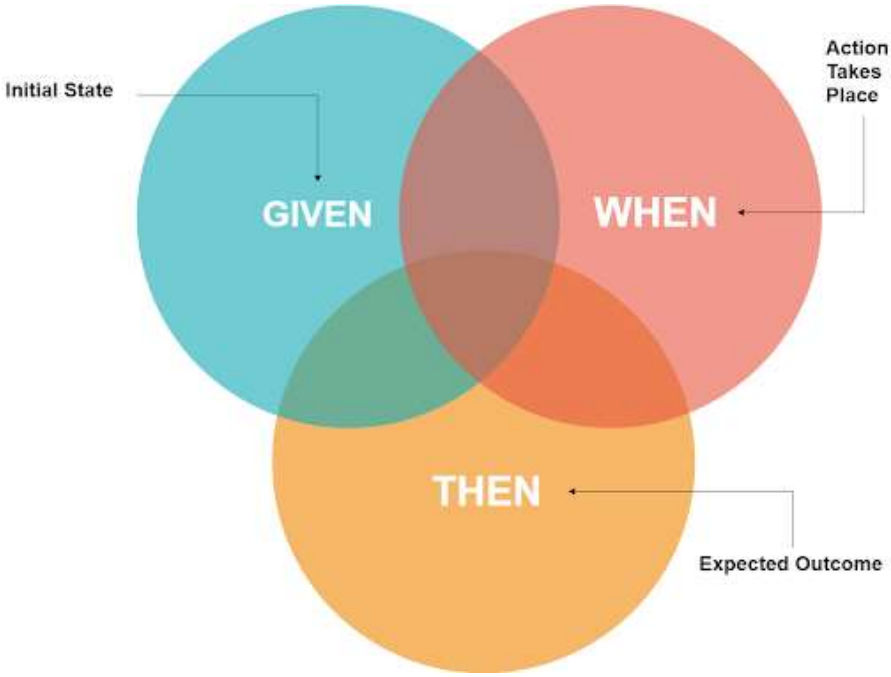
تصویر ۳-۵ : توسعه ویژگی محور

۳- ۷ توسعه رفتار محور

این رویکرد در سال ۲۰۰۳ در ارتباط با TDD توسط دن نورس[3] ایجاد شد. سپس توسعه پیدا کرد و به‌عنوان یک رویکرد مجزا تحت عنوان رفتار محور یا BDD در سال ۲۰۰۹ معرفی شد. در این روش دو تیم توسعه و کسب‌وکار در کنار هم قرار می‌گیرند. این کار به‌منظور دستیابی سریع‌تر به اهداف پروژه صورت می‌گیرد. در اینجا هدف شفاف‌سازی رفتارهای نرم‌افزار برای تیم توسعه است. بزرگ‌ترین تفاوت توسعه رفتار محور نسبت به روش آزمون محور، در نوشتن آزمون به زبان طبیعی است، به‌طوری‌که افراد غیر برنامه‌نویس بتوانند آنها را بخوانند. به‌جای توسعه توابع و عملیات، با استفاده از جملات خواهیم گفت که کد و برنامه ما چه کاری را دقیقاً انجام خواهد داد. توسعه رفتار محور برای به‌وجودآوردن همکاری مشترک مشتری و توسعه‌دهنده در تحلیل نیازمندی‌ها است. توسعه رفتار محور تکنیک‌ها و قواعد اساسی، توسعه آزمون محور را با ایده‌های طراحی دامنه محور و تجزیه‌وتحلیل و طراحی شیءگرا ادغام می‌کند تا برای تیم‌های توسعه و مدیریت نرم‌افزار، ابزارها و فرایندهای مشترکی برای همکاری در توسعه نرم‌افزار ارائه کند. توسعه رفتار محور تا حد زیادی با استفاده از یک زبان مختص دامنه[4] از طریق بهره‌گیری از

[1] Feature-driven development
[2] Feature
[3] Dan North
[4] Domain-specific language

ساختارهای زبان طبیعی که می‌تواند رفتار و نتایج مورد انتظار را بیان کند، میسر می‌شود. توسعه رفتار محور هیچ الزام رسمی برای اینکه دقیقاً چگونه این داستان‌های کاربر باید نوشته شود ندارد، اما تأکید دارد که هر تیم، یک فرمت ساده و استاندارد برای نوشتن داستان‌های کاربر طراحی کند.



تصویر ۳-۶ : توسعه رفتار محور

روش رفتار محور، مشخص می‌کند که تحلیلگران کسب‌وکار و توسعه‌دهندگان باید در این زمینه همکاری داشته باشند و رفتارها را در قالب عبارت‌های داستان‌های کاربری مشخص کنند که هرکدام صریحاً در یک سند اختصاصی نوشته می‌شوند. به‌عنوان‌مثال Gherkin یک زبان خاص دامنه است که به توصیف رفتار سیستم بدون توجه به پیاده‌سازی می‌پردازد. این زبان قابلیت‌های موردنیاز یک سیستم را با استفاده از ویژگی‌ها و سناریوهای مرتبط به ویژگی‌ها، تعریف می‌کند. یکی از چارچوب‌های رایج در روش رفتار محور، Cucumber نام دارد که امروزه به شکل گسترده‌ای در این رویکرد بکار می‌رود. از مزایای این روش می‌توان به شفافیت، همکاری و تعامل بیشتر، طراحی نرم‌افزار با پیروی از ارزش‌های کسب‌وکار، هزینه کمتر و اطمینان بیشتر اشاره کرد.

۳- ۸ توسعه عملیات یا دواپس

توسعه عملیات یا DevOps مخفف «توسعه نرم‌افزار» و «عملیات فناوری اطلاعات» است. دواپس، مجموعه‌ای از روش‌ها، فرایندها و ابزارهایی است که با تمرکز بر ارتباطات، همکاری و یکپارچگی بین تیم‌های توسعه نرم‌افزار و عملیات فناوری اطلاعات، ارزش‌های تولید شده را به طور سریع و مداوم به مشتریان نهایی می‌رساند. این اصطلاح در پی مجموعه رویدادهایی در سال ۲۰۰۹ که در

بلژیک توسط پاتریک دبوآ[1] برگزار شد کم‌کم بر سر زبان‌ها افتاد. دبوآ در جریان توسعه یک سیستم دولتی متوجه شد، سیستم ادمین‌ها و توسعه‌دهندگان در تعامل با هم به مشکلاتی برخورده‌اند که تأثیر منفی در روند اجرای کار داشته است. به دلیل اینکه دواپس یک تغییر فرهنگی است و همکاری‌ای بین توسعه، عملیات و تست می‌باشد، یک مجموعه ابزار واحد برای آن وجود ندارد.

تصویر ۳-۷ : دواپس

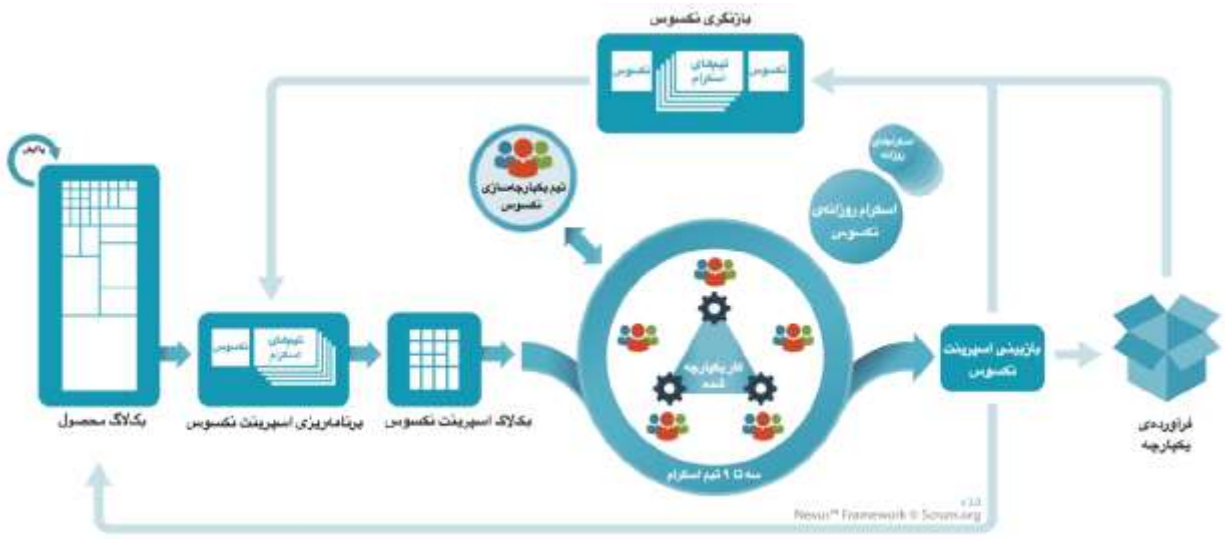
می‌توان دواپس را به‌عنوان تقاطع توسعه، عملیات و تضمین کیفیت در نظر گرفت. تکنولوژی‌های جدید علاوه بر توسعهٔ عملکرد نرم‌افزارها، بر روی فرهنگ سازمانی نیز تأثیرگذار هستند. تغییراتی که دوآپس ایجاد می‌کند ترکیبی از هر دو تغییر فرهنگی و فناوری‌های جدید است. ایدهٔ دواپس اصلاح فضای کار برای همکاری، بحث و به اشتراک گذاشتن تجربیات است. توجه به این روش تا حدودی به ابری شدن بسیاری از زیرساخت‌ها نیز برمی‌گردد. اینجا بود که همه متوجه نگرانیِ پاتریک شدند، ارتباط ضعیف بین تیم‌های تضمین کیفیت، عملیات، توسعه و برنامه‌نویسی، باعث می‌شد فرایند تولید محصول کند پیش برود. هر زمان مشکلی ایجاد می‌شد این تیم‌ها یکدیگر را سرزنش و محکوم می‌کردند. در حقیقت تعاملی که میان تیم‌ها برقرار می‌شود و البته خودکارسازی بسیاری از روال‌های تکراری، منجر به تسریع چرخهٔ تولید محصول و تحویل به مشتری می‌شود. در حال حاضر نیز دوآپس به‌عنوان بزرگ‌ترین قدم بعد از چابک در صنعت IT شناخته می‌شود. به‌کارگیری دوآپس منجر به افزایش اعتماد بین کارکنان، رفع فوری مشکلات، انتشار سریع‌تر نسخه‌های نرم‌افزار، مدیریت بهتر کارها و در نهایت رضایت مشتری می‌شود. در دواپس برخلاف چابک که به بهبود ارتباط تیم توسعه و مشتری می‌پردازد، به بهبود روابط بین تیم توسعه و عملیات توجه بیشتری می‌شود.

۳- ۹ نکسوس

وقتی یک تیم بر روی پروژه‌ای کار می‌کند، همان اسکرام گزینه مناسبی است. اما تصور کنید چند تیم قرار است به‌صورت هم‌زمان بر روی یک پروژه کار کنند، در این موارد راهکار چیست؟ نفراتی که اسکرام را توسعه داده‌اند، به این موضوع نیز فکر کرده‌اند و بر

[1] Patrick Debois

اساس نیازهایی که برای هماهنگی چند تیم وجود داشته، نکسوس[1] را توسعه داده‌اند. عمده راهنمایی‌های چارچوب نکسوس توسط کن شوئبر تدوین شده است. قطعات سازنده چارچوب جدید همان اجزاء اسکرام است که مواردی به آن افزوده شده است. نکسوس یک چارچوب برای توسعه و نگهداری محصولات مقیاس‌پذیر و طرح‌های توسعه نرم‌افزار است. نکسوس یک چارچوب شامل نقش‌ها، رویدادها، مصنوعات و تکنیک‌هایی است که تلاش تقریباً ۳ الی ۹ تیم مستقلِ اسکرام که همگی بر روی یک بک‌لاگ محصول مشترک، جهت ساختن محصولات یکپارچه با هدفی واحد، کار می‌کنند را به هم متصل می‌کند. زمانی که چند تیم برای ساختن یک محصول با همکار می‌کنند، دستکم در طول یک اسپرینت، وابستگی‌های زیادی رخ می‌دهد نکسوس همانند یک اسکلت نگهدارنده بر روی تیم‌های اسکرامی که برای تولید یک محصول واحد با هم ترکیب شده‌اند قرار می‌گیرد. تفاوت در این است که نکسوس به وابستگی‌ها و ارتباط بین تیم‌های اسکرام و ارائه حداقل یک محصول یکپارچه در هر اسپرینت توجه بیشتری دارد.



تصویر ۳-۸ : نکسوس

یکی از نقش‌هایی که در این چارچوب اضافه شده «تیم یکپارچه‌سازی نکسوس» است. این نقش اضافه شده تا به‌منظور حصول بهترین نتایج، بر نحوه استفاده از نکسوس و اجرای اسکرام، رهبری و نظارت داشته باشد. این تیم شامل یک مالک محصول، یک اسکرام مستر و اعضاء تیم یکپارچه‌سازی است.

۳- ۱۰ اسکرام‌بان

اسکرام‌بان دو رویکرد چابک، اسکرام و کانبان را ترکیب می‌کند تا یک چارچوب مدیریت برای بهبود مهندسی نرم‌افزار ایجاد کند. اسکرام‌بان بهترین ویژگی‌های هرکدام از این دو رویکرد را به ارث می‌برد. اگرچه مطالعات کمی در مورد نحوه اثربخشی این متدولوژی ترکیبی چابک انجام شده است. موفقیت پروژه‌های توسعه نرم‌افزار عمیقاً به استفاده درست از یک متدولوژی توسعه مناسب وابسته است. بر اساس گزارش Standish Group حدود ۴۲ درصد پروژه‌هایی که از رویکرد چابک استفاده کرده‌اند به موفقیت رسیده‌اند که این میزان بیشتر از درصد موفقیت پروژه‌هایی است که از روش‌های توسعه سنتی نرم‌افزار بهره برده‌اند.

[1] Nexus

روش‌های چابک تکراری، افزایشی هستند و رویکرد مشارکتی را بین تیم‌های خودسازمانده و فرا وظیفه‌ای افزایش می‌دهند. امروزه اسکرام متداول‌ترین روش چابک است که در توسعه نرم‌افزار به کار می‌رود. اسکرام از طریق تکرارهای جعبه - زمانی، بازخورد مستمر و اولویت‌بندی وظایف به هدف‌هایش دست می‌یابد. در سوی دیگر، کانبان، به این گستردگی در توسعه نرم‌افزار بکار گرفته نشده بود. در سال ۲۰۰۴، کانبان به کوشش دیوید اندرسون، وقتی در مایکروسافت با یک تیم کار می‌کرد، وارد قلمرو توسعه نرم‌افزار شد. انجام کارها با متدولوژی چابک، به چیزی بیشتر از تغییر در فرایند احتیاج دارد. چنین تحولی مستلزم تغییر در تفکر نیز هست. برای اینکه روش اجایل را به طور کامل اجرا و یک محیط سازنده ایجاد کنید باید طرز تفکر چابک داشته باشید و شیوه تفکرتان را درباره اولویت‌ها و شکست خوردن تغییر دهید. در تفکر اجایل، اولویت‌ها باید به شکلی پیش‌دستانه انتخاب شوند نه اینکه در واکنش به اتفاقاتی که رخ می‌دهد رتبه‌بندی شوند و شکست نباید با تنبیه همراه باشد، بلکه باید از آن یاد گرفت. هنگامی‌که یک سازمان تغییرات لازم را برای استفاده از روش‌های چابک آغاز می‌کند، معمولاً کار را با یک تیم شروع می‌کند. مشتری و یا قسمت‌های دیگر سازمان با دیدن تحولات به وجود آمده توسط این تیم تمایل بیشتری به این طرز تفکر پیدا خواهد کرد و ترغیب مدیران نیز راحت‌تر صورت می‌پذیرد (Nikitina, et. al., 2012).

اکنون ما اسکرام و کانبان را به‌عنوان روش‌های چابک می‌شناسیم. به نظر می‌رسد اسکرام برای پروژه‌های توسعه و محصولات به‌خوبی قابل‌استفاده است. کانبان برای پشتیبانی تولید پاسخگوست. اسکرام‌بان که از مزایای هر دو روش بهره می‌برد برای پروژه‌های نگهداری مناسب‌تر است. بااین‌حال امروزه اسکرام‌بان در صنایع خدماتی که در آن هم پروژه‌های توسعه و هم نگهداری داریم به شکل گسترده‌ای در حال استفاده است (Bhavsar, et. al., 2020). به طور خلاصه آنچه که در اسکرام انجام می‌دهیم به شرح ذیل است:

- سازمان را به تیم‌های کوچک، خودسازمانده و فرا وظیفه‌ای تقسیم می‌کنیم.
- کار را به لیست‌های کوچک تحویل - دادنی تقسیم می‌کنیم و میزان تلاش نسبی برای هر مورد را تخمین می‌زنیم.
- زمان را به تکرارهای کوتاه با طول ثابت (معمولاً ۱ تا ۴ هفته) تقسیم می‌کنیم تا تکه‌ای از کار در پایان تحویل شود.
- با بینش به‌دست‌آمده در پایان هر تکرار، برنامه انتشار را بهینه کرده و اولویت‌ها را با همکاری مشتری به‌روز می‌کنیم.
- فرایند را با نگاه به عملکرد گذشته بهینه می‌کنیم.

در کانبان روش کار متفاوت است و به طور خلاصه اقداماتی به شرح ذیل صورت می‌پذیرد:

- جریان کار بصری می‌شود. کار به قطعاتی شکسته می‌شود. روی کارت‌هایی نوشته شده و روی تابلو قرار می‌گیرد. از ستون‌هایی با نام مشخص برای مشخص نمودن روند کار روی تابلو استفاده می‌شود.
- کار در حال انجام محدود می‌گردد.
- زمان لازم برای تکمیل کار اندازه‌گیری می‌شود، فرایند در جهت کم کردن این زمان در آینده و قابل پیش‌بینی نمودن آن بهینه می‌شود.

نتیجه مستقیم این تفاوت در قوانین، نحوه برخورد با موارد کار در طول زمان است. در اسکرام کارهایی را که قرار است در اسپرینت بعدی انجام شود انتخاب می‌کنیم. شما اسپرینت را قفل می‌کنید و تمام کارهای داخل آن را انجام می‌دهید و به طور معمول پس از چند هفته لیست کارها خالی می‌شود. در کانبان اندازه صف محدود است و به همین نسبت کارهایی در حال انجام محدود می‌شود. در اینجا اسپرینت و پایانی بر آن وجود ندارد. کار همین‌طور ادامه می‌یابد. به‌این‌ترتیب اسکرام‌بان از دو ویژگی مهم روش‌های ذکر شده بهره می‌برد:

- از ماهیت تجویزی اسکرام برای چابک بودن بهره می‌برد.
- از بهبود فرایند کانبان استفاده می‌کند و به تیم اجازه می‌دهد تا به طور مداوم فرایندشان رو بهبود ببخشند.

با بهره‌گیری از سیستم کششِ کانبان، جریان نرم‌تر می‌شود درعین‌حال که قابلیت فرایند نیز بهبود می‌یابد. اکنون ما اسکرام و کانبان را به‌عنوان روش‌های چابک می‌شناسیم (Ahmad, et. al., 2018). به‌این‌ترتیب می‌توان از بافرهای بین - فرایندی و نمودارهای جریان استفاده کرد تا نقاط ضعف و فرصت‌ها برای کایزن[1](بهبود مستمر) مقابل ما قرار گیرد. بر همین اساس می‌توان بهبودهای حاصل را به شرح ذیل بیان نمود:

- کیفیت
- تصمیم‌گیری و تحویل به هنگام[2]
- کوتاه کردن زمان تکمیل کار
- کایزن (بهبود مستمر)
- کم کردن و حتی حذف اتلاف‌ها
- بهبود فرایندها با افزودن ارزش‌های اسکرام در مواقع نیاز

چه زمانی استفاده از اسکرام‌بان بهتر خواهد بود:

- برای پروژه‌های نگهداری (کارهای رویداد - محور[3] و میزهای امداد[4])
- مراحل پکیج کردن کار
- پروژه‌هایی با داستان‌های مکرر و غیرمنتظره کاربر
- در مواردی که اسکرام با فرایندها، منابع و موارد جریان کار به چالش کشیده شده است.

[1] Kaizen
[2] Just in time (JIT)
[3] Event-driven work
[4] Help desk

به نظر می‌رسد اسکرام برای پروژه‌های توسعه و محصولات به‌خوبی قابل‌استفاده است. کانبان برای پشتیبانی تولید پاسخگوست (Qureshi, et. al., 2019). اسکرام‌بان که از مزایای هر دو روش بهره می‌برد برای پروژه‌های نگهداری مناسب‌تر است. بااین‌حال امروزه اسکرام‌بان در صنایع خدماتی که در آن هم پروژه‌های توسعه و هم نگهداری داریم به شکل گسترده‌ای در حال استفاده است.

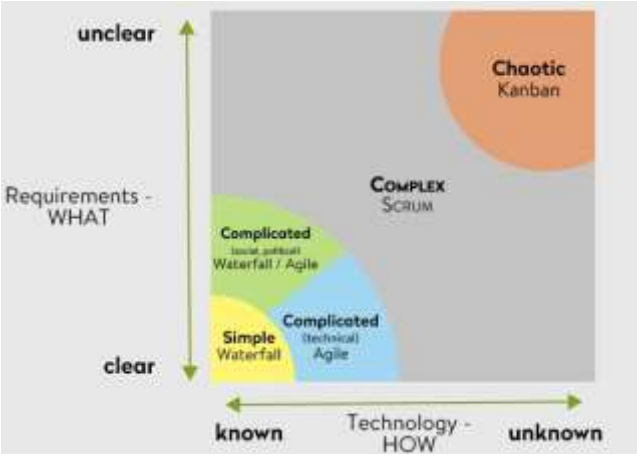
وقتی یک تیم بر روی پروژه‌ای کار می‌کند، همان اسکرام گزینه مناسبی است. اما تصور کنید چند تیم قرار است به‌صورت هم‌زمان بر روی یک پروژه کار کنند، در این موارد راهکار چیست؟ نفراتی که اسکرام را توسعه داده‌اند، به این موضوع نیز فکر کرده‌اند و بر اساس نیازهایی که برای هماهنگی چند تیم وجود داشته، نکسوس[1] را توسعه داده‌اند. عمده راهنمایی‌های چارچوب نکسوس توسط کن شوئبر تدوین شده است.

با آنکه می‌دانیم اسکرام و کانبان امتحان خود را پس داده‌اند اما این تحقیق توصیه می‌کند پیش از حرکت به سمت چابکی و تفکر ناب، موقعیت سازمان خود، نوع پروژه و محصول و انتظارات ذی‌نفعان را به طور کامل بررسی کنید. بری این منظور چارچوبی وجود دارد که کانه‌وین[2] نامیده می‌شود. این چارچوب در سال ۱۹۹۹ توسط آقای دیو اسنودن[3] ارائه شده است. یک چارچوب مفهومی برای کمک به تصمیم‌گیری است. این چارچوب خصوصیات پنج حوزه متفاوت را تعریف و مقایسه می‌کند: ساده،[4] دشوارفهم،[5] بی‌نظم،[6] پیچیده[7] و نابسامان[8] که زمانی رخ می‌دهد که نمی‌دانید در کدام حوزه قرار دارید. طبق مطالعات صورت‌گرفته، ابعاد متعدد توسعه نرم‌افزار فقط در یکی از دامنه‌های کانه‌وین قرار نمی‌گیرد (O'Connor, Lepmets, 2015). بیشتر فعالیت‌های توسعه نرم‌افزار در حوزه‌های دشوارفهم و پیچیده قرار می‌گیرد، اما می‌دانیم که کل فعالیت‌ها نیز در این دو حوزه نیستند و گاهی مسائل خیلی ساده خواهند بود.

تصویر ۳-۹ : چارچوب کانه‌وین

[1] Nexus
[2] Cynefin
[3] Dave Snowden
[4] Simple
[5] Complicated
[6] Chaotic
[7] Complex
[8] Disorder

پس از تعیین حوزه مسئله بر اساس چارچوب کانه‌وین، می‌توان نگاشتی بین روش و حوزه مسئله ایجاد کرد.



تصویر ۳-۱۰ : نگاشت بین روش و حوزه مسئله

مسئله دیگر تعیین محدودیت‌های اصلی سیستم است. در این زمینه از مفاهیم تئوری محدودیت‌ها[1] بهره می‌بریم. این تئوری در سال ۱۹۸۴ توسط گلدرت[2] ارائه شده است. نزدیکی مشخصی بین این تئوری و مفاهیم تفکر ناب در تولید و صنایع و سازمان‌ها وجود دارد که برای اجرای پروژه‌ها به کمک اسکرام‌بان سازنده خواهد بود.

یک راهنما در زمینه کاربست اسکرام‌بان توسط گری استراون[3] مطابق با کتاب کوری لاداس وجود دارد که ۷ گام را برای اجرای اسکرام‌بان پیشنهاد می‌کند. گام‌های ۱ تا ۵ منطبق بر فلسفه اسکرام هستند. تابلوی چند ستونه نمایش کارها به‌خوبی نمایشی است از کانبان، بااین‌حال توصیه می‌شود با مطالعهٔ سازمان، نوآوری بیشتری در اجرا داشته باشیم.

مراحل این روش به شرح ذیل می‌باشد:

- مصورسازی کارها بر روی تابلوی کانبان با ستون‌های مشخص حتماً اجرا شود.
- فرایند انتساب کارها به تیم‌ها و نفرات را متوقف کنید.
- کارهای در حال انجام را محدود کنید.
- با اضافه‌کردن ستون بافر در هر ستون سیستم کششی را اجرا کنید.
- با اضافه‌کردن ستونی مانند آماده[4] مرتب‌سازی کارها بر اساس اولویت را در این ستون انجام دهید.
- تخمین‌زدن را متوقف کنید.
- برنامه‌ریزی ماشه‌ای را به‌جای جلسات هر دوهفته یکبار امتحان کنید.

[1] Theory of constraints
[2] Eliyahu M. Goldratt
[3] Gary Straughan
[4] Ready

طبق فلسفه ناب، هر چیزی که ارزشی نمی‌سازد باید حذف شود. ازآنجایی‌که تخمین زدن بک‌لاگ ارزشی را اضافه نمی‌کند شدید توصیه شده تا فرایند تخمین زدن را کنار بگذارید. برنامه‌ریزی برای ورود کارهای جدید تنها زمانی صورت می‌گیرد که تعداد کارهای موجود در ستون Todo به کمتر از حد تعیین شده رسیده باشد.

۳- ۱۱ جدول زمان مروری بر تحولات و کارهای گذشته

اکنون‌که با عمده‌ترین روش‌های توسعه و مدیریت پروژه‌های سنتی و چابک آشنا شده‌ایم، در این قسمت در قالب یک جدول زمانی عمده‌ترین تحولات و کارهای صورت‌گرفته را مرور خواهیم کرد و متوجه خواهیم شد تعدادی از مفاهیم جدید ریشه در نظریه‌های قدیمی دارد و در مواردی روش‌های چابک مسیر کاملاً متفاوتی از روش‌های سنتی را طی کرده است. در جدول ۳-۱ تحولات مهم این صنعت از سال ۱۹۶۸ میلادی تا سال ۲۰۲۱ بررسی شده است.

جدول ۳– ۱ : جدول زمانی مروری بر تحولات و کارهای گذشته در حوزه توسعه نرم‌افزار

سال	تحولات مهم در حوزه مدیریت و توسعه نرم‌افزار
۱۹۶۸	مقاله ملوین کانوی[۱] در سال ۱۹۶۷ ارائه و رد شد، اما بعداً به‌عنوان «قانون کانوی» شناخته شد. این قانون بیان می‌کند: سازمان‌هایی که سیستم‌ها را طراحی می‌کنند، محصولاتشان توسط ساختار ارتباطی داخلی سازمان‌ها محدود می‌شود.
۱۹۷۰	بری بوهم[۲] روش Wideband Delphi را ارائه کرد که به شکلی پیش‌گام روش برنامه‌ریزی پوکر[۳] در اسکرام امروزی است.
۱۹۷۶	مجموعه مقالات پانزل[۴] در توصیف ابزارهایی با ویژگی‌های بسیار شبیه تست واحد که گواه سابقه طولانی آزمون واحد خودکار است.
۱۹۷۶	انتشار مقاله توسط گلنفورد مایرز[۵] که در آن به‌عنوان یک اصل موضوعی[۶] بیان می‌کند: لازم نیست یک توسعه‌دهنده کد خود را تست کند. از این زمان به‌عنوان عصر تاریک تست نرم‌افزار[۷] یاد شده است.
۱۹۷۷	ایجاد ابزار ساخت برای سیستم‌های یونیکس، اصل خودکارسازی ساخت نرم‌افزار ایده جدیدی نیست.
۱۹۸۰	بحث اساسی در زمینه توسعه افزایشی که در نسخه ویرایش شده‌ای از هارلان میلز[۸] با عنوان «اصول مهندسی نرم‌افزار» در بخش سیستم‌های فدرال IBM مطرح شده و به‌ویژه مقاله دائر[۹] که به تنظیم مراحل افزایش برای به حداکثر رساندن عملکردهای آن توصیه شده است.
۱۹۸۰	مفهوم «کنترل بصری»[۱۰] از سیستم تویوتا که به‌نوعی بعدها به «رادیاتورهای اطلاعاتی»[۱۱] تبدیل می‌شوند.

[1] Melvin Conway
[2] Barry Boehm
[3] Planning poker
[4] D. Panzl
[5] Glenford Myers
[6] Axiom
[7] Dark Ages of Developer Testing
[8] Harlan Mills
[9] Dyer
[10] Visual control
[11] Information radiators

۱۹۸۳	طیف گسترده‌ای از تکنیک‌های «آزمایش فاکتور انسانی[1]» که نمایانگر آزمایش قابلیت استفاده[2] است توسط Xerox PARC استفاده شد.
۱۹۸۴	یک مطالعه تجربی توسط بری بوهم، در مورد پروژه‌هایی با استفاده از نمونه‌سازی که یک استراتژی تکرارشونده دارد. این موضوع نشان می‌دهد که رویکردهای تکراری در همان زمان موردتوجه جدی قرار گرفته است که احتمالاً توسط عواملی مانند ظهور کامپیوترهای شخصی و رابط‌های گرافیکی کاربر هدایت شده است.
۱۹۸۴	ظهور مفهوم فاکتورینگ که بعدها به بازسازی کد[3] بدل می‌شود.
۱۹۸۴	توجه به رویکردهای افزایشی درحالی‌که انتقادات به روش‌های آبشاری از پیش‌تر شروع شده است.
۱۹۸۵	ارائه اولین جایگزین افزایشی صریح به‌جای رویکرد آبشاری توسط تام گلیب[4] که با نام EVO شناخته می‌شود.
۱۹۸۶	در یک مقاله مشهور بری بوهم، مدل مارپیچی توسعه و پیشرفت نرم‌افزار را ارائه کرد.
۱۹۸۶	مقاله نانوکا و تائیچی که با الهام از بازی راگبی فرایند تولید محصول از تعامل مداوم یک تیم چندرشته‌ای را مورد بررسی قرار می‌دهد و امروزه به‌عنوان الهام‌بخش روش اسکرام شناخته می‌شود.
۱۹۸۸ تا ۱۹۹۰	ظهور برنامه‌های دارای رابط گرافیکی[5] کاربر و چالش‌های تست این نوع جدید از نرم‌افزارها و ارائه طرح‌هایی برای تست توسط شرکت‌هایی چون مرکوری و ...
۱۹۸۸	معرفی مفهوم «جعبه زمان[6]» توسط اسکات شولتز[7] به‌عنوان سنگ بنای رویکرد نمونه‌سازی سریع تکراری
۱۹۸۹	معرفی تکنیک CRC توسط وارد کانینگهام[8] و کنت بک[9] در یک مقاله مشترک
۱۹۹۰	معرفی اصطلاح «بازسازی کد» در مقاله‌ای به قلم بیل اوپدیک[10] و رالف جانسون[11]
۱۹۹۰	باتوجه‌به افزایش محبوبیت ابزارهای توسعه سریع[12] و محیط‌های توسعه یکپارچه،[13] ابزارهای ساخت نیز با شهرت‌های جدید و متفاوتی بار دیگر مطرح می‌شوند.
۱۹۹۱	جیمز مارتین[14] و مفاهیم جعبه زمان و تکرارها و توسعه سریع مطرح می‌شوند.
۱۹۹۱	ظهور اولین چارچوب‌های تست واحد
۱۹۹۲	ابداع اصطلاح Dynamic Duo توسط لری کنستانتین[15]
۱۹۹۳	نتیجه مطالعات ویلسون و همکاران مبنی‌بر مزایای همکاری و مشارکت برنامه‌نویسان، برنامه‌نویسی زوجی
۱۹۹۳	الگوی اصلی جلسات روزانه و سرپایی توسط جیم کاپلین[16] ارائه می‌شود.

[1] Human factors testing
[2] Usability testing
[3] Code refactoring
[4] Tom Gilb
[5] GUI
[6] Timebox
[7] Scott Schultz
[8] Ward Cunningham
[9] Kent Beck
[10] Bill Opdyke
[11] Ralph Johnson
[12] RAD
[13] IDE
[14] James Martin
[15] Larry Constantine
[16] Jim Coplien

۱۹۹۳	ابداع اسکرام توسط جف سادرلند
۱۹۹۴	نوشتن چارچوب تست واحد برای زبان اسمال‌تاک توسط کنت بک
۱۹۹۴	روش توسعه پویای سیستم‌ها[۱] پس از مشخص‌شدن مشکلات توسعه سریع برنامه کاربردی آغاز شد. اولین تعریف از این روش در سال ۱۹۹۵ منتشر شد. پس از نظارت و بازبینی عملی، نسخه بعدی در همان سال و نسخه سوم آن در سال ۱۹۹۷ منتشر شد. در سال ۲۰۰۰، نسخه‌ای از این روش باتوجه‌به کسب‌وکارهای الکترونیکی منتشر شد و با نام eDSDM شناخته شد.
۱۹۹۵	مقاله آلیستر کاکبرن در مورد بیان دلایل توجه روزافزون به توسعه تکراری و افزایشی
۱۹۹۵	معرفی اسپرینت به‌عنوان پایه تکرار در اسکرام
۱۹۹۵	معرفی پادالگوها[۲] در توسعه نرم‌افزار
۱۹۹۵	کن شوئبر و جف سادرلند مشترکاً اسکرام را به دنیا معرفی می‌کنند.
۱۹۹۶	استیو مک‌کانل ساخت روزانه و آزمون دود[۳] را معرفی می‌کند.
۱۹۹۷	معرفی اسکرام روزانه که در مستندات اولیه وجود نداشت.
۱۹۹۸	عبارت اول تست به آزمون محور تغییر پیدا کرد.
۱۹۹۸	اولین مقاله در ارتباط با برنامه‌نویسی مفرط منتشر می‌شود.
۱۹۹۹	رابرت سی‌مارتین[۴] به شکل مشخص در ارتباط با چابک از اصطلاحات افزایشی و تکراری بهره می‌برد. توسعه ویژگی محور اولین‌بار در سال ۱۹۹۹ توسط پیتر کود، اریک لوفور و جف دلوکا معرفی شد. سپس در سال ۲۰۰۲ توسط استیفن پالمر و مک فلشینگ بهبودهایی در آن ایجاد شد و به‌عنوان فرایندی چابک و انطباق‌پذیر برای پروژه‌های متوسط و بزرگ معرفی گردید.
۲۰۰۰	تکنیک تست «شیء ساختگی[۵]» معرفی می‌شود.
۲۰۰۰	نمودار برن‌دان[۶] توسط کن شوئبر معرفی می‌شود.
۲۰۰۰	مفهوم سرعت[۷] تیم به برنامه‌نویسی مفرط وارد می‌شود.
۲۰۰۱	انتشار بیانیه چابک توسط ۱۷ توسعه‌دهنده دنیای نرم‌افزار
۲۰۰۱	جلب شدن توجه همگان به «برنامه‌نویسی ناب[۸]» با انتشار مقاله‌ای توسط مری پاپندیک
۲۰۰۱	اسکرام اسکرام‌ها[۹] توسط جف سادرلند در مقاله‌ای معرفی می‌شود.
۲۰۰۱	ران جفریز و معرفی «داستان‌های کاربر» در برابر «موارد کاربرد»
۲۰۰۱	ابداع اصطلاح «رادیاتورهای اطلاعات» توسط آلیستر کاک‌برن
۲۰۰۲	برنامه‌نویسی زوجی یا دونفره
۲۰۰۲	معرفی برنامه‌ریزی پوکر توسط جیمز گرنینگ[۱۰]

[1] DSDM
[2] Antipattern
[3] Smoke Test
[4] Robert C. Martin
[5] Mock Object
[6] Burndown chart
[7] Velocity
[8] Lean Programming
[9] The Scrum of Scrums
[1] James Grenning [0]

۲۰۰۳	معرفی نسخه صنعتی برنامه‌نویسی مفرط توسط جاشوآ کرفسکی[1]
۲۰۰۳	توسعه مبتنی بر آزمون پذیرش توسط کنت بک معرفی شد. این یک روش توسعه مبتنی بر ارتباط بین مشتریان تجاری، توسعه‌دهندگان و آزمایش‌کنندگان است.
۲۰۰۳	معرفی پینگ پونگ پروگرامینگ که برنامه‌نویسی زوجی و آزمون محور را در هم ادغام می‌کند.
۲۰۰۳	مری و تام پاپندیک کتاب توسعه نرم‌افزار ناب را منتشر می‌کنند.
۲۰۰۳	معرفی کتاب توسعه آزمون محور توسط کنت بک انجام می‌شود. توسعه رفتار محور در سال ۲۰۰۳ در ارتباط با توسعه آزمون محور توسط دن نورس ایجاد شد. سپس توسعه پیدا کرد و به‌عنوان یک رویکرد مجزا تحت عنوان رفتار محور در سال ۲۰۰۹ معرفی شد.
۲۰۰۳	معرفی تابلوی وظایف پنج ستونه توسط مایک کوهن
۲۰۰۳	طراحی دامنه محور[2] توسط اریک ایوانز[3] معرفی می‌شود.
۲۰۰۴	اجرای اسکرام در آمازون در تمام لایه‌های تجارت شرکت، نتایج در سال ۲۰۱۲ توسط اتلس[4] منتشر شد. گرچه روش‌های چابک از سال ۱۹۹۹ در این شرکت بکار می‌رفته است اما اوج این کاربردها به سال‌های ۲۰۰۵ تا ۲۰۰۹ برمی‌گردد. به طور مثال «قانون دو پیتزا»[5] از پیش از به‌کارگیری اسکرام در آمازون مطرح بوده است.
۲۰۰۴ تا ۲۰۰۶	تشکیل جلسات روزانه حول رویکردهای چابک، با تأکید بر تشکیل جلسات در کنار تابلوی وظایف به‌عنوان نمونه، تابلوی اسکرام و یا کانبان برای ایجاد درک بهتر در مورد پیشرفت پروژه و گلوگاه‌ها. ورود کانبان آن به دنیای نرم‌افزار نتیجه کارهای دیوید اندرسون می‌باشد که نتیجه تجربیات او و همکارانش در مایکروسافت را در فاصله سال‌های ۲۰۰۴ تا ۲۰۱۰ در برمی‌گیرد.
۲۰۰۵	استفاده از اصطلاح Backlog grommingکه بعدها به Backlog refinement تغییر یافت.
۲۰۰۵	مایکروسافت پس از تأخیرهای مکرر در عرضه Sql Server 2005 از اسکرام و XP برای بهبود فرایند توسعه نرم‌افزار بهره برده است.
۲۰۰۶	توجه بیشتر به توسعه رفتار محور و when-then canvas
۲۰۰۶	معرفی Niko-niko calendar که به اعضاء تیم اجازه ثبت روحیات خود در پایان آن روزکاری را می‌دهد.
۲۰۰۷	تشکیل گروه لیست پست‌سپاری Kanbandev[6] برای مرور و بحث در ارتباط با کانبان و مفاهیم چابک
۲۰۰۷	انتشار گزارش‌های اولیه از دیود اندرسون[7] که به‌کارگیری کانبان در پروژه‌ها را نشان می‌داد.
۲۰۰۷	محبوبیت تابلوهای سه ستونه کانبان که عبارات To Do و In progress و Done را شامل می‌شد.
۲۰۰۷	ارائه مقالات در زمینه ترکیب تفکر ناب و چابک برای ایجاد روشی در جهت بهبود روند تولید محصولات کارخانه‌ای در حوزه صنعت، صنعت از سال‌ها پیش تحت کارهای پیشتازان ژاپنی این عرصه با مفاهیم تولید ناب آشنا بوده است، تحولات چابک در حوزه نرم‌افزار بار دیگر ترکیب و اشتراک مزایای این تفکرات را در پی داشت. امروزه اکثر صنایع مفاهیم پایه تولید خود را با تفکر چابک گره می‌زنند تا تولیدی با کیفیت و به‌موقع داشته باشند که رضایت مشتری و گسترش کسب‌وکار را همراه خواهد داشت. زیمنس[1] از ادغام اسکرام و توسعه آزمون

[1] Joshua Kerievsky
[2] Domain-driven design
[3] Eric Evans
[4] Alan Atlas
[5] Two pizza rule
[6] Mailing list
[7] David Anderson

	محور و تست خودکار بهره برده است.
۲۰۰۸	در مقاله منتشر شده توسط گابریل بنفیلد[۲] به درس‌هایی که یاهو از به‌کارگیری اسکرام گرفته اشاره می‌شود. ارتقا زمان تحویل پروژه در شرکت اینتل پس از تلفیق روش قدیمی و اسکرام در مقاله‌ای به قلم پت الور[۳] خود گویای مزایای چنین ترکیباتی است.
۲۰۰۹	اقبال به توسعه عملیات یا دواپس توسط پاتریک دبوآ
۲۰۰۹	معرفی کتاب اسکرام‌بان توسط کوری لاداس[۴] بر پایه کارهای قبلی و پروژه‌های انجام شده
۲۰۰۹	افزایش مقیاس‌پذیری چابکی با استفاده از روش ناب در مقالات متعددی مطرح می‌شود.
۲۰۱۱	معرفی نسخه‌های اولیه SAFe توسط دین لفینگول[۵]
۲۰۱۱ تاکنون	ظهور و تکمیل مفاهیم چابک مقیاس‌پذیر با روش‌هایی چون: • Nexus • Scrum at scale • Enterprise Scrum • XSCALE • LeSS • Setchu • Agile path • Holistic Software Development • Scrum of Scrums • Agile Porfolio Management •
۲۰۱۲	مطالعات بر توسعه و افزایش بهره‌وری صنعتی متمرکز است. کارخانه‌های اتومبیل‌سازی، شرکت‌های دارویی به‌خوبی از ترکیب تفکر چابک و ناب بهره می‌برند. مقاله پیتر گرین[۶] از مزایای چابک در موفقیت فضای فوق رقابتی آن زمان برای شرکت ادوبی می‌گوید.
۲۰۱۵	مقاله جیمز کاپلین[۷] در مورد مزایای تفکر دونفره و برنامه‌نویسی زوجی
۲۰۱۵	مقالات و مطالعات در زمینه جایگاه روش‌های ترکیبی کانبان و اسکرام در تضمین کیفیت محصول نرم‌افزاری ارائه می‌شود.
۲۰۱۷	گردآوری و تکمیل تعاریف DAD[۸] توسط اسکات امبر[۹] و همکارانش
۲۰۱۷	ارائه اولیه مفاهیم تست چابک توسط جانت گرگوری[۱۰] و لیزا کریسپین[۱۱]
۲۰۱۷	بر اساس نتایج مطالعات در این سال‌ها گرایش‌هایی برای ترکیب روش‌های سنتی و چابک بیشتر شده است. دلیل

[1] Siemens
[2] Gabrielle Benefield
[3] Pat Elwer
[4] Corey Ladas
[5] Dean Leffingwell
[6] Peter Green
[7] James Coplien
[8] Disciplined agile delivery
[9] Scott Ambler
[10] Janet Gregory
[11] Lisa Crispin

	عمده ساختار سنتی سازمان‌ها و مدیریت در حرکت به سمت چابکی است. بر همین اساس روش‌هایی که مزایای هر دو روش را به کار می‌برند موردتوجه قرار می‌گیرد. به‌این‌ترتیب چابکی در پاسخ به تغییرات به همراه نظام‌مند بودن روش‌های سنتی می‌تواند برای حرکت یک سازمان به سمت چابکی الگوی مناسبی ارائه دهد.
۲۰۲۰	ارائه نسخه جدید سند راهنمای اسکرام، با اصلاحات جدید و کاهش پیچیدگی‌ها و تأکید بیشتر بر افراد. این راهنما در ۲۵ سالگی ارائه این راهنما توسط مبدعان آن جف سادرلند و کن شوئبر معرفی شد. تغییرات این نسخه که از اهمیت ویژه‌ای برخوردار است به شرح ذیل می‌باشد: • تجویزات کم‌تر • تمرکز تیم روی یک محصول • معرفی هدف محصول • معرفی هدف اسپرینت • خود مدیریتی بالاتر از خودسازماندهی • تأکید بر چرایی ارزشمندی هر اسپرینت • ساده‌سازی زبان راهنما برای مخاطبان عمومی‌تر
۲۰۲۱	هماهنگی برای برگزاری کنفرانس XP برای ارائه آخرین تحولات این حوزه

۳ – ۱۲ مقایسه روش‌ها

در این قسمت شباهت‌ها و تفاوت‌های اصلی سه روش اسکرام، کانبان و اسکرام‌بان در قالب یک جدول مقایسه می‌شوند. این جدول می‌تواند تااندازه‌ای به تیم‌ها و سازمان‌ها در انتخاب یکی از این روش‌ها برای چابک‌سازی کمک نماید.

جدول ۳–۲ : مقایسه اسکرام، کانبان و اسکرام‌بان

ردیف		اسکرام	کانبان	اسکرام‌بان
۱	**تیم**	بین ۳ تا ۹ نفر	بدون محدودیت	عدد خاصی مشخص نشده است.
۲	**نقش‌ها**	سه نقش: اسکرام‌مستر، مالک محصول و تیم توسعه	تجویز نشده و می‌تواند شامل متخصصین و نفرات مرتبط باشد و یا از نقش‌های اسکرام بهره ببرد.	غیرتجویزی است اما وجود نفرات متخصص لازم است.
۳	**جلسات**	برنامه‌ریزی اسپرینت، اسکرام روزانه، بازبینی و بازنگری اسپرینت	جلسات ایستاده روزانه و سایر جلسات مبتنی بر تقاضا	از ترکیب دلخواهی از جلسات اسکرام و کانبان می‌توان بهره برد.
۴	**قوانین**	به شدت سخت‌گیرانه و تجویزی	قوانین انعطاف‌پذیر	بین اسکرام و کانبان و نه سخت‌گیرانه
۵	**تکرارها**	بین ۱ تا ۴ هفته (هر اسپرینت)، دوهفته رایج‌تر است.	جریان مداوم و پی‌درپی بر اساس تقاضا و نیاز	تکرارهایی که بهتر است بیشتر از ۲ هفته نشود.
۶	**تابلو**	بعد از هر اسپرینت بازنشانی می‌شود.	پیوسته استفاده می‌شود.	پیوسته استفاده می‌شود.
۷	**روتین**	اسپرینت	بر اساس تقاضا و برنامه‌ریزی انتشار	بر اساس تقاضا و برنامه‌ریزی باکت

	برنامه‌ریزی			
۸	**تخمین وظایف**	قبل از اسپرینت انجام می‌شود.	اختیاری و در حین برنامه‌ریزی	اختیاری
۹	**اختصاص وظایف**	به اعضا اختصاص داده می‌شود.	توسط اعضا انتخاب می‌شود.	به اختیار اعضا انتخاب می‌شود.
۱۰	**محدودکردن وظایف**	به اسپرینت محدود می‌شود.	با WIP محدود می‌شود.	با WIP محدود می‌شود.
۱۱	**معیارهای عملکرد**	Burndown chart	Lead and Cycle time Cumulative flow	Lead and Cycle time Average cycle time
۱۲	**افزودن وظیفه جدید**	اجازه داده نشده است. گاهی با نظر مالک محصول	آزاد	آزاد

۳- ۱۳ جمع‌بندی

در این فصل به‌مرور انواع متدولوژی‌ها و چارچوب‌های تفکر چابک پرداختیم. معایب و مزایای این روش‌ها مورد بررسی قرار گرفت. مشخص شد نوع پروژه، بودجه و زمان، تعداد افراد حاضر در تیم و سازمان، تفکر مدیران سازمان‌ها در انتخاب یک رویکرد نقش عمده‌ای دارند. زمان‌بندی، کنترل ریسک‌ها و مباحث مدیریت مهندسی نرم‌افزار در قالب اجرای یکی از این روش‌ها به نتایج بهتری می‌انجامد. آنچه که مشخص است، می‌توان این روش‌ها را برای تعداد محدودی از پروژه‌های سازمان به‌صورت آزمایشی مورداستفاده قرارداد و از نتایج آن برای انتخاب رویکرد متناسب با سازمان بهره برد. امکان استفاده ترکیبی از این روش‌ها نیز به بهره‌برداری مؤثر از مزایای هر روش کمک شایانی می‌کند. بر همین اساس در فصل بعد به دنبال بررسی مزایا و معایب روش‌های ترکیبی از جمله اسکرام‌بان‌فال خواهیم بود.

فصـــل چهارم

استفاده از متدولوژی

ترکیبی اسکرام‌بان‌فان

مقدمه

مطابق مطالب فصل گذشته پی بردیم که متدولوژی اسکرامبان، می‌تواند بر تعدادی از این چالش‌های توسعه نرم‌افزار از جمله: کنترل جریان کار، مدیریت زمان تدارک، تحویل مداوم و یکپارچه غلبه کند. بااین‌وجود اسکرامبان نمی‌تواند همه چالش‌ها را پوشش دهد. یک متدولوژی نوآورانه مانند اسکرامبان‌فال می‌تواند در حل برخی از مسائل پروژه یاری‌دهنده باشد. اسکرامبان‌فال، ترکیب چابکی از اسکرام و کانبان به همراه روش آبشاری در مدیریت مهندسی نرم‌افزار است. در این فصل ابتدا معماری سازمانی را مورد بررسی قرار می‌دهیم، سپس به تشریح متدولوژی ترکیبی اسکرامبان‌فال می‌پردازیم و مزایا و معایب بکارگیری آن در سازمان‌ها و صنایع را بررسی می‌کنیم. یک نسخه از این متدولوژی را در چارچوب فرایند اکلیپس پیاده‌سازی می‌کنیم و از آن به‌عنوان راهنمایی برای ساخت نسخه‌های سفارشی برای سایر صنایع و سازمان‌ها بهره می‌بریم.

۴ – ۱ معماری سازمانی

معماری سازمانی[1] امروزه یکی از عوامل مهم و کلیدی در انجام مأموریت‌های یک سازمان محسوب می‌شود. امروزه تهیه و تدوین برنامه‌های جامع فناوری اطلاعات با رویکرد معماری سازمانی نیاز اصلی مدیران برای کاهش هزینه‌ها، افزایش کارایی، سودآوری، بهره‌وری و اثربخشی است. هدف معماری سازمانی این است که فناوری اطلاعات را از حالت یک ابزار خارج کرده و به یکی از منابع سازمان در کنار سایر منابع سازمان تبدیل نماید به شکلی که در خدمت مأموریت‌های سازمان باشد. تعریف رسمی معماری سازمانی به شرح ذیل است: «معماری سازمانی رویکردی است یکپارچه و جامع که جنبه‌ها و عناصر مختلف یک سازمان (سیستم) را با نگاه مهندسی تفکیک و تحلیل می‌نماید و شامل مجموعه مستندات، مدل‌ها، استانداردها و اقدامات اجرایی برای تحول از وضعیت موجود به وضعیت مطلوب با محوریت فناوری اطلاعات است که در قالب یک چرخه تکرارپذیر اجرا شده و به‌صورت مداوم توسعه و به‌روزرسانی می‌شود». مفهوم و کاربرد معماری سازمانی مبتنی بر دو اصل محوری است:

- اصل اول: تقدم برنامه‌ریزی و طراحی بر پیاده‌سازی و اجرا
- اصل دوم: مهندسی همه جوانب و عناصر سازمان به‌صورت یکپارچه

مزایا و دستاوردهای معماری سازمانی بر اساس دو اصل محوری گفته شده، به‌قرار زیر است:

- کاهش دوباره کاری‌ها و اشتباهات در اجرا
- بهره‌وری بهتر و طول عمر بالاتر
- قابلیت توسعه و گسترش در آینده
- کاهش هزینه‌های پیاده‌سازی، نگهداشت و توسعه
- استفاده بهینه از زمان و منابع موجود
- پیش‌بینی و آمادگی برای شرایط آینده
- نگاه جامع و یکپارچه به مدیریت و تحول سازمان

[1] Enterprise architecture

- توازن و تناسب بین جوانب و عناصر سازمان
- هم راستایی بین کسب‌وکار با فناوری اطلاعات

چارچوب‌های معماری سازمانی به‌صورت کلی به چهار دسته اصلی تقسیم می‌شوند:

- چارچوب‌های معماری سازمانی عمومی (همه منظوره): چارچوب زکمن، چارچوب سازمانی اپن گروپ یا توگف
- چارچوب‌های معماری سازمانی ملی (دولتی)
- چارچوب‌های معماری سازمانی نظامی
- چارچوب‌های معماری سازمانی مختص صنایع

بااین‌حال فرایند معماری سازمانی بسیار کند و فرسایشی هستند و در نتیجه موجب عدم اجرای صحیح، توقف‌های زیاد و حتی شکست معماری سازمانی می‌شود. این آشفتگی و تلاطم در مسئله به کمک چابکی قابل‌حل می‌باشد. چابکی در این موضوع به توانایی برای ایجاد و پاسخگویی به تغییرات محیط متلاطم تجاری گفته می‌شود. متدولوژی‌های چابک تطبیق‌پذیر هستند و خودشان را با تغییرات تطبیق می‌دهند. رویکرد چابک در علوم مختلف نتایج درخشانی ایجاد نموده و در حوزه معماری نرم‌افزار و معماری سازمانی به‌خوبی بکار گرفته شده است. سؤال دیگر این است که چرا به چارچوبی برای معماری سازمانی نیاز داریم؟ پاسخ این است که: چارچوب به معنای نظام، روش و استاندارد انجام معماری است، لذا بدون داشتن چارچوب (استاندارد و دستورالعمل) مناسب، انجام معماری ممکن نبوده یا نتیجه‌بخش نیست (شمس و راضی، ۱۳۸۶).

چارچوب باید بر اساس نیاز داخلی تدوین و سفارشی شود بنابراین هر سازمان باید از طریق انجام مناسب معماری سازمانی، نیازمندی‌های خود را شناخته و راهکارهای حل مشکلات را کشف نماید. مشکل کندی و پرهزینه بودن اجرای معماری سازمانی به تغییرات مداوم در حوزه فناوری و مأموریت‌های سازمان برمی‌گردد. فرایند معماری سازمانی باید در محیط‌های پرتلاطم تجاری با کمترین هزینه، کمترین زمان و به شکل بهینه انجام پذیرد. سؤال اصلی اینجاست که آیا همه تغییرات فناوری و مأموریتی سازمان را می‌توان پیش‌بینی کرد؟ پاسخ این است که سازمان‌های امروزی به دلیل گستردگی و پیچیدگی و رقابت‌های گسترده و تغییرات و پیشرفت‌های مکرر فناوری در یک دریای مواج حرکت می‌کنند لذا همه مشکلات و تغییرات احتمالی را نمی‌توان پیش‌بینی کرد. معماری سازمانی چابک یک الگوواره طراحی و مدل‌سازی برای معماری سازمانی است که از ویژگی تطبیق پذیری خود در مقابل تغییرات مداوم برای پیاده‌سازی معماری سازمانی با هزینه مناسب در زمان معقول بهره می‌برد و پاسخ مناسبی برای اجرای دقیق معماری سازمانی ارائه می‌دهد. مشکلاتی که در هنگام بروز تغییرات پیش‌بینی‌نشده در جریان فرایند معماری سازمانی موجب کندی اجرای فرایند و حتی شکست آن می‌شود را می‌توان به شرح ذیل برشمرد:

- پیچیدگی تولید محصولات و فرآورده‌های معماری سازمانی
- تولید حجم عظیمی از مستندات
- وجود افراد تطبیق‌ناپذیر در سازمان و حتی در تیم معماری سازمانی
- اصرار بر مسائل تکنیکی به‌جای همکاری و کار تیمی و تعاملات
- استفاده از متدولوژی‌های نامناسب
- عدم وجود خلاقیت در افراد و دنبال‌کردن محض برنامه‌ها و چارچوب‌ها

- وجود فازهای طولانی در اجرا
- برگزاری جلسات غیرکارشناسی

اما مزایای بکارگیری روش‌های چابک در فرایند معماری سازمانی را می‌توان به شرح ذیل بیان کرد:

- برقراری و تسهیل ارتباط بین ذی‌نفعان و سهام‌داران
- انتقال ساده و مؤثر تجربیات
- مدیریت بهتر تغییرات
- کاهش حجم مستندات کاغذی
- کاهش هزینه‌ها، زمان اجرا و افزایش کیفیت
- افزایش انگیزه در کارکنان و ذی‌نفعان
- بهبود مستمر روش‌ها و فرایندها
- افزایش انسجام در تیم معماری سازمانی
- افزایش استفاده از مؤلفه‌ها و الگوهای مناسب

در چارچوب معماری سازمانی، دامنه معماری به شش زیر دامنه تقسیم می‌شود:

- زیر دامنه برنامه راهبردی (به‌عنوان ورودی معماری سازمانی)
- زیر دامنه معماری کسب‌وکار
- زیر دامنه معماری اطلاعات و داده
- زیر دامنه معماری نرم‌افزارهای کاربردی
- زیر دامنه معماری زیرساخت فناوری
- زیر دامنه معماری امنیت

چارچوب معماری سازمانی همان گونه که از نام آن مشخص است یک چارچوب و مدل است و تا زمانی که به‌صورت مؤثر و کاربردی توسط متولیان مربوطه مورداستفاده و پیاده‌سازی قرار نگیرد، منجر به تغییروتحول در سطح سازمان و دستگاه‌های زیرمجموعه نخواهد شد. در این پژوهش، بیشتر بر زیر دامنه معماری نرم‌افزارهای کاربردی، اطلاعات و داده تمرکز شده است و به‌صورت خاص نحوه مدیریت و توسعه برنامه‌های کاربردی به کمک یک متدولوژی ترکیبی چابک به نام اسکرامبان‌فال موردتوجه قرار گرفته است. پیش از آن پادالگوهای مطرح در یک سازمان یا یک تیم توسعه نرم‌افزار را مرور خواهیم کرد.

۴ – ۲ پادالگوها

پادالگو یا ضدالگو،[1] راه حلی است تکرارشونده برای یک مسئله در حوزه‌ای خاص که در عمل موفق نبوده است. پادالگوها در مهندسی نرم‌افزار برای تکرار نکردن اشتباهات گذشته کشف می‌شوند. اگر یک الگو در حوزه موردنظر اعمال نشود ممکن است به پادالگو تبدیل شود. از مهم‌ترین پادالگوها می‌توان به پادالگوهای فرایند اشاره کرد. پادالگوهای فرایند، راه‌حل‌های نامناسب برای

[1] AntiPattern

مشکلات در فرایند ایجاد نرم‌افزار هستند. این پادالگوها به سه دستهٔ پادالگوهای توسعه، معماری و مدیریتی تقسیم می‌شوند. لازم است پیش از تدوین یک متدولوژی برای یک تیم یا سازمان این پادالگوها شناسایی و راه‌حل‌های اجتناب از آنها اجرایی شود.

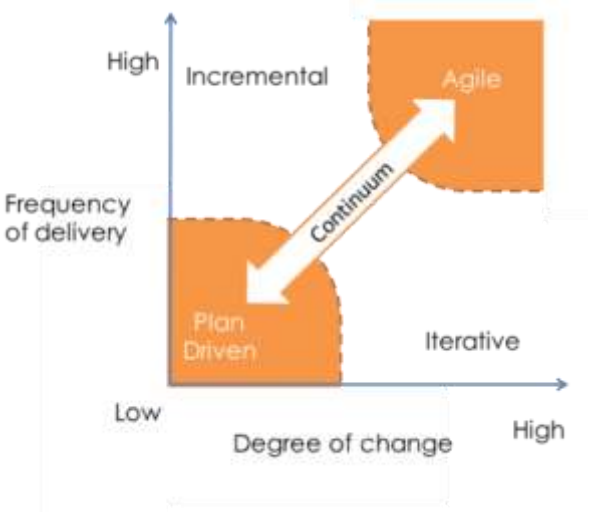
جدول ۴– ۱ : پادالگوهای فرایند ایجاد نرم‌افزار

پادالگوهای مدیریتی	
فلج تجزیه‌وتحلیل	تلاش برای دستیابی به کامل بودن و کمال در مرحله تجزیه‌وتحلیل به انسداد پروژه در این مرحله منتهی می‌شود.
خطر ایمیل	ایمیل یک روش مفید اما ناپایدار برای برقراری ارتباط است. از استفاده از ایمیل برای پیام‌های حساس، بحث‌برانگیز یا تقابلی خودداری کنید.
مرگ در برنامه‌ریزی	برنامه‌ریزی خیلی زیاد برای پروژه‌های نرم‌افزاری باعث به تعویق افتادن کارهای توسعه و انجام کارهای بی‌فایده خواهد شد. بهتر است روند توسعه تکراری که شامل برنامه‌ریزی معقول متناسب با واقعیت‌های شناخته شده است دنبال شود و سپس برنامه‌ریزی مجدد افزایشی در ادامه انجام شود.
سوءمدیریت پروژه	عدم توجه به مدیریت فرایند تولید نرم‌افزار منجر به بدون جهت شدن کار و مشکلات دیگر می‌شود و مشکلات پروژه به‌موقع تشخیص داده نمی‌شود. نظارت و کنترل مناسب پروژه‌های نرم‌افزاری برای موفقیت در فعالیت‌های توسعه ضروری است.
ترس از پیروزی	افراد تیم، شامل توسعه دهندگان با نزدیک شدن به پایان پروژه کارهای عجیبی انجام می‌دهند زیرا از پایان کار یا یک بخش از پروژه به‌واسطه وجود مشکلات در کارهایشان هراس دارند و سعی در تعویق کار خواهند داشت. هنگامی‌که اتمام پروژه نزدیک است، اعلام صریح موفقیت برای محیط پروژه مهم است.
دود و آیینه‌ها	کاربران نهایی به‌اشتباه تصور می‌کنند که نمایش شکننده یک امکان، قابلیتی است که برای استفاده عملیاتی آماده است. رعایت اصول اخلاقی مناسب برای مدیریت انتظارات، ریسک، مسئولیت و عواقب آن در محاسبات فروش و موقعیت‌های بازاریابی مهم است.
دشمنی	مدیرانی که با همتایان خود طولانی‌مدت درگیر می‌شوند، تأثیرات منفی جدی بر کارکنان خود دارند. برطرف‌کردن اختلافات با تسهیل‌کننده حرفه‌ای و یا جلسات ناهار کاری مؤثر خواهد بود.
پادالگوهای معماری	
معماری مفروض	سیستم بدون یک معماری مستند شده توسعه می‌یابد، اغلب از معماری پروژه‌ها و تجربیات قبلی استفاده می‌شود. درحالی‌که باید معماری بر اساس چندین دیدگاه مرتبط با ذی‌نفعان تهیه و مستند شود.
هوای خودت را داشته باش	در فرایندهای نرم‌افزاری سند محور، افراد به‌جای تصمیم‌گیری در مورد طرح معماری، مدام گزینه‌های دیگری را به سند اضافه می‌کنند و حجم آن را بالا می‌برند تا در آینده مسئولیتی متوجه آنها نباشد.
طراحی در کمیته	طرح‌های کمیته بیش از حد پیچیده و فاقد چشم‌انداز معماری مشترک است. طرحی سنگین با چسبندگی پایین با مفهوم و تعریف غیرواضح ایجاد می‌شود. بهتر است تیمی کوچک طراحی را انجام دهد و سپس به تصویب کمیته برساند.
اختراع مجدد چرخ	ساخت و طراحی هر سیستم به‌صورت مجزا بدون درنظرگرفتن وجوه تشابه و تجربیات پروژه‌ها و کارهای قبلی. بهتر است دانش کارهای قبلی استخراج و ثبت گردد تا در پروژه‌ها و تکرارهای بعدی قابل‌استفاده باشند.
پادشاه قدیمی شهر یورک	در این پادالگو مدیران با دیدگاه قدیمی تساوی‌طلبانه در تصمیم‌گیری‌ها به رأی همه افراد تیم وزن یکسان می‌دهند. از آنجایی در تیم‌ها، افراد با دید پیاده‌سازی نفرات بیشتری را نسبت به افراد با دید انتزاعی تشکیل می‌دهند. در تصمیم‌گیری‌ها تأثیر بیشتری خواهند گذاشت. راه‌حل دادن وزن بیشتر به نفرات با دید انتزاعی در تصمیم‌گیری‌ها خواهد بود.
کد اسپاگتی	یک ساختار موردی نرم‌افزار باعث ایجاد مشکلاتی در بهینه‌سازی و توسعه بیشتر برنامه خواهد شد. به دلیل طراحی ضعیف، مجموعه‌ای از کدها در داخل هم پیچ‌وتاب خورده و ساختاری شبیه ظرف ماکارونی را شکل می‌دهند. خوانایی درک نحوه انجام یک کار در این مدل به شدت پایین می‌آید. توسعه تکراری و ریفکتورینگ راه‌حل این ضد الگو است.

چاقوی ارتشی سوئیسی	توجه بیش از حد در طراحی واسط‌ها منجر به اشیایی با متدهای فراوان می‌شود که سعی دارند هر نیاز احتمالی را پیش‌بینی کنند. این ضد الگو منجر به طراحی‌هایی می‌شود که درک، استفاده و اشکال‌زدایی آن دشوار است. راه‌حل، تعریف دقیق و مشخص هر جز و استفاده از انتزاع به شکل مناسب است.
پادالگوهای توسعه	
جریان مواد مذاب	به کد مرده و طراحی فراموش شده اشاره دارد که دیگر امکان تغییر آن وجود ندارد. موقع توسعه نرم‌افزار همه چیز مانند مواد مذاب زنده و قابل تغییر است اما با گذشت زمان مانند مواد مذاب سر و سخت می‌شوند. کد مرده از نشانه‌های نگهداشت ناپذیری سیستم است. از دلایل بروز این مشکل می‌توان به: گرگ‌های تنها در تیم توسعه، استفاده از متدهای کلاسی که خود هنوز تکمیل نشده و عدم حذف روش‌های آزمایشی پیاده‌سازی یک متد اشاره کرد. راه‌حل استفاده از فرایندهای کنترل پیکربندی است که کد مرده را حذف و تکامل و ریفکتور طراحی در جهت افزایش کیفیت می‌باشد.
دیدگاه مبهم	مدل‌ها بدون یک دیدگاه مشخص تولید می‌شوند. عدم وضوح در دیدگاه مدل‌سازی منجر به ابهامات مشکل‌ساز در مدل‌های شی می‌شود. راه‌حل مدل‌سازی هر دیدگاه به طور جداگانه در فازهای مختلف است.
چکش طلایی	به تکنولوژی یا مفهومی گفته می‌شود که به تعداد زیای از مشکلات اعمال می‌شود. شعار این پادالگو این است که «وقتی چکش تنها ابزار است، بقیه چیزها میخ است». راه‌حل افزایش دانش توسعه دهندگان است تا افراد بتوانند گزینه‌های مختلف را برای حل مسائل در نظر بگیرند.
قدم‌زدن در میدان مین	استفاده از تکنولوژی به‌خصوص موضوعات جدید ممکن است با خطاهای گسترده‌ای در حین استفاده و بعد از تحویل به مشتری همراه باشد. این پادالگو اخطار می‌دهد اگر بدون آگاهی در این میدان قدم بگذارید قطعاً شکست خواهید خورد. راه‌حل به اهمیت تست در توسعه تأکید می‌کند. توسعه آزمون محور و حتی گسترش نفرات تیم تست نسبت به تیم توسعه هم پیشنهاد می‌گردد.
مدیریت قارچی	تیم توسعه از کاربران سیستم جدا نگه‌داشته می‌شوند. شعار این است «تیم توسعه را در تاریکی نگه دار و تغاله اطلاعات را به آنها بده». تیم توسعه به دور از مشتری نگه داشته می‌شود تا از ایجاد مسئولیت‌های جدید به دلیل ارتباط جلوگیری شود. مدیریت قارچی فرض می‌کند که نیازمندی‌ها در شروع پروژه به طور کامل توسط تحلیلگران شناسایی شده است و این نیازمندی‌ها در طول پروژه ثابت خواهد ماند. این پادالگو و مشکلات ناشی از آن باعث چرخش از روش‌های آبشاری و سنتی له تکراری و افزایشی شد. امروزه توسعه در قالب تیم‌های چابک خودسازمانده و فرا وظیفه‌ای که تعاملات بالایی با مشتری و کاربر نهایی دارند انجام می‌شود.
شی یا کلاس خدا	در شی گرایی وقتی یک کلاس، کارهای بسیار زیادی را به صورت انحصاری انجام می‌دهد اتفاق می‌افتد. به این ضد الگو Blob هم گفته می‌شود. یک دیدگاه روالی در شی گرایی باعث ایجاد چنین کلاس بزرگی با مجموعه وظایف متعدد می‌شود در حالیکه سایر کلاس‌ها تنها نقش نگهداری پاره‌ای از اطلاعات بدون کارکرد خاصی را برعهده خواهند داشت. راه حل ریفکتور و جداسازی این وظایف در کلاس‌های دیگر است.
برنامه‌نویسی مبتنی بر کپی مداوم قطعات کد	برنامه‌نویس از قاعده «کار تکراری نکن» بهره نمی‌برد و به‌جای ایجاد راه‌حل‌های عمومی اقدام به درج تکه کدهای موجود در جاهای دیگر برنامه به‌صورت تکراری می‌کند. راه‌حل ایجاد باکس‌هایی از کد با قابلیت استفاده مجدد است تا مشکلات نگهداری ناشی از چنین تکرارهایی با اختلافات ناچیز به حداقل برسد.
منقضی شدن پیوسته	اشاره به نسخه‌های جدید و مداوم بسته‌ها و چارچوب‌های نرم‌افزاری دارد. راه‌حل غلبه بر این تغییرات مداوم استفاده از «استاندارد سیستم باز» است. در واقع مجموعه‌ای از استانداردها که با تغییرات نسخه‌های مختلف نرم‌افزارها و زیرساخت‌ها تغییر نمی‌کنند.
تجزیه تابعی	این پادالگو توسط برنامه نویسانی با ذهنیت برنامه‌نویسی روالی و تابعی وارد سیستم‌های شی گرا می‌شود. درست است که تجزیه وظایف در قالب چند کلاس به‌جای یک کلاس بزرگ موردی مناسب به حساب می‌آید اما در این پادالگو با نادیده گرفتن تفکر شی گرایی تنها مجموعه‌ای از توابع در کلاس‌های جدید جداسازی می‌شوند و ساختار شی گرای برنامه از دست می‌رود.

۴ – ۳ از برنامه محور تا چابک

اگر چرخه‌های عمر پروژه‌های مختلف را در یک پیوستار بررسی کنیم، از یک طرف توسعه برنامه-محور و در سوی دیگر به چابک می‌رسیم. برای درک بهتر این پیوستار دو جنبه اصلی چابکی یعنی، تحویل زودهنگام که غالباً صورت می‌گیرد و سازگار شدن با تغییر را در نظر می‌گیریم. اگر بخواهیم آن را بر روی یک نمودار دوبعدی ترسیم کنیم، چیزی مانند شکل ذیل را خواهیم داشت.



تصویر ۴-۱ : مقایسه توسعه برنامه محور و چابک

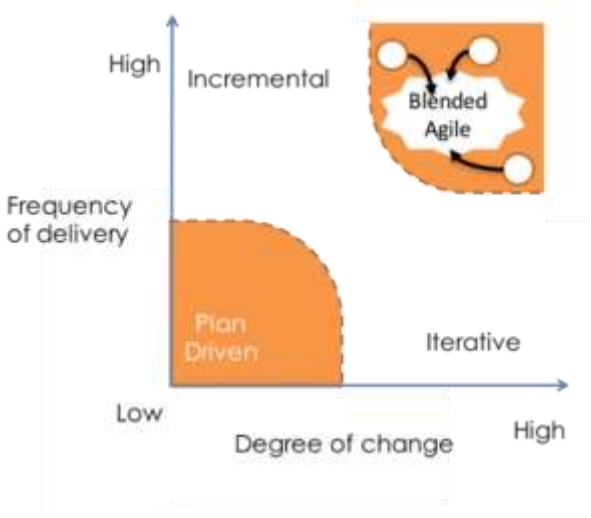
در این پیوستار از سمت چپ پایین که رویکرد برنامه-محور نامیده می‌شود تا بالا سمت راست که رویکردهای چابک نامیده می‌شوند، درجات مختلف تحویل (اشاره به افزایشی بودن) و تغییر (اشاره به تکراری بودن فرایند) وجود دارد. آن دسته از تکنیک‌هایی که به درجه بالایی از تحویل و انطباق‌پذیری دست می‌یابند را چابک می‌نامیم.

۴ – ۴ مخلوط یا ترکیبی

تقسیم‌بندی قسمت ۴-۱ خیلی ساده است. در دنیای واقعی، ما فقط از یک رویکرد استفاده نمی‌کنیم. تقریباً همیشه تکنیک‌های مختلف را با هم ترکیب می‌کنیم. برای کمک به درک بهتر این موضوعات و ترکیبات لازم است تعاریف مفیدی در این زمینه را مطرح کنیم.

چابک مخلوط، ترکیبی از دو یا چند روش، تکنیک و چارچوب تثبیت‌شده چابک است. مثلاً افزودن مفاهیم کانبان و کار در جریان به اسپرینت‌ها می‌تواند نمونه‌ای از این کار باشد. این یک روش شناخته‌شده است که ما سعی می‌کنیم به کمک ترکیب چنین تکنیک‌هایی در انجام کار و توسعه پروژه بهتر عمل کنیم. پس در قالب یک فرمول ساده می‌توان گفت:

Blended = Agile + Agile = Better Agile



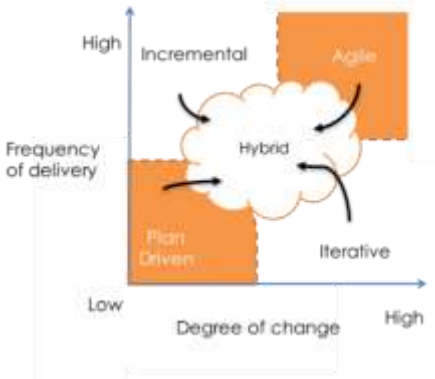
تصویر ۴-۲ : چابک مخلوط

اما موقعیتی را در نظر بگیرید که در آن نمی‌توان صرفاً از مجموعه‌ای از روش‌های چابک بهره برد. در چنین مواقعی که محدودیت‌ها و یا خواسته‌هایی وجود دارد که نیاز به وقوع و به‌کارگیری عناصر غیر چابک باشد چه می‌توان کرد؟ در چنین مواردی باید رویکرد ترکیبی را بکار بست.

براین‌اساس چابک ترکیبی یا هیبریدی، ترکیبی از روش‌های چابک با سایر تکنیک‌های غیر چابک است. به‌عنوان‌مثال، یک تلاش دقیق و پر از جزئیات برای مرحله الزامات و مهندسی نیازمندی‌ها که به همراه یک اسپرینت و نهایتاً تحویل تدریجی و افزایشی می‌آید را می‌توان یک رویکرد ترکیبی دانست. به همین ترتیب، نمونه‌سازی مکرر یک طرح که به دنبال یک رویکرد برنامه-محور اجرا می‌شود را نیز می‌توان یک رویکرد ترکیبی یا هیبریدی دانست. به طور خاص ایده این است که اگرچه شما یک رویکرد غیر چابک دارید اما شاید بتوان برخی از تکنیک‌های چابک را برای پاسخ به مسائل خاص در داخل آن تزریق کرد.

در یک فرمول ساده می‌توان این رویکرد را به شکل زیر در نظر گرفت:

Hybrid = non-Agile + Agile = something in between that makes sense



تصویر ۴-۳ : روش ترکیبی

درست مثل هر چیز دیگری در این جهان، یک دلیل درست و یک دلیل غلط برای انجام هر کاری وجود دارد. اگر بخواهیم صادق باشیم، دلیل غلط مخلوط کردن تکنیک‌های مختلف عقب نیفتادن از سایر گروه‌ها و موضوعات جدید است. اما باید دقت کرد که انجام تکنیک‌های چابک هدف نیستند. هدف ارائه یک خروجی صحیح از ارزش‌های کسب‌وکار به کمک تکنیک‌های مناسب است.

اگر چرخه‌های عمر پروژه‌های مختلف را در یک پیوستار بررسی کنیم، از یک طرف توسعه برنامه-محور و در سوی دیگر به چابک می‌رسیم. برای درک بهتر این پیوستار دو جنبه اصلی چابکی یعنی، تحویل زودهنگام که غالباً صورت می‌گیرد و سازگار شدن با تغییر را در نظر می‌گیریم. اگر بخواهیم آن را بر روی یک نمودار دو بعدی ترسیم کنیم، چیزی مانند شکل ذیل را خواهیم داشت.

اما برای بکارگیری چنین رویکردهایی و حرکت سازمان به سمت چابکی دو سناریوی عمده وجود دارد که تیم‌ها را به استفاده از روش‌های ترکیبی ترغیب می‌کند:

- متناسب با هدف: برای پروژه‌های با ریسک کمتر روش‌های برنامه-محور استفاده می‌شوند و هزینه کمتری را ایجاد می‌کنند. برای پروژه‌های با ریسک بالاتر از تکنیک‌های تکرارشونده در جهت تکرار فعالیت‌ها استفاده می‌شود تا به مرور مسائل آشکار و حل شوند. برای پروژه‌هایی که نیاز به تحویل با سرعت بالا دارند، تکنیک‌های افزایشی برای اطمینان از تعامل و درگیری مشتری مناسب‌ترند. برای محیط‌های پیچیده ممکن است چابک سربار بالاتری داشته باشد اما در یک دید کلی و جامع نتایج بهتری را می‌تواند فراهم آورد. در نهایت می‌پذیریم که هر کدام قدرت و مزایای خاص خود را دارند. مخلوط کردن اینها با هم آن هم به روش صحیح و بر اساس بافت پروژه و سازمان کمک کننده خواهد بود.
- انتقال به چابک: بسیاری از تیم‌ها قادر نیستند یک شبه به روش‌های چابک روی بیاورند. هر چه سازمان بزرگ‌تر باشد، قطعات متحرک بیشتری دارد و مدت زمان بیشتری برای جا به جایی لازم است. اگر سازمان و یا تیم توسعه چندین سال در دنیای مبتنی بر روش برنامه-محور زندگی کرده باشد، روش‌های چابک برایش بسیار متفاوت به نظر می‌رسد. در نتیجه ورود اولیه به دنیای چابک یک ادغام نه چندان تمیز و صحیح خواهد بود. اگرچه این مسئله پذیرفته شده است اما با استفاده از روش‌های ترکیبی به شکل مناسب این انتقال را در جهت مناسب سوق می‌دهد.

مسئله این است که به سادگی اعلام نکنید، ما چابک هستیم، واقعیت این است که شما تقریباً همیشه از ترکیبی از تکنیک‌های مختلف استفاده می‌کنید. در عوض، یک استراتژی بهتر این است که متوقف شویم و فکر کنیم که کدام روش برای مکان ما بهترین است و چه چیزی را می‌خواهیم به دست آوریم. اختلاف نظرهای شدیدی بین واژه‌های مخلوط، ترکیبی و هیبریدی وجود دارد. با این حال برای ساده کردن بحث آنچه ما به آن خواهیم پرداخت ترکیبی از متدولوژی‌های چابک خواهد بود که از مزایای روش‌های سنتی و آبشاری نیز در موقعیت مناسب بهره خواهد برد. تفکر چابک و تفکر ناب را نیز ترکیب خواهیم کرد و همچنان از محاسن هر روش سود خواهیم برد.

در سال ۲۰۱۴ شرکت اچ‌پی پژوهشی را در بین ۴۰۰ متخصص در این زمینه انجام داد و نهایتاً اعلام کرد روش‌هایی که رویکردهای سنتی و آبشاری را با تفکر چابک در هم مخلوط می‌کنند به سود کمتری دست می‌یابند و در مسیر خود موفقیت کمتری را تجربه می‌کنند. فرهنگ سازمانی و درون تیمی هم نقش عمده‌ای دارد. در رویکردهای سنتی افراد بر تهیه مستندات و تهیه یک چشم‌انداز کلی در همان ابتدای کار تأکید دارند، چیزی که در روش‌های چابک توصیه نمی‌شود. طبق این تحقیق، تیم‌ها به دنبال کسب

مزایای بیشتری هستند اما نمی‌خواهند روش توسعه سنتی و فرهنگ جا افتاده در سازمان را رها کنند. آنها به دنبال راضی کردن مشتریان هستند، اما تحویل مداوم، حلقه‌های بازخورد کوتاه و پذیرش تغییرات را اجرا نمی‌کنند. سازمان به دنبال تیم‌های بهره‌ور می‌باشد اما حقیقت تیم‌های خودسازمانده و فرا وظیفه‌ای را نادیده می‌گیرد.

طبق نظرسنجی سالیانه شرکت دیجیتال ای‌آی که با نام گزارش وضعیت چابک منتشر می‌شود، در سال ۲۰۲۰، اسکرام با ۵۸ درصد پیشتاز است. پس از اسکرام، اسکرام‌بان با سهم ۱۰ درصدی به چشم می‌خورد. در این میان ۴۳ درصد عنوان کرده‌اند از روش‌های ترکیبی چابک و سنتی بهره می‌برند. ۲۸ درصد اعلام کرده‌اند در پروژه‌هایی به طور مجزا و خاص از چابک و در سایر پروژه‌ها از روش‌های سنتی بهره برده‌اند. ۲۰ درصد به طور خالص از چابک بهره برده‌اند و ۹ درصد هنوز فقط از روش‌های سنتی استفاده می‌کنند.

درست مانند استفاده از افراد یا ابزار مناسب برای کار، استفاده از بهترین روش توسعه نرم‌افزار و مدیریت پروژه برای تحویل بهنگام و با کیفیت به مشتریان مسئله‌ای کلیدی است. با این حال یک رویکرد واحد که برای هر موقعیتی مناسب باشد فراهم نیست و باید بسته به موقعیت از بین متدولوژی‌های موجود دست به انتخاب زد.

امروزه در همه جا از ناکارآمدی روش آبشاری در صنعت توسعه نرم‌افزار و مدیریت پروژه‌های پیچیده شنیده می‌شود. در سال ۱۹۷۰ آقای وینستون رویس مقاله‌ای را منتشر کرد و در آن تجربیات خود در زمینه توسعه نرم‌افزارهای بزرگ و پیچیده ناسا را توضیح داد. در ابتدای مقاله به کمک یک شکل مراحل تولید را بررسی کرد، این شکل مراحلی مانند: مهندسی نیازمندی‌ها، تحلیل، طراحی، کدنویسی و تست را در برمی‌گرفت و حالت آبشاری در نمایش داشت. در ادامه مقاله رویس اشاره می‌کند که این روش برای توسعه نرم‌افزارهای بزرگ و پیچیده مناسب نمی‌باشد و بهتر است از روش‌های چرخشی مبتنی بر پروتوتایپ استفاده شود. عموم خوانندگان آن مقاله از سر بی دقتی تنها به نمودار موجود در مقاله توجه کرده بودند و آن را تأییدی بر روش آبشاری از سوی آقای رویس دانسته بودند. در سال ۱۹۸۵ وزارت دفاع امریکا به دنبال یک روش استاندارد برای توسعه پروژه‌های خود بود. با استناد به عکس مقاله آقای رویس همین روش آبشاری را به عنوان یک استاندارد برای پروژه‌ها برگزید. به دلیل سادگی طرح به سرعت مورد استقبال عمومی جامعه توسعه‌دهندگان قرار گرفت. گرچه وزارت دفاع بعدها این اشتباه را تصحیح کرد اما به دلیل گستردگی ایده و اقبال عمومی کار از کار گذشته بود.

۴ – ۵ روش اسکرام‌بان‌فال

اسکرامبان‌فال[1] ترکیب چابکی از اسکرام و کانبان به همراه روش آبشاری در مدیریت مهندسی نرم‌افزار است. روش‌های چابک مانند یک سقف برای شیوه‌های توسعه نرم‌افزار هستند که بر اساس بیانیه چابک ساخته شده‌اند. رویکرد چابک نحوه تفکر در مورد چشم‌انداز محصول را از نظر ارزش‌های مشتری، کیفیت محصول، حداکثرسازی بهره‌وری و نقش تیم نرم‌افزار در توسعه محصول را تغییر می‌دهد. اسکرام و کانبان همه این مفاهیم و ارزش‌ها را برای توسعه نرم‌افزارهای سازمانی تحت پوشش قرار می‌دهند. اسکرام و کانبان هر دو چابک هستند و ویژگی‌های خود را از بیانیه چابک مشتق کرده‌اند. اسکرام یک چارچوب محبوب و فراگیر در میان همه روش‌های مدیریت مهندسی نرم‌افزار چابک است. اسکرام بر اساس ویژگی‌های خود شامل: مصنوعات، رویدادها، ستون‌ها،

[1] Scrumbanfall

نقش‌ها و ارزش‌ها باعث تغییر در شکل توسعه نرم‌افزار در سازمان‌ها با فعال‌کردن قابلیت‌های مهندسی مجدد فرایند کسب‌وکار چابک شده است. اسکرام یک روش نسل اول و انتخاب اول اکثر تیم‌ها بوده است. درحالی‌که کانبان روش نسل دومی و گزینه دوم در میان خانواده چابک است و هر دو به شکل گسترده‌ای در سازمان‌های توسعه نرم‌افزار استفاده می‌شوند. همه عوامل اساسی را باید در هنگام انتخاب چارچوبی به‌عنوان یک روش استاندارد برای توسعه نرم‌افزار و مدیریت پروژه در نظر گرفت. اسکرام با چالش‌هایی روبرو است: مدیریت زمان تدارک، کنترل جریان کار، مشارکت مستقیم ذی‌نفعان خارجی به‌عنوان تصمیم گیرندگان مهم، اندازه تیم و نقش‌های مشخص در آن، قابلیت‌های توسعه نرم‌افزار مقیاس بزرگ در محیط‌های توزیع شده و چشم‌انداز نامشخص محصول در مرحله اولیه پروژه.

اسکرامبان، می‌تواند بر تعدادی از این چالش‌ها از جمله: کنترل جریان کار، مدیریت زمان تدارک، تحویل مداوم و یکپارچه غلبه کند. بااین‌وجود اسکرامبان نمی‌تواند همه چالش‌ها را پوشش دهد. یک متدولوژی نوآورانه مانند اسکرامبان‌فال می‌تواند در حل برخی از مسائل پروژه یاری‌دهنده باشد: مستندسازی نیازمندی‌های پروژه، برنامه‌ریزی، تخمین اولیه، چشم‌انداز واضح محصول در فازهای آغازین شروع پروژه. برآورد توسعه نرم‌افزار، هزینه و محدودیت زمانی برای ساخت نرم‌افزاری با کیفیت شامل نیازمندی‌های عملکردی و غیر عملکردی را تضمین می‌کند. مهندسی مجدد فرایند کسب‌وکار چابک به چارچوب چرخه عمر فرایند مناسبی نیاز دارد که متناسب با ساختار سازمانی از قدرت فناوری‌های اتوماسیون در زمینه هوش مصنوعی و یادگیری ماشین برای مدیریت پروژه‌های نرم‌افزاری بهره ببرد.

۴ – ۶ مهندسی متدولوژی

الگوهای فرایند، تکه‌های قابل‌استفاده مجدد هستند که از فرایندهای مختلف ایجاد نرم‌افزار بیرون کشیده شده‌اند و برای حل یک مشکل در فرایند متدولوژی، راه‌حل خوبی ارائه می‌دهند. الگوی فرایند توسط جیم کاپلین[1] در سال ۱۹۹۴ مطرح شد. کاپلین الگوی فرایندی را با نام «الگوی فعالیت در سازمان و پروژه‌های آن» تعریف کرد. از مشکلات عمده چنین الگویی می‌توان به ریزدانه بودن و تمرکز بیشتر بر جنبه‌های مدیریتی نسبت به فرایندهای نرم‌افزاری اشاره کرد. الگوهای شی گرای فرایند توسط اسکات امبلر[2] معرفی شد. امبلر، با کنار هم قراردادن فازها، فرایند کلی ایجاد نرم‌افزار[3] را تعریف کرد. بر همین اساس متدولوژی EUP را به‌عنوان گسترشی بر RUP ارائه کرد. مشکل عمده عناصر فرایند OOSP وابستگی زیاد بین آنها بود. به این صورت که نمی‌توان از هر یک از آنها به‌عنوان یک تکه فعالیت جداگانه استفاده کرد. در سال ۱۹۹۶ متدولوژی OPEN[4] به عنوان یک متدولوژی نسل سومی سنگین وزن از تجمیع چهار متدولوژی ایجاد شد. به دلیل سنگین بودن این متدولوژی سازندگان آن تصمیم گرفتند به جای اینکه یک متدولوژی جامع را ارائه کنند، یک چارچوب ایجاد فرایند به نام OPF و یک کتابخانه از مؤلفه‌های متدولوژی به نام OPFRO ارائه کنند. به این ترتیب هر فرد یا سازمان، متدولوژی خود را با انتخاب مؤلفه‌ها از کتابخانه و تجمیع آنها طبق چارچوب ایجاد فرایند خواهد ساخت.

[1] Jim Coplien
[2] Scott Ambler
[3] OOSP: Object-oriented Software Process
[4] Object-oriented Process, Environment and Notation

۴ – ۷ متدولوژی پیشنهادی این پژوهش

در این پژوهش به جای چارچوب مهندسی متدولوژی OPF از چارچوب فرایند اکلیپس[1] بهره برده شده است. این چارچوب انواع فعالیت‌ها، نقش‌ها و محصولات را به عنوان تکه‌های متدولوژی در کتابخانه خود دارد که از اتصال آنها می‌توانیم متدولوژی دلخواه مطابق نیازمندی‌های سازمان مربوطه را بسازیم. برای این منظور از EPFC[2] بهره برده شده است. نتیجه کار در گیت‌هاب جهت استفاده و گسترش سایر پژوهشگران قرار داده شده است.[3] در مرحله نخست، بر اساس پلاگین‌های برنامه‌نویسی مفرط و اسکرام روش کانبان به عنوان یک پلاگین اضافه شد. سپس متدولوژی اسکرامبان اضافه گردید و بر اساس نقش‌ها، فعالیت‌ها و محصولات این روش‌ها متدولوژی اسکرامبان‌فال ارائه گردیده است. یک آموزش مقدماتی برای شروع کار با محیط EPFC نیز در این آدرس[4] به صورت چندرسانه‌ای ارائه شده است.

[1] EPF: Eclipse Process Framework
[2] EPF Composer
[3] https://github.com/mohammadkad/MKE
[4] https://www.aparat.com/v/MKhAm

فصـــل پنجم

جمع‌بندی و پیشنهادها

مقدمه

برای به‌دست‌آوردن یک دید بهتر از وضعیت فعلی دنیای چابک، باید نگاهی به آمارها داشته باشیم، نظرسنجی‌های سالیانه شرکت کولب‌نت[1] داشته باشیم. در زمان تهیه این فصل، گزارش دوره پانزدهم این نظرسنجی منتشر نشده است، بنابراین به نتایج بین سال‌های ۲۰۱۶ تا ۲۰۲۰ نگاه خواهیم کرد. سیزدهم اشاره خواهد شد. اما پیش از آن بر اساس آمار ارائه شده در گزارش Chaos که به مقایسه روش‌های سنتی و چابک می‌پردازد نرخ موفقیت و شکست این رویکردها را بررسی خواهیم کرد (CHAOS Report, 2020).

جدول ۵–۱ : درصد موفقیت و شکست پروژه‌های چابک و آبشاری

سال	۲۰۱۵			۲۰۲۰		
نتیجه	شکست	با اشکال	موفقیت	شکست	با اشکال	موفقیت
چابک	۹٪	۵۲٪	۳۹٪	۹٪	۴۹٪	۴۲٪
آبشاری	۲۹٪	۶۰٪	۱۱٪	۲۹٪	۵۷٪	۱۴٪

به‌کارگیری رویکرد چابک به خوبی نتایج بهتری نسبت به روش‌های سنتی در طول این سال‌ها برای سازمان‌ها و تیم‌های توسعه به دنبال داشته است. اما برای تشخیص اینکه کدام روش‌های چابک محبوبیت بیشتری دارند به جدول ۵-۲ نگاه کنید (State of Agile Report, 2020).

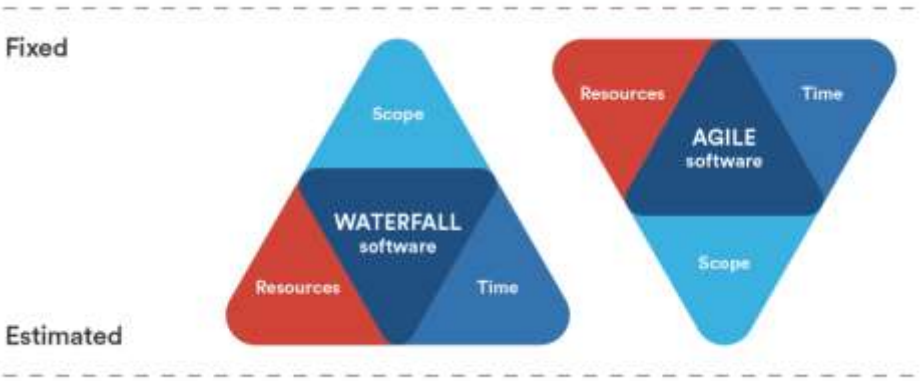
جدول ۵–۲ : درصد استفاده از روش‌های رویکرد چابک در توسعه نرم‌افزار

سال / روش	۲۰۱۶	۲۰۱۷	۲۰۱۸	۲۰۱۹	۲۰۲۰
اسکرام	۵۸٪	۵۸٪	۵۶٪	۵۴٪	۵۸٪
اسکرام و مفرط	۱۰٪	۱۰٪	۶٪	۱۰٪	۸٪
کانبان	۵٪	۵٪	۵٪	۵٪	۷٪
اسکرام‌بان	۷٪	۸٪	۸٪	۸٪	۱۰٪

اسکرام همچنان به عنوان پرکاربردترین متدولوژی چابک معرفی شده است. به طور مشخص ۷۲ درصد شرکت‌کنندگان اعلام کرده‌اند از اسکرام و یا روش‌های ترکیبی که شامل اسکرام بوده برای توسعه پروژه‌های خود بهره می‌برند. بر همین اساس همچنان مسئله فرهنگ سازمانی اصلی‌ترین مانع بر سر پذیرش و مقیاس‌پذیر کردن چابکی در سازمان است. مقاومت عمومی برای تغییر، حمایت و پشتیبانی ناکافی مدیریت و فرهنگ سازمانی که در مغایرت با ارزش‌های چابکی هستند همچنان موانع اصلی در این زمینه‌اند. برای موفقیت در زمینه چابکی سه عامل: مربیان چابک داخلی، حمایت مالی و برنامه‌های آموزشی ارائه شده توسط شرکت بیشترین اهمیت را داشته‌اند.

[1] Colabnet

مسئله مهم دیگر دراین‌بین توجه به مثلث مدیریت پروژه[1]است. هر پروژه دارای سه قید زمان[2]، هزینه[3]و محدوده[4]است. کیفیت در میان این مثلث قرار دارد. مدیر پروژه باید بر اساس بودجه و زمانی که در اختیار دارد با دقت محدوده کار را مشخص نماید. متدولوژی و فرایندهای انتخابی در این مسیر نقش اساسی دارند. ازآنجایی‌که این سه قید در رئوس یک مثلث قرار دارند تغییر در هرکدام بر عوامل دیگر تأثیرگذار خواهد بود.



تصویر ۵-۱ : مقایسه رویکرد چابک و سنتی بر اساس مثلث مدیریت پروژه

۵ – ۱ نتایج حاصل از تحقیق

به نظر می‌رسد گرچه پذیرش و حمایت برای چابک شدن در حال گسترش است، اما تمام تیم‌های یک سازمان به طور کامل چابکی را نپذیرفته‌اند. از دلایل عمده پذیرش چابکی می‌توان به مواردی چون: تسریع در تحویل نرم‌افزار، افزایش توانایی در مدیریت تغییر اولویت‌ها، افزایش بهره‌وری، بهبود همکاری کسب‌وکار و فناوری اطلاعات، بهبود کیفیت نرم‌افزار، بهبود پیش‌بینی تحویل، بهبود شفافیت پروژه، کاهش هزینه پروژه، بهبود روحیه تیم، کاهش ریسک پروژه، بهبود رشته مهندسی، افزایش قابلیت نگهداری نرم‌افزار و مدیریت بهتر تیم‌های توزیع‌شده اشاره کرد. بر همین اساس تنها ۲۲ درصد سازمان‌ها دارای تیم‌های کاملاً چابک هستند. ۲۶ درصد اعلام کرده‌اند که بیشتر از نصف تیم‌ها چابک هستند. ۴۸ درصد کمتر از نیمی از تیم‌هایشان چابک هستند. در این راستا فقدان آموزش و تجربه، فرهنگ سازمانی متفاوت با تفکر چابک، نادیده‌گرفتن مدیران سازمان در فرایند تبدیل، متناسب نبودن ابزارها با پروسه تبدیل و نداشتن مجموعه‌ای از سنجه‌ها برای رصد میزان پیشرفت در چابک شدن، از چالش‌های عمده در حرکت یک سازمان به سمت چابک شدن به‌حساب می‌آیند. موقعیت هر سازمان و نفرات آن منحصربه‌فرد است. در جهانی که تحویل به‌موقع نیاز مشتری ملاک مهمی به شمار می‌رود، سازمان‌ها با انتخاب هوشمندانه یک روش چابک می‌توانند از مشکلات موجود در این مسیر با هزینه کمتری عبور کنند. اسکرام‌بان با بهره‌گیری از مزایای اسکرام و کانبان و بدون تحمیل تغییرات بر سازمان سنتی موجود یکی از بهترین انتخاب‌ها خواهد بود. اما برای تمرکز بر مهندسی نیازمندی‌ها، تهیه

[1] Project management triangle
[2] Time
[3] Cost
[4] Scope

مستندات با جزئیات و تعریف فازهای نگهداری متدولوژی اسکرامبان‌فال را نیز می‌توان در نظر گرفت (State of Agile Report, 2020).

هر سازمان، چشم‌اندازی از خواسته‌ها و نیازهای خود دارد. چارچوب‌هایی چون اسکرام و روش‌هایی چون اسکرام‌بان و اسکرامبان‌فال می‌تواند به سازمان‌ها در مدیریت کارها برای دستیابی به این چشم‌انداز کمک کند. اما افزایش مهارت در این روش‌ها و در نتیجه افزایش چابکی، هدف نهایی نیست بلکه ابزاری مؤثر و مقرون‌به‌صرفه برای دستیابی به اهداف کسب‌وکار است. به طور مثال برخلاف مدل بلوغ قابلیت یکپارچه[1] که هدف نهایی رسیدن به «سطح پنجم» مدل بلوغ است در اسکرام وضعیت پایانی وجود ندارد. در تفکر چابک و ناب فلسفه نوعی بهبود مستمر است که با تعیین یک سطح نهایی در تضاد است. بر همین اساس مایک کوهن معتقد است جمله «بالاخره به چابکی رسیدیم!» نادرست است.

براین‌اساس، تعریف یک چشم‌انداز، تهیه نقشه راه محصول، ایجاد یک برنامهٔ تحویل نتایج، برنامه‌ریزی برای اسپرینت‌ها و تهیه تابلوی کانبان، جلسات ایستاده روزانه، بازبینی و بازنگری بر کارهای انجام شده در کنار ارزش‌هایی چون شجاعت، تمرکز، تعهد، احترام و باز بودن در یک تفکر تدریجی - افزایشی اثرات خود در توسعه نرم‌افزار را به‌خوبی به اثبات رسانده است. افزایش مهارت در این روش‌ها و داشتن چابکی، فرایند بهبود مستمر و بی‌پایانی است که هدف آن، افزایش سودآوری در کنار رضایت مشتری است.

در فولاد سبا این مسیر با بکارگیری و پررنگ کردن مزایای کانبان در متدولوژی اسکرام‌بان در پروژه‌های سطح دو فولاد سبا شروع شد و نتایج قابل قبولی در کاهش زمان تحویل خواسته‌ها به همراه داشت.

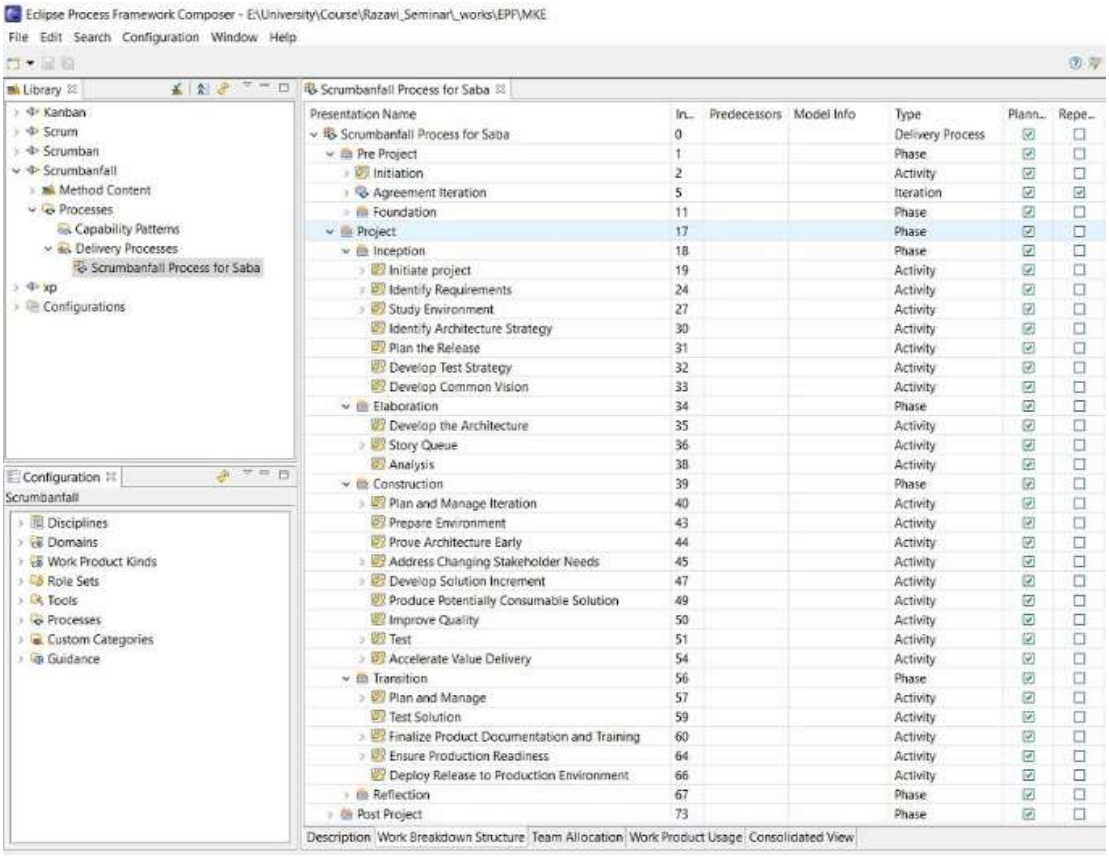
جدول ۵–۳ : نتایج بکارگیری متدولوژی اسکرام‌بان در پروژه توسعه خط دو فولاد سبا

	نمایش پیشرفت کار	میانگین زمان تدارک[2]	میانگین زمان چرخه[3]	درصد مطابقت با درخواست صادره	تعداد کار در جریان
آبشاری و سنتی	گانت چارت، عدم تطبیق نقاط عطف برنامه‌ریزی شده با واقعیت موجود	۸ تا ۱۰ هفته	۵ هفته	٪۶۵	متغیر
چابک (اسکرام‌بان)	تابلوی کانبان، منطبق با وضعیت جاری پروژه در حال اجرا	۴ هفته	۲ هفته	٪۸۵	۳

با این حال پس از مدتی با بررسی بازخوردها مشخص شد، اسکرامبان تنها می‌تواند بر تعدادی از این چالش‌ها از جمله: کنترل جریان کار، مدیریت زمان تدارک، تحویل مداوم و یکپارچه غلبه کند. اما اسکرامبان نمی‌تواند همه چالش‌ها را پوشش دهد. یک متدولوژی نوآورانه مانند اسکرامبان‌فال می‌تواند در حل برخی از مسائل پروژه از جمله: مستندسازی نیازمندی‌های پروژه،

[1] CMMI
[2] Lead time
[3] Cycle time

برنامه‌ریزی، تخمین اولیه، چشم‌انداز واضح محصول در فازهای آغازین شروع پروژه به‌خصوص برای سازمان‌هایی که حرکت آرامی به سمت چابکی دارند یاری‌دهنده باشد.



تصویر ۵-۲ : متدولوژی اسکرامبان‌فال ارائه‌شده

براین‌اساس، تعریف یک چشم‌انداز، تهیه نقشه راه محصول، ایجاد یک برنامهٔ تحویل نتایج، برنامه‌ریزی برای اسپرینت‌ها و تهیه تابلوی کانبان، جلسات ایستاده روزانه، بازبینی و بازنگری بر کارهای انجام شده در کنار ارزش‌هایی چون شجاعت، تمرکز، تعهد، احترام و باز بودن در یک تفکر تدریجی - افزایشی اثرات خود در توسعه نرم‌افزار را به‌خوبی به اثبات رسانده است. افزایش مهارت در این روش‌ها و داشتن چابکی، فرایند بهبود مستمر و بی‌پایانی است که هدف آن، افزایش سودآوری در کنار رضایت مشتری است.

۵ – ۲ پیشنهادها

برای گسترش موضوع و کارهای آتی موضوعات ذیل را می‌توان مطرح کرد:

- سازمان‌ها، صنایع بزرگ و سنتی ایران و چالش‌ها برای متقاعدکردن مدیران و کارکنان برای پذیرش چابکی.
- منطبق سازی تفکر چابک با الگوی تعالی سازمانی.
- اندازه‌گیری میزان مؤثر بودن روش‌های ترکیبی به کمک متریک‌های چابک و تفکر ناب
- به‌کارگیری متدولوژی اسکرامبان‌فال ارائه شده در این تحقیق برای پروژه‌های اتوماسیونی صنایع فولاد و بررسی بازخوردها

مراجع

مراجع

بی‌بالان، یوسف، اسماعیلی، علیرضا. (۱۳۹۷). «**اصول و روش کاربردی اسکرام**»، تهران: انتشارات صفار.

خاتمی‌نژاد، داود. (۱۳۹۵). «**توسعه چابک نرم‌افزار**»، تهران: انتشارات دانشگاهی کیان.

شمس، فریدون، راضی، علی، (۱۳۸۶). «**ضرورت بکارگیری ایده چابکی در معماری سازمانی**»، چهارمین کنفرانس بین‌المللی فناوری اطلاعات و ارتباطات، تهران.

گلشاهی، فرناز، جاودانی گندمانی، تقی، (۱۳۹۷). «**بررسی و مقایسه روش‌های ترکیبی چابک با تمرکز بر تفکر ناب و روش اسکرام در توسعه محصول نرم‌افزاری**»، سومین کنفرانس آخرین دستاوردهای علمی در حوزه مهندسی کامپیوتر پردازش نرم و تکنولوژی‌های نوین پردازشی، بروجن، دانشگاه آزاد اسلامی واحد بروجن.

گلشاهی، فرناز، جاودانی گندمانی، تقی، آقایی، گلناز، (۱۳۹۹). «**یک مدل ترکیبی اسکرام و ناب به‌منظور افزایش راندمان تیم توسعه و کاهش هزینه توسعه محصول نرم‌افزاری**»، نهمین کنفرانس بین‌المللی فناوری اطلاعات، کامپیوتر و مخابرات.

مولاناپور، رامین، پورنادر، مهردخت. (۱۳۹۱). «**روش‌شناسی ایجاد سیستم‌های اطلاعاتی**»، تهران: انتشارات آتی‌نگر.

Ahmad, Muhammad Ovais, Dennehy, Denis, Conboy, Kieran, "***Kanban in software engineering: A systematic mapping study***", Journal of Systems and Software, Vol. 137, No. 1, PP. 96-113, 2018.

Albarqi, Aysha Abdullah, Qureshi, Rizwan, "***The Proposed L-Scrumban Methodology to Improve the Efficiency of Agile Software Development***", I.J. Information Engineering and Electronic Business, Vol. 10, No. 3, PP. 23-35, 2018.

Alsaqqa, Samar, Sawalha, Samer, "***Agile Software Development: Methodologies and Trends***", International Journal of Interactive Mobile Technologies, Vol. 14, No. 11, PP. 23-35, 2020.

Alsaqqa, Samar, Sawalha, Samer, "***Agile Software Development: Methodologies and Trends***", International Journal of Interactive Mobile Technologies, Vol. 14, No. 11, PP. 23-35, 2020.

Beck, Kent, "Embracing change with extreme programming", International Journal of Computer (IEEE), Vol. 32, No. 10, PP. 70-77, 1999.

Bhavsar, Krunal, Gopalan, Samir, Shah, Vrutik, "***Scrumbanfall: an agile integration of scrum and kanban with waterfall in software engineering***", International Journal of Innovative Technology and Exploring Engineering (IJITEE), Vol. 9, No. 4, PP. 2075-2084, 2020.

CHAOS Report, "***CHAOS Report 2020***", 2021 by The Standish Group International, Inc, web site: https://www.standishgroup.

Choras, Michal, Springer, Tomasz, Kozik, Rafal, "***Measuring and improving agile processes in a small-size software development company***", IEEE, Vol. 8, PP. 78452- 78466, 2020.

Jovanovic, Milos, Mesquida, Antoniliuis, Mas, Antonia, "***Agile transition and adoption frameworks, issues and factors: a systematic mapping***", IEEE Access, Vol. 8, PP. 15711- 15735, 2020.

Kumar, Rakesh, Maheshwary, Priti, Malche, Timothy, "***Inside agile family software development methodologies***", International Journal of Computer Sciences and Engineering, Vol. 7, No. 6, PP. 650-660, 2019.

Neelu, Lalband, Kavitha, D, "***Software Development Technique for the Betterment of End User Satisfaction using Agile Methodology***", TEM Journal UIKTEN-Association for Information Communication Technology Education, Vol. 9, No. 3, PP. 992-1003, 2020.

Nikitina, Natalja, Kajko-Mattsson, Mira, "***From scrum to scrumban: A case study of a process transition***", 2012 International Conference on Software and System Process (ICSSP), PP. 140-149, 2012.

O'Connor, Rory, Lepmets, Marion, "***Exploring the use of the cynefin framework to inform software development approach decisions***", Proceedings of the 2015 International Conference on Software and System Process, PP. 97-101, 2015.

Pressman, Roger S, "***Software Engineering: A Practitioner's Approach***" 6th Edition, McGraw-Hill, 2005.

Saleh, Sabbir, Huq, Syed Maruful, Rahman, Ashikur, "***Comparative study within Scrum, Kanban, XP focused on their practices***", 2019 International Conference on Electrical, Computer and Communication Engineering (ECCE), PP. 1-6, 2019.

State of Agile Report, "**14th Annual State of Agile Report**", 2021 Digital.ai, web site:https://stateofagile.com.

Zayat, Wael, Senvar, Ozlem, "***Framework Study for Agile Software Development Via Scrum and Kanban***", International Journal of Innovation and Technology Management (IJITM), Vol. 17, No. 4, PP. 1-24, 2020.

واژه‌نامه

واژه‌نامه فارسی‌به‌انگلیسی

احترام	Respect
ارزش	Value
اسپرینت	Sprint
اسکرام	Scrum
اسکرام‌بان	Scrumban
آزمون	Test
باز بودن	Openness
بازبینی	Review
بازنگری	Retrospective
تجویزی	Prescriptive
تطبیقی	Adaptive
تعهد	Commitment
تفکر سیستمی	System thinking
تمرکز	Focus
توسعه	Development
چابک	Agile
چارچوب	Framework
چرخه حیات	Life cycle
چشم‌انداز	Vision
رفتار	Behavior
روش‌شناسی	Methodology
شجاعت	Courage
فرایند	Process
قوانین	Rules
کانبان	Kanban
کریستال	Crystal
مدیریت پروژه	Project Management
ناب	Lean
نرم‌افزار	Software
نقش‌ها	Role

واژه‌نامه انگلیسی‌به‌فارسی

Adaptive	تطبیقی
Agile	چابک
Behavior	رفتار
Commitment	تعهد
Courage	شجاعت
Crystal	کریستال
Development	توسعه
Focus	تمرکز
Framework	چارچوب
Kanban	کانبان
Lean	ناب
Life cycle	چرخه حیات
Methodology	روش‌شناسی
Openness	باز بودن
Prescriptive	تجویزی
Process	فرایند
Project Management	مدیریت پروژه
Respect	احترام
Retrospective	بازنگری
Review	بازبینی
Role	نقش‌ها
Rules	قوانین
Scrum	اسکرام
Scrumban	اسکرام‌بان
Software	نرم‌افزار
Sprint	اسپرینت
System thinking	تفکر سیستمی
Test	آزمون
Value	ارزش
Vision	چشم‌انداز

Abstract

When we start a software project, choosing the right development model increases the success rate of that project. The right model helps the project to be completed on time with the budget in accordance with the needs of the organization. However, there is no fixed version for this. During these years, the software life cycle has shifted from repetitive and traditional models to incremental adaptive patterns in the form of self-organized and cross-functional teams. Agile team is able to respond to changes appropriately and in a timely manner and attract customer satisfaction.
In this research, in the first step, with the help of Cynefin decision-making framework, the characteristics of the development environment in Saba Steel Automation Project have been identified. Then, the advantages of using Scrum framework and Kanban method for this environment have been investigated. Scrumbanfall methodology has been selected as a software development model after the successful results of using Scrumban methodology. The reasons for such a choice and the steps taken are explained in this study.

Keywords

Agile, Hybrid Methodology, Scrumbanfall, Software Development, Enterprise architecture

Payam Noor University

Faculty of Engineering

Seminar Report

Department of Computer Engineering and Information Technology

Hybrid Methodologies in The Software Development in Large Organizations and Industries

(Saba Steel Case study)

Mohammad Kadkhodaei Elyaderani

Supervisor:

Dr. Seyed Ali Razavi Ebrahimi

February 2021