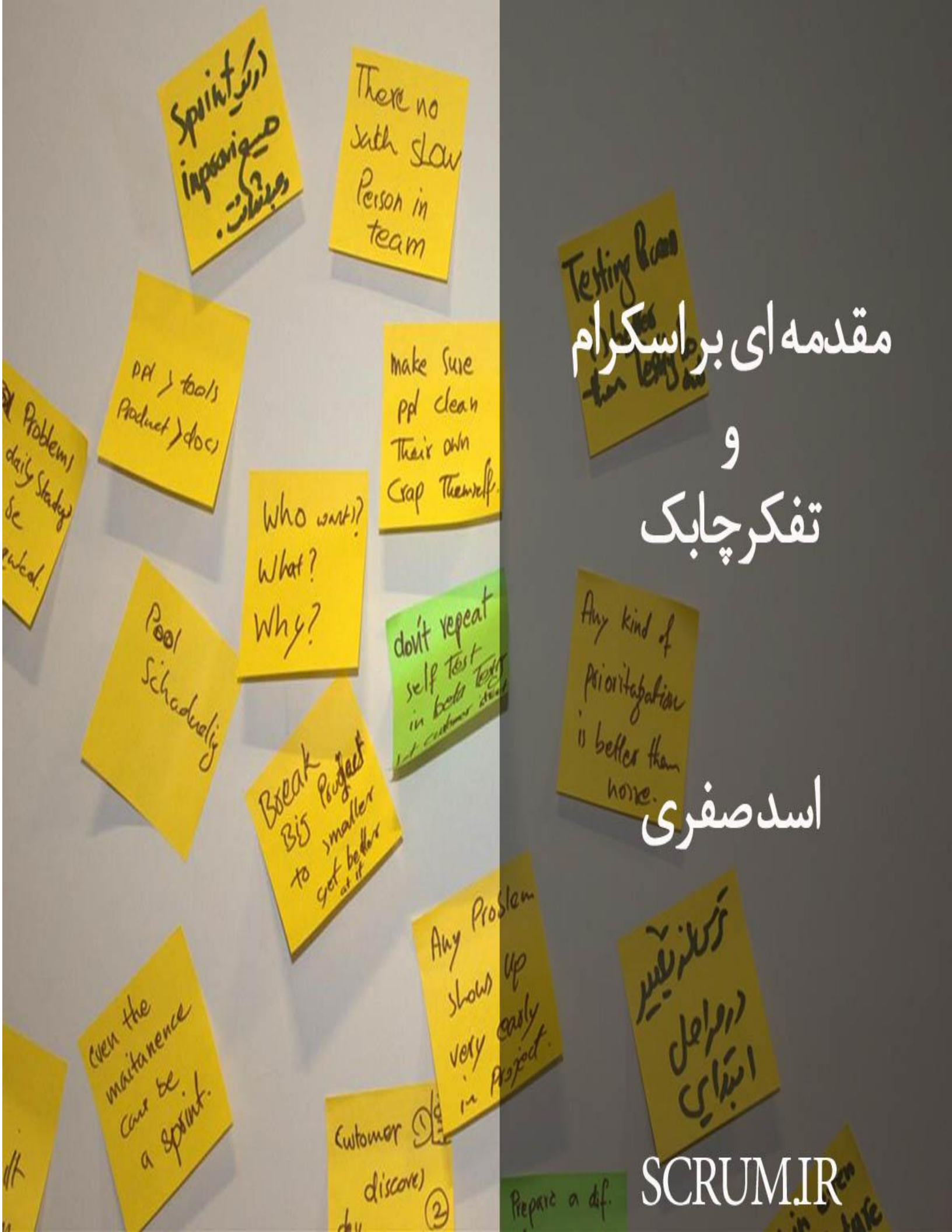


مقدمه ای بر اسکرام و تفکر چابک

اسد صفری

SCRUM.IR

مقدمه ای بر اسکرام

و

تفکر چابک

اسد صفری

آبان 91

Scrum.ir

اجایل[1] یا تفکر چابک به زبان بازاری

قبل از اینکه چیزی به اسم نرم افزار یا کامپیوتر (به معنای امروزی) وجود داشته باشد شرکت ها و صنعت گرانی مانند فورد یا تویوتای ژاپن با چالش هایی در حوزه تولید محصولات مشتری پسند و با کیفیت مواجه بودند . شرکت هایی که سعی در کشف روش های تولید با هزینه پایین و با کیفیت بالا داشتند و از مشاورینی چون دمینگ در سال هایی مانند 1946 بهره جستند. یعنی از زمانی که بشر اقدام به تولید محصولی کرده است سعی کرده آن را با هزینه کم تولید نماید و در همین راستا به روشها و متدهایی دست یافته است که این روش ها بع د توسط دیگر صنعت ها مورد استفاده قرار گرفته است.

مثلا یکی از همین روش ها ، تولید ناب یا Lean می باشد. در Lean چند اصل اساسی مطرح شده است مثلا از بین بردن اتلافات تولید. به طوری که ما در لین بدنبال اتلافات و هر تلاش بی ارزش چرخه کاری می گردیم و با از بین بردن این اتلافات در سیستم تولید بهینه سازی می نماییم.
حال در طی زمان صنعت گران روش ها و اسم های زیادی را به دنیا برای تولید محصولات با ارزش و با کیفیت و البته با هزینه کم معرفی کردند ، البته در این مابین روش هایی هم برای مدیریت منابع انسانی سازمان ها نیز معرفی می شد که چگونه نیروی کار را انگیزه نگه داریم .

زمانیکه کامپیوتر ها پای خود را به خانه ها و سازمانهای ما گشودند، شاهد به وجود آمدن صنعت جدیدی به نام صنعت توسعه نرم افزار بودیم. خروجی این صنعت همانند دیگر صنایع محصولاتی بودند. اما هر جا که محصولی وجود داشته باشد پای یک مشتری در میان است و هر جا پای مشتری در میان باشد حق هم با او خواهد بود . بنابه به شیرین بودن (دلچسب و پر سود و میلیاردهایی همچون مایکروسافت) این صنعت ، به زودی شاهد ورود نفرات زیادی به این صنعت بودیم و نفرات زیاد معادل با تولید محصولات زیادی هم خواهد بود. اما آیا این محصولات به اندازه محصولات تویوتا با کیفیت بودند؟ آیا مشتریان این محصولات خوشحال بودند؟ آیا توسعه گران نرم افزار ها (همان کارگران کارخانه ها) از کار خود راضی بودند؟ و...

اما وضعیت نرم افزاری ها خرابتر از دیگر صنعت گران بود . زیرا نرم افزاری ها خودشان مشتری نرم افزارها و محصولات دیگران بودند و به نوعی این باعث به وجود آمدن سریع تکنولوژی های جدیدی می شد و نرم افزاری ها عملا مشتری را فراموش کرده بودند و غرق تکنولوژی های جدید شده بودند. و کیفیت هم که بسیار خراب . نرم افزار ها پر از باگ و خطا و مشتری ها هم کلافه و سردرگم.

اما بالاخره روزی فرا رسید که صنعت نرم افزار هم حق را به مشتری داد. از اینرو افرادی اقدام به تحقیق در صنایع دیگر کردند (آنها چگونه کار می کنند ، تویوتا چگونه مشتریان خود را خوشحال نگه می دارند؟ مشتری وفادار، ولی چگونه؟ و ...)

[1] Agile

و بدینگونه شد که پس از جمع آوری متدهای تحول دیگر صنایع و بومی سازی آن در صنعت نرم افزار تفکر چابک به وجود آمد. تفکر چابک در سال 2001 توسط 17 نفر از متخصصین نرم افزار طی بیانیه چابک رسما مطرح شد و اکنون به عنوان یک الگوی موفق در سرتاسر جهان مورد استفاده قرار گرفته است.

به طور خلاصه در تفکر چابک یک سری ارزش و اصول معرفی شده است که با به کار بستن آنها در محیط توسعه می توان به نتایجی مانند محصولات کارآمد ، مشتری خوشحال ، نیروی کار با انگیزه دست یافت . اما مشکلی که وجود داشت این بود که تفکر چابک در حد یک بیانیه یا تعریف بود و هیچ راه حل عملی برای آن مطرح نشده بود. در همین زمان متدهایی مطرح شدند (البته قبل از اجایل مطرح شده بودند) که اصول و ارزش های اجایل در آنها نهادینه شده بود.

یکی از این متدها اسکرام (Scrum) است.

اسکرام یکی از متدهای رایج و پرطرفدار تفکر چابک می باشد که تیم ها در آن با همکاری خود مشتری چند هفته یکبار خروجی از نرم افزار را بیرون می دهند و فیدبک ذینفعان را دریافت می کنند و طبق بازخورد ها محصول را در مسیر درست قرار می دهند و اینگونه می شود محصولات مشتری پسندی به وجود می آید.

مشکل بزرگی که در صنعت نرم افزار وجود داشت این بود که خروجی پروژه ها مثلا بعد از دو سال هزینه بدرد نخور می شد ، یعنی تیم دو سال بر روی پروژه کار میکرد و نهایتا مشتری میگفت این آنچیزی نیست که من می خواستم و ... . اما بعد از ظهور تفکر چابک ما مجبوریم مشتری را در طی روند تولید محصول دخیل نماییم به طور مداوم بازخوردهای او را بگیریم ، خلاقیت ایجاد نماییم و به طور کلی محصول را خوشمزه کنیم و نهایتا شاهد بیگ بنگ نباشیم.

Agile Software Development یا توسعه نرم افزار چابک چیست؟

تفکر چابک (Agile) مجموعه ای از ارزش ها و اصول جهت توسعه نرم افزار های کارکننده توسط تیم های خود سازمانده میباشد. ارزش ها و اصول چابک (Agile) در سال ۲۰۰۱ توسط ۱۷ نفر از اساتید معتبر جهانی صنعت توسعه نرم افزار طی یک بیانیه با عنوان بیانیه توسعه چابک تنظیم و ارائه گردید. اساس و هدف این اصول و ارزش ها ارائه نرم افزار کارا و یا محصول کارآ به مشتری می باشد.

اما چه نیازی به روش چابک بود؟ !به عبارتی چه شد که این ۱۷ نفر (بعلاوه چند ده نفر دیگر که به صورت غیر مستقیم در این بیانیه تاثیر داشتند) در سال ۲۰۰۱ اقدام به انتشار چنین روشی کردند!؟

جواب کاملا واضح است: ضعف های موجود در روش های سنتی باعث شد که این دوستان روش چابک را معرفی نمایند.

روش های سنتی و یا موجود چه مشکلاتی داشتند؟

روش های موجود آن زمان (و فعلی این زمان ایران) که بیشتر به صورتی فاز به فاز عملیات اجرا می شد و نتیجه کار از قبل به صورت یک طراحی اولیه دقیق (Up-front Design) پیش بینی می شد یک سری ایراد اساسی و کلی داشت:

- صرف زمان زیاد و در واقع اتلاف زمان برای طراحی اولیه دقیق در فاز اولیه پروژه
- مشخص نبودن و واضح نبودن نیازمندی ها بدلیل ارتباطات کاغذی به جای ارتباطات چهره به چهره و کج روی های متعاقب
- بالا بودن هزینه تغییرات
- به طول انجامیدن پروژه و گذر از زمان تعیین شده در حد بسیار زیاد در بسیاری از پروژه ها
- عدم وجود زمان برای آزمایش محصول و متعاقبا محصولات پر از باگ و بدون کیفیت
- عدم شفافیت در پروسه توسعه و تولید : هیچ کس حتی مدیر پروژه نمی دانست که چقدر از محصول مورد نظر تکمیل شده است؟ چه اهدافی باقی مانده است؟ و. ...

بزرگترین مشکل این پروسه ها ناکارآ بودن محصولات اشان بود. اما چرا محصولات این پروسه ها ناکارآ تشریف داشتند؟

به این دلیل که عملکرد این پروسه ها بر اصل Anticipation یا پیش بینی استوار بود . در پیش بینی یک سری موارد بر روی کاغذ و به عنوان قرار داد بسته جهت پیاده سازی تحویل تیم توسعه داده می شد و تیم توسعه موظف می شد در یک زمان مقرر و با یک منابع مشخص و ثابت این وظایف را پیاده سازی نماید . این اصل همراه خود در بردارنده طراحی دقیق اولیه یا Big Design Up-Front بود (همان صحبت معروفی که شاید شنیده باشید :"ما اگر ۱ سال وقت داشته باشیم ۱۰ ماه آن را برای تحلیل و طراحی و ۲ ماه آن را برای پیاده سازی صرف می کنیم.)

طراحی اولیه دقیق (Big Design Up-Front) خوب است ولی در جایی که همه چیز ثابت باشد و نیازی به تغییر نداشته باشیم ولی آیا به راستی در جهان واقع و در اکثر پروژه هامان نیاز به تغییر نداریم؟ !

زمانیکه در پروژه ای طراحی اولیه دقیق (Up-Front Design) می شود مسلما فاز تست آخرین فازی خواهد شد که بر روی محصول انجام می شود. ولی آیا این فاز تاثیر گذار می تواند باشد؟ محصولی که کامل ساخته شده است تزریق کیفیت به آن ساده نیست و در بعضی از موارد غیر ممکن خواهد بود. بهتر است به جای این , محصول با کیفیت ساخته شود و نه کیفیت به آن در آخر پروژه تزریق شود.

درکل اصل پیش بینی که متد های آن زمان , بر آن استوار بودند رابطه ی خوبی با تغییر نداشتند زیرا آن ها عادت داشتند همه چیز را از قبل پیش بینی کنند و تغییر برای آنها بسیار هزینه بر بود. اما چرا تغییر برای آن پر هزینه بود؟ زیرا آنها تغییرات و کج روی های خود را خیلی دیر متوجه می شدند (اکثرا در آخر پروژه) و در محور زمان هر چقدر تغییر و یا مشکل دیر کشف شود رفع و یا ایجاد آن مشکل تر – پیچیده تر و پرهزینه تر خواهد بود. به همین دلیل آنها تغییر را رد می کردند.

رد کردن تغییر به منزله ناکارآمد کردن محصول و خروجی پروژه می باشد . هر تغییر و یا اصلاحی که در محصول نیاز می شود برای تکمیل و کارآمدتر کردن آن می باشد : به این دلیل که مشتری نیازمند تغییر است (زیرا تجارت او در حال تغییر است و یا اصلا او در شناخت نیازها اشتباه کرده است و یا هر چیز دیگری) و اگر تغییر انجام نشود محصول به درد تجارت و کسب وکار مشتری نخواهد خورد و اگر تغییر نیز انجام شود هزینه آن بالا خواهد بود , که در این صورت هم پروژه به صرف نخواهد بود.

یکی دیگر از مع ایب اصلی پیش بینی جلوگیری از به وجود آمدن نوع آوری در محصول می باشد . یعنی عملا نوع آوری وجود نخواهد داشت.

مشکلات موجود در پروسه های موجود و خروجی های بی کیفیت و محصولات ناکارآ در آن زمان و بخصوص اتلاف و نارضایتی نیروی انسانی پروژه ها و ضررهای مالی کلان در حوزه فناوری اطلاعات این دوستان را بر آن داشت که روش جدیدی برای اصلاح پروسه توسعه نرم افزار معرفی نمایند که همه ما آن را با نام Agile یا چابک می شناسیم.

نقطه عکس حالت پیش بینی حالت Adaptation یا انطباق سازی می باشد که تفکر چابک بر آن استوار می باشد.

در انطباق سازی (Adaptation) به جای یک طراحی دقیق و برای سازگاری بیشتر محصول با نیاز های واقعی مشتری از Real-Time Planning و Emergent Design استفاده می شود. یعنی به حد کافی در زمان های مناسب نیازمندی هایی از طریق ارتباط چهره به چهره دائم و فیدبک مشتری دریافت م ی شود که فقط آنها بر اساس عکس بزرگ محصول برنامه ریزی و طراحی می شوند و نه بیشتر .

Iterative و Incremental (تکراری – افزایشی) بودن تفکر چابک فرصت مغتنمی برای ایجاد یک بستر انطباق سازی (Adaptation) به وجود آورده است. بدین صورت که محصول به صورت افزایشی ساخته می شود و در چرخش ها یا تکرار های معلوم و کوتاه در اختیار مشتری قرار داده می شود در هر تکرار و یا چرخش فیدبک های مشتری به عنوان نیازمندی های جدید وارد پروسه توسعه می شوند که بدلیل فیدبک های دائم و زود هزینه تغییرات بسیار پایین خواهد آمد.

علاوه بر این در چابک بدلیل افزایشی بودن , پروسه آزمایش و یا تست به صورت یکپارچه با توسعه بخش ها انجام می شود که این نوید دهنده یک محصول با کیفیت در هر Release خواهد بود. به عبارت ساده تر به جای تزریق کیفیت , بخش ها را با کیفیت می سازیم که این هم هزینه نگه داری هر بخش را کاهش می دهد و هم محصول نهایی با کیفیت می شود و هم تغییرات قابل اجرا خواهند بود. در حالی که در روش پیش بینی پروسه تست در فاز آخر انجام می شد.

به طور کلی برای ویژگی یک متد بر پایه انطباق سازی (Adaptation) می توان موارد زیر را بر شمرد:

- Real-Time Planning
- Emergent Design
- Integrated Testing
- Collaborative Discussions
- Just-in-Time & Just-Enough Requirements

به طور خلاصه در تفکر چابک هدف این است که محصولی کارآمد تحویل مشتری داده شود و محصول کارآمد محصولی است که با نیاز های مشتری سازگار باشد. برای همین اصول و ارزشهایی برای دست یابی به این مهم در بیانیه توسعه چابک آمده است.

ارزش های توسعه نرم افزار چابک

افراد و تعاملات بالاتر از فرآیندها و ابزارها
نرم افزار کارا بالاتر از مستند سازی جامع
همکاری مشتری بالاتر از قرارداد کار
جوابگویی به تغییرات بالاتر از پیروی یک طرح

با وجود اینکه در موارد سمت چپ ارزش هایی وجود دارد ولی
موارد سمت راست ارزش بیشتری برای ما دارند

ارزش **افراد و تعاملات** به معنی ایجاد یک محیط فعالانه و همکارانه بین اعضای تیم می باشد . ارزش**همکاری مشتری** به معنی دست یابی به فیدبک های مشتری و ایجاد نوع آوری می باشد . ارزش**جوابگویی به تغییرات** هم نیز در جهت دست یابی به ارزش والای **نرم افزار کارآ** می باشد.

به عبارت ساده تر ما تیمی داریم که تعاملات خوبی بین اعضای آن در جریان است و این تیم با گرفتن فیدبک های مشتری به صورت متوالی و سازگار کردن محصول با نیازهای مشتری در حال توسعه نرم افزار کارآ می باشد.

همه این مواردی که ذکر شد ارزش هایی می باشند که انتظار می رود یک سازمان چابک برای خود داشته باشد . ولی اینها وحی منزل نیستند و سازمان ها می توانند ارزش های خود را برای چابک سازی خود داشته باشند.

در کنار ارزش های سازمان چابک ۱۲ اصل چابک نیز وجود دارد که سازمان های چابک برای دست یابی به ارزش های چابک می توانند پیرو آن اصول باشند:

اصول بیانیه چابک

ما از این اصول پیروی می کنیم:

بالاترین اولویت ما رضایت مشتری از طریق
تحویل به موقع و مداوم نرم افزار ارزشمند می باشد

پذیرائی از نیازهای در حال تغییر , حتی آن هایی
که در اواخر توسعه پدید آور می شوند. فرآیند های چابک
تغییرات را جهت رقابت بر سر مشتری مهار و کنترل می نمایند

تحویل نرم افزار کارکننده غالبا از چند هفته تا چند ماه
یک بار انجام می شود که زمانبندی کوتاه تر ترجیح داده می شود

ذینفعان تجاری و توسعه دهندگان باید هر روزه در
طول پروژه با هم کار کنند

پروژه ها را بر روی افراد با انگیزه بنا کنید. محیط لازم را
به آنها بدهید و از نیازهای آن ها پشتیبانی نمایید وبه
آنها اعتماد نمایید تا کارها را انجام بدهند

کارآمدترین و موثرترین روش برای انتقال و رساندن
اطلاعات به تیم توسعه , گفتگوی چهره به چهره و رودرو می باشد

نرم افزار کارکننده اصلی ترین معیار پیشرفت می باشد

فرآیند های چابک توسعه پایدار را ترویج می دهند.
حامیان مالی , توسعه دهندگان و کاربران باید قادر به
حفظ سرعت پیشرفت ثابتی برای یک مدت نامحدود باشند

توجه مداوم به برتری فنی و طراحی خوب باعث
افزایش چابکی می شود

اصل سادگی ضروری می باشد

بهترین معماری ها , نیاز مندی ها و طراحی ها از تیم های
خود سازمانده پدید آور می شود

در فواصل منظم , تیم برچگونگی موثرتر شدن تامل وتفکر می نماید
و سپس تیم رفتار خود را بر اساس بازتاب این تفکر تنظیم و هم سو می نماید

جمع این ارزش ها و اصول در یک سازمان باعث چابک شدن آن سازمان می شود. البته به این نکته توجه نمایید که سازمان چابک کار نمی کند بلکه چابک می شود.

اما چگونه می توان چابک شد ؟

برای چابک شدن باید در پروسه توسعه و یا حتی سطوح کلان سازمان مانند مدیریت منابع انسانی پروژه و یا هر سطحی ارزش های و اصول چابک رعایت شوند و در نظر گرفته شوند . به عبارتی باید همه سازمان چابک شود و نه فقط بخش یا واحد توسعه نرم افزار. به همین دلیل حرکت سازمان به سمت چابک را تغییر یا Change گفته نمی شود و از اصطلاح Transformation یا تحول استفاده می شود. یعنی باید سازمان در راه چابک شدن متحول شود.

نتیجه گیری

تفکر چابک یک تفکر ناب در زمینه توسعه نرم افزار می باشد که خروجی و هدف آن ارائه نرم افزار کارآ می باشد . در تفکر چابک هزینه توسعه بدلیل ناب (Lean) بودن و تحلیل و طراحی سازگار پایین خواهد بود. در تفکر چابک بدلیل Iteration عمل کردن و ارتباط چهره به چهره دائم با مشتری و آزمایش یکپارچه شاهده محصول با کیفیت و کارآ خواهیم بود . در Agile به دلیل خود سازمانده بودن تیم ها شاهد نفرات و تیم های خوشحال و راضی خواهیم بود . و سازمان نیز بدلیل چابک بودن دارای سود بالایی خواهد بود.

اسکرام چیست؟

اسکرام یک چارچوب توسعه نرم افزار از سری متدهای تفکر چابک می باشد.

اسکرام یک چارچوب یا فرآیند؟ مسئله این است

در این موضوع کاملا بین متخصصان اسکرام دوگانگی وجود دارد . اشخاصی مانند کن شوئبر (مبدع اسکرام) دائما از لفظ چارچوب استفاده می کنند و تاکید می نمایند که همه باید این مورد را قبول داشته باشند ولی بعضی دیگر از دوستان از لفظ فرآیند و یا متدولوژی برای اسکرام استفاده می کنند.

با استناد به اصل اسکرام و تجربیات در این مورد می توان بیان کرد که چارچوب عنوان مناسب تری برای اسکرام خواهد بود. شاید سوال پیش بیاید که خوب چارچوب باشد , چه فرقی خواهد کرد ؟مایک کان می تواند تعریف خوبی برای شما ارائه دهد:

Scrum is a framework. So instead of providing complete, detailed descriptions of how everything is to be done on the project, much is left up to the team. This is done because the team will know best how to solve its problem.

به معنی : اسکرام یک فریم وورک (چارچوب) می باشد. پس به جای اینکه اسکرام جزئیات دقیق و کاملی در مورد اینکه کارها در پروژه چگونه باید انجام شوند , بیشتر آن را به تیم واگذار می کند . این کار عملی خواهد بود زیرا تیم خواهد فهمید که چگونه به بهترین نحو مشکل خود را حل نماید.

برای مثال می توان جلسه برنامه ریزی اسپرینت (Sprint Planning) را درنظر گرفت : در اسکرام آمده است که خروجی این جلسه می تواند (بایدی در کار نیست) تعدادی آیتم باشد که آن ها در اسپرینت بعدی پیاده سازی خواهند شد . اما در متدولوژی های دیگر, ضوابط ورودی , تعیین وظایف , ضوابط خروجی و دیگر مسائل جلسه توسط متدولوژی تعیین می شوند و همه آنها لازم الاجرا هستند.

به عبارت ساده تر , در چارچوب نسخه پیچی نداریم ولی در متدولوژی نسخه های پیچیده شده لازم الاجرا می باشند.

پس قابل نتیجه گیری است که در چارچوب اسکرام, کشف راه حل برای مشکلات به تیم واگذار می شود و از ن سخه پیچی (ارائه حل ها) خبری نیست و بیشتر سعی اسکرام کشف و نمایان کردن مشکلات می باشد به جای اینکه مشکلات را حل نماید.

برای تکمیل تعریف اسکرام , در راهنمای اسکرام کن شوئبر و جف سادرلند چنین می خوانیم:

Scrum employs an iterative, incremental approach to optimize predictability and control risk.

به این معنی که : اسکرام برای بهینه سازی پیش بی نی و کنترل ریسک از یک روش تکرار ی - افزایشی یا iterative incremental بهره می جوید.

تعریف اسکرام ما به اینجا رسیده است که: اسکرام یک چارچوب برای توسعه نرم افزار می باشد که از روش تکراری- افزایشی یا Iterative - Incremental بهره می جوید. اما در تعریف اول مطلبی بود که در این یکی تعریف نیست : "از سری متدهای تفکر چابک."

اسکرام یکی از چارچوب های محبوب و پرکاربرد تفکر چابک محسوب می شود.

پس برای کامل کردن تعریف می توان گفت : **اسکرام یک چارچوب توسعه نرم افزار چابک می باشد که از روش تکراری- افزایشی یا Iterative Incremental بهره می جوید.**

گاها در مقالات اسکرام می خوانیم که از اسکرام به عنوان متدولوژی سبک یاد می کنند . البته این وزن کردن متد ها زیاد درست نمی باشد. درهمین باب می خوانیم:

Scrum is an agile framework for completing complex projects.

یا گاها می شنویم که بعضی از افراد می گویند که اسکرام فقط برای پروژه ها و یا شرکت های کوچک موثر است: در همین باب می توانیم به مثال های زیر رجوع کنیم:

پیاده سازی اسکرام در گوگل
استفاده یاهو از اسکرام
اسکرام در مایکروسافت
تجربه اسکرام درsalesforce

به نظر می رسد که سبک و سنگین کردن اسکرام و یا محدود کردن آن به پرو ژه های کوچک کاری است غیر عادلانه و شاید کوته فکرانه. پس اسکرام چارچوبی است که در نرم افزار های پیچیده و بزرگ نیز طبق تجربه جهانی جواب خواهد داد.

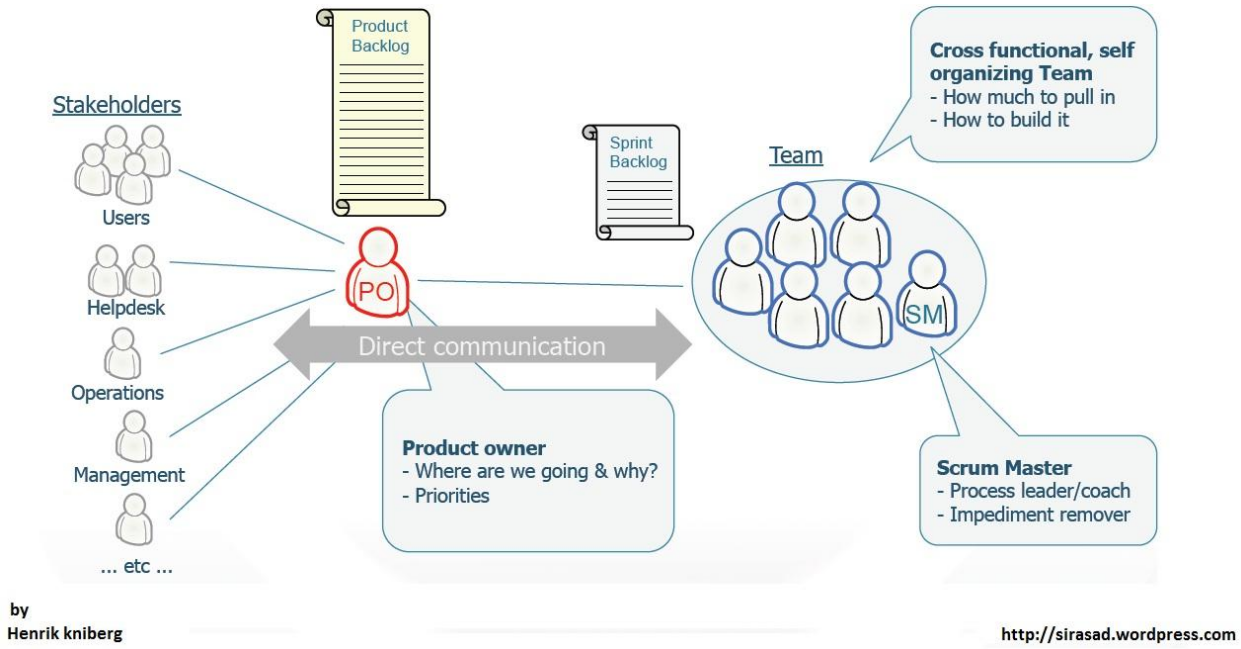
نتیجه گیری بخش ۱

اسکرام یک چارچوب توسعه نرم افزار چابک می باشد که برای توسعه نرم افزار از روش تکراری-افزایشی بهره می جوید. این چارچوب توانایی اداره پروژه های کوچک و هم پروژه های بزرگ را دارا می باشد.

بخش دوم

در بخش اول مقاله تعریفی از اسکرام ارائه شد. در بخش دوم سعی خواهد شد که مبانی و اصول اسکرام تشریح گردند.

اساس و کل اسکرام به صورت شکل زیر می باشد:



از طرف چپ شکل بالا شروع می کنیم:

در سمت چپ این عکس شاهد وجود کاربران ، ذینفعان ، مدیران و دیگرنقش های موجود هستیم . این نقش ها {که شامل درخواست کنندگان محصول و یا ذینفعان محصول و یا استفاده کننده می باشد } از طریق رابط خود یعنی مالک محصول یا Product Owner با تیم توسعه دهنده محصول ارتباط برقرار می کنند.

همانطور که اشاره شد PO یا Product Owner نماینده گروه ذینفعان محصول می باشد PO. به عنوان یکی از نقش های اصلی در محیط اسکرام به حساب می آید . که مسلما هر نقش باید یک سری وظیفه داشته باشد که وظایف PO به این شکل می باشد:

- تعیین هدف و چرایی هدف برای تیم؟ (او مقصد نهایی را نشان می دهد ولی به این کار نخواهد داشت که چگونه تیم می خواهد به آنجا دست یابد)
- اولویت بندی.

اهداف و خواسته های مالک محصول در یک لیست به نام بک لاگ محصول یا Product Backlog جمع آوری می شوند. به عبارت ساده تر Product Backlog شامل تمام خواسته و ویژگی های مورد نظر مالک محصول برای گنجاندن در محصول مورد نظر می باشد.اما این لیست باید مرتب شود ولی بر چه اساسی ؟ اینکار نیز به مالک محصول محول شده است تا او درخواست های

خود را بر اساس اولویت های تجارت خود و یا ذینفعان خود تعیین کند . شاید سوال مطرح شود که چه نیازی به اولویت بندی است؟

از آنجایی که فقط ٪۲۰ از ویژگی های ارائه شده در قالب محصول برای مشتری با اهمیت و با ارزش می باشند چرا نباید این ویژگی ها را قبل از دیگر ویژگی های غیر ضروری ارائ ه داد . به این عمل در واقع تضمین بازگشت سرمایه می گویند . هر چقدر زودتر نیازهای اصلی مشتری را رفع کنید برگشت سرمایه زودتر انجام خواهد شد.

اما علاوه بر مالک محصول نقشی به نام تیم وجود دارد که کار او توسعه محصول مورد نظر خواهد بود . این تیم می تواند شامل تمام نقش های توسعه نرم افزار مانند برنامه نویس , تحلیل گر و طراح ,DBA و ... باشد. تیم در صورت وجود نفرات زیاد می تواند به چندین تیم مجزا نیز تقسیم شود . تعداد ۹-۵ نفر برای هر تیم ترجیح داده می شوند . تیم های اسکرام بر خلاف دیگر تیم ها دو ویژگی اساسی دارند:

- **Self-Organize یا خود سازمانده هستند** .از متد مدیریت Micromanagement یا مدیریت دستور-کنترل برای این تیم ها استفاده نمی شود و این تیم ها خود-سازمانده می باشند . در واقع به این تیم ها فقط گفته می شود که می خواهیم چه کاری انجام دهیم و نه اینکه بگوییم این کار را انجام بده و اینگونه هم انجام بده.
- **Cross-Functional کار می کنند** . اعضای هر تیم به صورت Incremental در فکر پیاده سازی یک واحد کوچک به نام Story می باشند . هر Story در واقع می تواند یک عملیات محصول در نظر گرفته شود. این تیم ها بر خلاف تیم های دیگر که معمولا به صورت Component کار می کنند و هدفشان تکمیل جزئی از محصول هست می باشند.

از جمله وظایف این نقش می توان به موارد زیر اشاره کرد:

- تعیین کردن اینکه چه مقدار کار می توانند در طی یک اسپرینت (۲-۴ هفته) انجام دهند.
- چگونه باید کارها را انجام دهند.

اسپرینت در اسکرام همان Iterative هایی هستند که در بالا ذکر شد . این تکرارها معمولا در اسکرام بین ۴-۲ هفته یکبار انجام می شود . در شروع هر اسپرینت مقداری درخواست مشتری (User Story) با توجه به ظرفیت تیم سرعت تیم یا Velocity انتخاب می شوند و در لیست Sprint Backlog قرار می گیرند. این لیست شامل تمام درخواست های مشتری می شود که تیم متعهد شده است که در این اسپرینت پیاده سازی نماید.

درخواست های مشتری چه در Product Backlog و چه در Sprint Backlog در قالبی به نام User Story نگه داری می شوند و به عبارتی به هر آیتم Backlog یک User Story گفته می شود. نکته مهم در مورد User Story این می باشد که این توضیحات باید به صورت علائم تجاری و مشتری فهم باشند و نه اختصارات فنی . معمولا بهتر است خود مالک محصول این توضیحات را در حضور تیم آماده کند.

برای اینکه بتوان میزان ظرفیت تیم و میزان بهره ور ی آن ها را سنجید باید به هر داستان کاربر یک وزن اطلاق شود . واحد این وزن Story Point می باشد. اما در باره نحوه وزن دادن هر داستان : ساده ترین درخواست مشتری را انتخاب کنید . به آن

Story Point 2بدهید. موارد دیگر باید بر اساس رابطه سختی آن ها با این داستان وزن دهی شوند. مورد بعدی را انتخاب کنید اگر دو برابر سخت تر از این داستان بود به آن ۴ و اگر کمی سخت بود به آن ۳ و اگر به سختی این داستان بود به آن ۱ بدهید.

همانطور که در بالا ذکر شد "در شروع هر اسپرینت مقداری درخواست مشتری (User Story) با توجه به ظرفیت تیم انتخاب می شوند" . ظرفیت تیم همان سرعت تیم خوانده می شود . به عبارت ساده تر تیم در طی اسپرینت ۴ هفته ای چند Story point می تواند انجام بدهد؟ تعیین این میزان برای هر تیم به دو روش امکان دارد:

- تعیین ظرفیت اسپرینت با توجه به عملکرد تیم در اسپرینت های قبلی
- تعیین ظرفیت تیم بر اساس توان نیروی کار موجود و میزان توجه آن ها به کار

به طور خلاصه تیم بر اساس ظرفیت و یا سرعت خود و رعایت اولویت های مالک محصول تعدادی User Story را از داخل Product Backlog انتخاب و به داخل Sprint Backlog کپی می کند. این فعالیت در طی جلسه برنامه ریزی اسپرینت و یا Sprint Planning انجام می شود . این جلسه در ابتدای هر اسپرینت برگزار می شود.

در گام بعدی تیم شروع به پیاده سازی هر یک از User Story های موجود در Sprint Backlog خواهد کرد . برای اینکه اعضای تیم اعلام کنند که چه کاری را انجام داده اند و چه کاری را می خواهند انجام بدهند جلسه ای با عنوان جلسه روزانه اسکرام و یا جلسه سرپایی روزانه در هر روز معمولا در شروع روز کاری به مدت حداکثر ۱۵ دقیقه برگزار می شود . در این جلسه هر کس ۳ سوال اصلی مطرح و به آن ها جواب می دهد:

- دیروز چه کاری انجام دادم
- امروز چه کاری می خواهم انجام بدهم
- چه موانعی در پیش رو دارم و با چه موارد پیش بینی نشده ای برخورد کرده ام ؟

طرح این سوالات برای ایجاد شفافیت و وضوح عملکرد تیم می باشد . این عمل باعث ژل شدن بیشتر تیم می شود.

در نهایت در زمان اتمام زمان اسپرینت , تیم اقدام به آماده سازی یک دمو از موارد تکمیل شده طی اسپرینت می کند . این دمو طی یک جلسه با عنوان Sprint Review یا بازبینی اسپرینت به دیدگان مشتری و دیگر تیم ها گذاشته می شود . هدف این جلسه دریافت بازخوردهای مالک محصول و دیگر نفرات حاضر درجلسه می باشد . تیم بازخورد های دریافتی را مغتنم شمرده و آن ها را در Product Backlog اعمال می نماید و این راهی برای دریافت و کنترل تغییرات می باشد.

آخرین جلسه بعد ا ز اتمام هر اسپرینت , جلسه Scrum Retrospective یا بازنگری عملکرد می باشد . این جلسه فرصتی با ارزش برای بهبود عملکرد تیم می باشد و معمولا فقط تیم در این جلسه حاضر می شود . در این جلسه عملکرد های اسپرینت بررسی و برای آینده یک راه حلی طرح می شود . در این جلسه سوال هایی مانند این ها مطرح می شود:

- کدامیک از عملکردهای ما درست بود

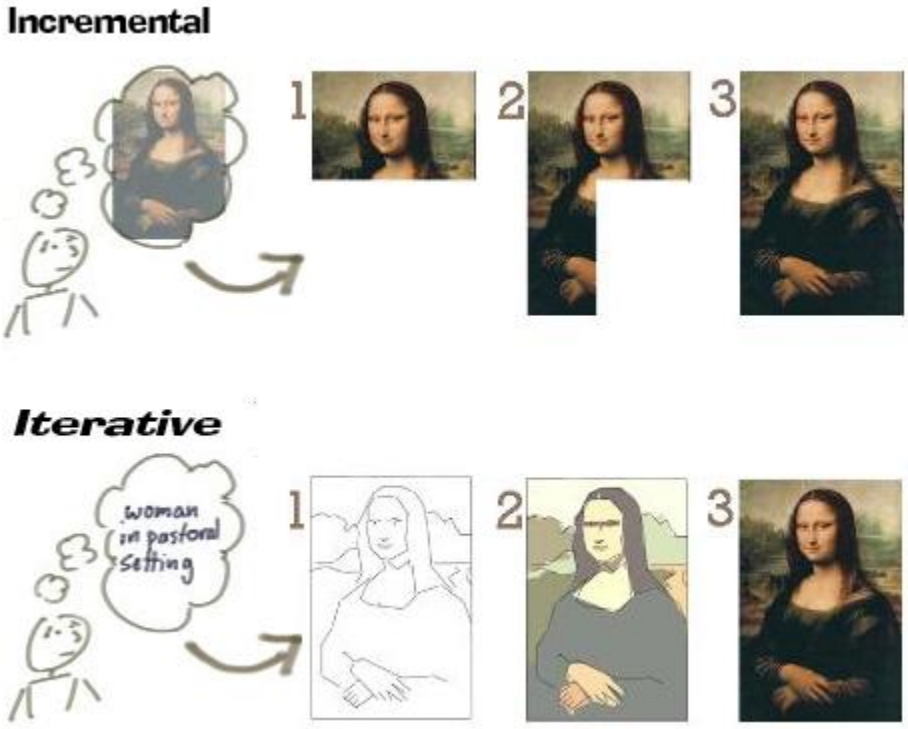
- کدامیک از عملکردهای ما نیاز به اصلاح دارد
- چه بهبودهایی را می توان برای آینده در نظر گرفت

بهبود ها و راه حل های کشف شده در این جلسه ۲ ساعته به مرور در اسپرینت های بعدی اعمال می شوند.

علاوه بر خود تیم نقشی به نام Scrum master یا مدیر اسکرام در داخل تیم وجود دارد . همانطور که از عنوان این نقش قابل ملاحظه است کار این نقش مدیریت پروسه اسکرام می باشد . اولا او پروسه اسکرام را در سازمان کلید می زند و ثانیا او کنترل می کند که آیا تیم و مالک محصول ,اسکرام را درست انجام می دهند ؟ علاوه بر این ها وظیفه حذف موانع از سر تیم در راه فرآیند اسکرام نیز بر عهده SM می باشد .برای هر تیم باید یک SM و یک PO مجزا وجود داشته باشد.

Iterative یا Incremental

یکی از سوالات اصلی که در هنگام سوئیچ از روش های قدیمی به اسکرام با آن مواجه می شویم Incremental (افزایشی) بودن و یا Iterative (تکراری – چرخشی) بودن اسکرام است. به عبارت دیگر ما قرار است به کدامین روش کار کنیم ؟ کل تعریف مدل تکراری و افزایشی در شکل زیر کاملا مشهود است:



همانطور که 100٪ متوجه شده اید هر دور روش دقیقا معکوس یک دیگر عمل می کنند . در روش افزایشی هدف توسعه یک محصول است اما تکه تکه . به عبارت دیگر ما آخر کار را به صورت قطعی می دانیم (همانطور که در تصویر معلوم شده است) پس نسبت به دانش اولیه و کاملمان , در حال تکمیل قسمتی از محصول به صورت 100٪ و کامل هستیم . اگر دقت کنید در متد افزایشی در سه مرحله ما به نقاشی دست پیدا کردیم . البته در مرحله قسمتی (تکه ای) از شکل به صورت کامل تمام شده است . یعنی در مرحله دوم دیگر نیاز نیست ما بر روی چشم ها دوباره کار کنیم زیرا که در مرحله قبل تمام شده است . این نوع متد نیازمند یک دید کلی و کامل از نتیجه نهایی می باشد . یعنی بدون طراحی Up-front قبل از شروع به پیاده سازی چنین متدی امکان پذیر نمی باشد.

ولی در متد دوم یا همان تکراری یا چرخشی شاهد این هستیم که در هر تکرار کل عکس شاهد دگرگونی می باشد . یعنی ما در هر تکرار بر روی کل عکس کار کرده ایم و در آخر تکرار کل عکس آماده است ولی نه به صورت 100 ٪ شکل . این نوع متد وابسته به خروجی تکرار قبلی است . به عبارت ساده تر در اول کار ما فقط می خواهیم عکس یک زن را بکشیم ولی طرح نهایی در ذهنمان وجود ندارد یعنی دانشمان کامل نیست. هر مر حله ای که به جلو می رویم یاد می گیریم و بر اساس یادگیری ها مرحله بعدی را شروع می کنیم . این همان بازخورد و بازبینی هایی هست که در اسکرام انجام می شود.

خوب شاید الان بتوانید بگویید که اسکرام برپایه کدام روش می باشد ؟ بلی , اسکرام ترکیبی از هر دوروش می باشد . (درست گفته بودید ؟) . به نظر می رسد متوجه شده باشید که هر دوروش یک سری معایب و مزایای بزرگ دارند:

در روش افزایشی ما بخشی از محصول را به صورت کامل تمام می کنیم (مزیت) ولی این تمام کردن نیازمند یک دانش کامل و فراگیر نسبت به کل محصول است که این محصول جز با طراحی های Up-front امکان پذیر نخواهد بود (عیب) { قابل ذکر است که عیب طراحی Up-front این است که در فاز شروعی پروژه زمانی زیادی برای اینگونه طراحی ها باید صرف شود که جز تلف کردن وقت چیزی نیست } . اما در روش تکراری ما نیازی به طراحی Up-front نخواهیم داشت زیرا رفته رفته طراحی میکنیم که اصلاحا به این نوع طراحی Just-In-time-Design اطلاق می شود . ولی در اینگونه طراحی می توانیم در دام کارهای 95٪ بیفتیم (شاید دیده باشید که پروژه در دو ماه به 95٪ می رسد ولی 5٪ پروژه 4 ماه طول می کشد که این همان دام 95٪ است که هیچ وقت تمام نمی شود)

اسکرام برای اینکه بتواند روش کاملی باشد از هر دو روش به‌ره جسته است . اسپرینت های یک پروژه همان تکرار های می باشند و انجام دادن ویژگی داخل هر اسپرینت مدل افزایشی خواهند بود . به عبارت سادتر در داخل یک تکرار که همان اسپرینت می باشد ما شاهد مدل افزایشی خواهیم بود.

دو نکته اساسی از نوشته های بالا قابل استنباط خواهد بود که هدف من از نگارش این مقاله فقط تبیین اهمیت این مطالب بوده است و نه ارائه یک سری تعریف کلاسیک:

1. ویژگی های داخل هر اسپرینت زمانی که توسط یک نفر از اعضای تیم برای پیاده سازی انتخاب می شوند باید به صورت افزایشی کار بشوند . یعنی اینکه باید تمام شود یعنی ویژگی های Done شدن را پیدا کند و کنار گذاشته شود و نه اینکه چند روزی کار شود بعد Stand-By و کار بروی ویژگی دیگر و بعد سوییچ به ویژگی قبلی و چند روزی دوباره کار و بعد دوباره . Stand-By برنامه نویس باید وظایف خود را در قبال ویژگی تمام کند و بلافاصله تست های لازم بر روی ویژگی باید انجام شود و خلاصه هر کاری که لازم است که این ویژگی Done شود باید پی در پی و بدون وقفه انجام شود تا سنت حسنه افزایشی به جا آورده شود.
2. اسپرینت ها روح اسکرام هستند . ما تکراری یا چرخشی کار می کنیم تا یاد بگریم و نه اینکه زود به زو د یک سری ویژگی به درد نخور تحویل مشتری بدهیم . ما برآیند یک اسپرینت را تحویل مشتری می دهیم و همراه مشتری اسپرینت و برآیند اسپرینت را بازبینی می کنیم تا یاد بگریم . یاد گیری منظور , اشتباهات نیست . بلکه یاد می گیریم که چگونه یک محصول با ارزش برای مشتری ارائه بدهیم. مایک کوهن اصلاح با ارزش دارد.Get Feedback, Learn and Adapt .

اسکرام واقعا زیباست . ما افزایشی کار می کنیم که بتوانیم یک سری ویژگی های محصول را به صورت کامل شده در آخر اسپرینت برای بازبینی به مشتری ارائه دهیم . تکراری کار می کنیم که بتوانیم بازخوردهای مشتری را دریافت کنیم.