عنوان: **راه‌های تامین Product backlog در تیم‌های مایکروسافت**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۹/۰۲ ۱۲:۵۴:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: Scrum

شاید مهم‌ترین رخداد وبلاگ‌های مرتبط با برنامه نویسی ایرانی در نیمه دوم سال 89، انتشار کتابچه اسکرام و XP ساده شده به زبان فارسی باشد. یکی از فصول این کتابچه، به روش‌های تهیه Product backlog اختصاص دارد که جزو قسمت‌های اولیه پروسه اسکرام است و می‌شود به آن یک to-do list الویت بندی شده هم گفت. تیم‌های مایکروسافت هم به نظر کمابیش بر همین اساس مدیریت می‌شوند. در ادامه لیستی از سایت‌هایی را مشاهده خواهید کرد که این تیم‌های گوناگون درون مایکروسافت از آن‌ها جهت تامین product backlog خود استفاده می‌کنند؛ کاربران (که در اینجا همان برنامه نویس‌ها هستند) با مراجعه به این سایت‌ها نیازهای خود را عنوان کرده و همچنین با وجود امکانات رای دهی، امکان تهیه لیست‌هایی اولویت بندی شده هم وجود دارد:

تیم ASP.NET

تیم WPF
تیم سیلورلایت
تیم Entity framework
تیم WCF RIA Services
تیم WCF Data Services

و ...

تیم‌های خارج از مایکروسافت هم از این ایده استفاده می‌کنند؛ مانند:

برنامه LINQPad

برنامه پروفایلر NHibernate

سیستم‌های وبلاگ دهی

شرکت Redgate

نظرات خوانندگان

نویسنده: مهران طاهری
تاریخ: ۱۳۸۹/۰۹/۰۲ ۱۴:۱۰:۳۱

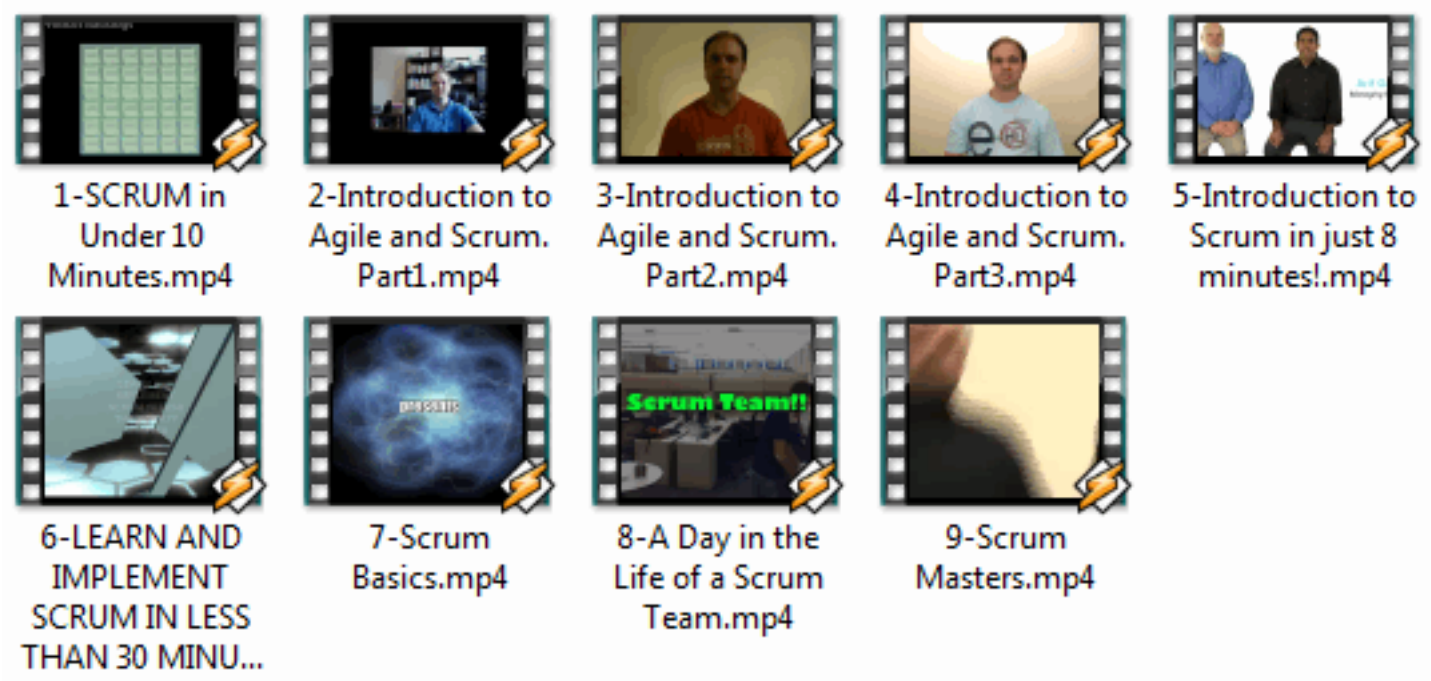
بسیار عالی و ممنون

عنوان: **ویدیوهای آموزشی اسکرام**
نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۹/۰۴ ۰۲:۱۳:۰۰
آدرس: www.dotnettips.info
برچسب‌ها: Scrum

تعدادی ویدیوی آموزشی رایگان مربوط به اسکرام را که از یوتیوب جمع آوری شده‌اند، می‌توانید از یکی از لینک‌های زیر دریافت کنید (تمام لینک‌ها ختم به یک فایل هستند):

+
+ ، + ، + ، + ، + ، +

این مجموعه شامل موارد زیر است:



نظرات خوانندگان

نویسنده: Asad Safari
تاریخ: ۱۳۸۹/۰۹/۰۴ ۱۰:۴۳:۱۳

ممنون جناب نصیری. سایت tvagile.com هم ویدئو های آموزشی خوبی راجع به اسکرام منتشر می کند که دیدن آنها خالی از لطف نیست.

موفق باشید

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۹/۰۴ ۱۰:۴۵:۴۶

ممنون. سایت مفیدی است.

نویسنده: نیما
تاریخ: ۱۳۸۹/۰۹/۰۴ ۱۹:۱۶:۳۱

سلام جناب نصیری
ممنون از لطف شما ولی لینکها قابل دانلود نیستند

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۹/۰۴ ۲۰:۱۴:۳۰

همان لینک‌های فوق:

http://hotfile.com/dl/84751395/ad2e988/scrum.7z.html

http://rs170cg2.rapidshare.com/files/432935299/scrum.7z

/http://www.zshare.net/download/830956779f195011

و ...

نویسنده: وحید نصیری
تاریخ: ۱۳۸۹/۰۹/۰۹ ۰۰:۲۰:۰۱

یک وبلاگ ایرانی در مورد اسکرام:

ScrumDevelopment

عنوان: **چرا توسعه چابک (Agile Development)؟**
نویسنده: توسعه چابک
تاریخ: ۱۳۹۲/۰۹/۱۱ ۱۲:۲۵
آدرس: www.dotnettips.info
گروه‌ها: Scrum, چابک, SOLID, Agile Developement, شی گرایی

خیلی از ما با کابوس پروژه ای که هیچ تجربه ای در انجام آن نداریم روبرو شده ایم. نبودن تجربه موثر منجر به خطاهای تکراری و غیر قابل پیش بینی شده و تلاش و وقت ما را به هدر می-دهد. مشتریان از کیفیت پایین، هزینه بالا و تحویل دیر هنگام محصول ناراضی هستند و توسعه دهندگان از اضافه کارهای بیشتر که منجر به نرم افزار ضعیت-تر می-گردد، ناخشنود.

همین که با شکستی مواجه می-شویم از تکرار چنین پروژه هایی اجتناب می-کنیم. ترس ما باعث می-شود تا فرآیندی بسازیم که فعالیت-های ما را محدود نموده و ایجاد آرتیفکت-ها [۱] را الزامی کند. در پروژه- جدید از چیزهایی که در پروژه‌های قبلی به خوبی کار کرده-اند، استفاده می-کنیم. انتظار ما این است که آنها برای پروژه جدید نیز به همان خوبی کار کند.

اما پروژه-ها آنقدر ساده نیستند که تعدادی محدودیت و آرتیفکت- ما را از خطاها ایمن سازند. با بروز خطاهای جدید ما آنها را شناسایی و رفع می-کنیم. برای اینکه در آینده با این خطاها روبرو نشویم آنها را در محدودیت-ها و آرتیفکت-های جدیدی قرار می-دهیم. بعد از انجام پروژه‌های زیاد با فرآیندهای حجیم و پر زحمتی روبرو هستیم که توانایی تیم را کم کرده و باعث کاهش کیفیت تولید می-شوند.

فرآیندهای بزرگ و حجیم می-تواند مشکلات زیادی را ایجاد کند. متاسفانه این مشکلات باعث می-شود که خیلی از افراد فکر کنند که علت مشکلات، نبود فرآیندهای کافی است. بنابراین فرآیندها را حجیم-تر و پیچیده-تر می-کنند. این مسئله منجر به تورم فرآیندها می-گردد که در محدوده سال ۲۰۰۰ گریبان بسیاری از شرکت-های نرم افزاری را گرفت.

اتحاد چابک

در وضعیتی که تیم-های نرم افزاری در بسیاری از شرکت-ها خود را در مردابی از فرآیندهای زیاد شونده می-دیدند، تعدادی از خبره-های این صنعت که خود را اتحاد چابک [۲] نامیدند در اویل سال ۲۰۰۱ یکدیگر را ملاقات کرده و ارزش هایی را معرفی کردند تا تیم-های نرم افزاری سریعتر نرم افزار را توسعه داده و زودتر به تغییرات پاسخ دهند. چند ماه بعد، این گروه ارزش-هایی تعریف شده را تحت مانیفست اتحاد چابک در سایت http://agilemanifesto.org منتشر کردند.

مانیفست اتحاد چابک
ما با توسعه نرم افزار و کمک به دیگران در انجام آن، در حال کشف راه‌های بهتری برای توسعه نرم افزار هستیم. از این کار به ارزش‌های زیر می-رسیم :

۱- افراد و تعاملات بالاتر از فرآیندها و ابزارها

۲- نرم افزار کار کننده بالاتر از مستندات جامع

۳- مشارکت مشتری بالاتر از قرارداد کاری

۴- پاسخگویی به تغییرات بالاتر از پیروی از یک برنامه

با آنکه موارد سمت چپ ارزشمند هستند ولی ما برای موارد سمت راست ارزش بیشتری قائل هستیم.

افراد و تعاملات بالاتر از فرآیندها و ابزارها
افراد مهمترین نقش را در پیروزی یک پروژه دارند. یک فرآیند عالی بدون نیروی مناسب منجر به شکست می-گردد و بر عکس

افراد قوی تحت فرآیند ضعیت ناکارآمد خواهند بود.

یک نیروی قوی لازم نیست که برنامه نویسی عالی باشد، بلکه کافیست که یک برنامه نویسی معمولی با قابلیت همکاری مناسب با سایر اعضای تیم باشد. کار کردن با دیگران، تعامل درست و سازنده با سایر اعضای تیم خیلی مهمتر از این که یک برنامه نویس با هوش باشد. برنامه نویسان معمولی که تعامل درستی با یکدیگر دارند به مراتب موفقتر هستند از تعداد برنامه نویسی عالی که قدرت تعامل مناسب با یکدیگر را ندارند.

در انتخاب ابزارها آنقدر وقت نگذارید که کار اصلی و تیم را فراموش کنید. به عنوان مثال می-توانید در شروع به جای بانک اطلاعاتی از فایل استفاده کنید، به جای ابزار کنترل کد گرانقیمت از برنامه رایگان کد باز استفاده کنید. باید به هیچ ابزاری عادت نکنید و صرفا به آنها به عنوان امکانی جهت تسهیل فرآیندها نگاه کنید.

نرم افزار کار کننده بالاتر از مستندات جامع

نرم افزار بدون مستندات، فاجعه است. کد برنامه ابزار مناسبی برای تشریح سیستم نرم افزاری نیست. تیم باید مستندات قابل فهم مشتری بسازد تا ابعاد سیستم از تجزیه تحلیل تا طراحی و پیاده سازی آن را تشریح نماید.

با این حال، مستندات زیاد از مستندات کم بدتر است. ساخت مستندات زیاد نیاز به وقت زیادی دارد و وقت بیشتری را می‌گیرد تا آن را با کد برنامه به روز نمایید. اگر آنها با یکدیگر به روز نباشند باعث درک اشتباه از سیستم می‌شوند.

بهتر است که همیشه مستندات کم حجمی از منطق و ساختار برنامه داشته باشید و آن را به روز نماید. البته آنها باید کوتاه و برجسته باشند. کوتاه به این معنی که ۱۰ تا ۲۰ صفحه بیشتر نباشد و برجسته به این معنی که طراحی کلی و ساختار سطح بالای سیستم را بیان نماید.

اگر فقط مستندات کوتاه از ساختار و منطق سیستم داشته باشیم چگونه می‌توانیم اعضای جدید تیم را آموزش دهیم؟ پاسخ کار نزدیک شدن به آنها است. ما دانش خود را با نشستن در کنار آنها و کمک کردن به آنها انتقال می-دهیم. ما آنها را بخشی از تیم می-کنیم و با تعامل نزدیک و رو در رو به آنها آموزش می-دهیم.

مشارکت مشتری بالاتر از قرارداد کاری

نرم افزار نمی-تواند مثل یک جنس سفارش داده شود. شما نمی‌توانید یک توصیف از نرم افزاری که می‌خواهید را بنویسید و آنگاه فردی آن را بسازد و در یک زمان معین با قیمت مشخص به شما تحویل دهد. بارها و بارها این شیوه با شکست مواجه شده است.

این قابل تصور است که مدیران شرکت به اعضای تیم توسعه بگویند که نیازهای آنها چیست، سپس اعضای تیم بروند و بعد از مدتی برگردند و یک سیستمی که نیازهای آنها را برآورده می‌کند، بسازند. اما این تعامل به کیفیت پایین نرم افزار و در نهایت شکست آن می‌انجامد. پروژه‌های موفق بر اساس دریافت بازخورد مشتری در بازه‌های زمانی کوتاه و مداوم است. به جای وابستگی به قرارداد یا دستور کار، مشتری به طور تنگاتنگ با تیم توسعه کار کرده و مرتبا اعمال نظر می-کند.

قراردادی که مشخص کننده نیازمندیها، زمانبندی و قیمت پروژه است، اساسا نقص دارد. بهترین قرارداد این است که تیم توسعه و مشتری با یکدیگر کار کنند.

پاسخگویی به تغییرات بالاتر از پیروی از یک برنامه

توانایی پاسخ به تغییرات اغلب تعیین کننده موفقیت یا شکست یک پروژه نرم افزاری است. وقتی که طرحی را می-ریزیم باید مطمئن شویم که به اندازه کافی انعطاف پذیر است و آمادگی پذیرش تغییرات در سطح بیزنس و تکنولوژی را دارد.

مسیر یک پروژه نرم افزاری نمی-تواند برای بازه زمانی طولانی برنامه ریزی شود. اولا احتمالا محیط تغییر می-کند و باعث تغییر در نیازمندی‌ها می-شود. ثانیا همین که سیستم شروع به کار کند مشتریان نیازمندی-های خود را تغییر می‌دهند. بنابراین اگر بدانیم که نیازها چیست و مطمئن شویم که تغییر نمی-کنند، قادر به برآورد مناسب خواهیم بود، که این شرایط بعید است.

یک استراتژی خوب برای برنامه ریزی این است که یک برنامه ریزی دقیق برای یک هفته بعد داشته باشیم و یک برنامه ریزی کلی برای سه ماه بعد.

اصول چابک

-۱ بالاترین اولویت ما عبارت است از راضی کردن مشتری با تحویل سریع و مداوم نرم افزار با ارزش. تحویل نرم افزار با کارکردهای کم در زود هنگام بسیار مهم است چون هم مشتری چشم اندازی از محصول نهایی خواهد داشت و هم مسیر کمتر به بیراهه می‌رود.

-۲ خوش آمدگویی به تغییرات حتی در انتهای توسعه. اعضای تیم چابک، تغییرات را چیز خوبی می‌بینند زیرا تغییرات به این معنی است که تیم بیشتر یاد گرفته است که چه چیزی مشتری را راضی می‌کند.

-۳ تحویل نرم افزار قابل استفاده از چند هفته تا چند ماه با تقدم بر تحویل در دوره زمانی کوتاهتر . ما مجموعه از مستندات و طرحها را به مشتری نمی‌دهیم.

-۴ افراد مسلط به بیزنس و توسعه دهندگان باید روزانه با یکدیگر روی پروژه کار کنند . یک پروژه نرم افزاری نیاز به هدایت مداوم دارد.

-۵ ساخت پروژه را بر توان افراد با انگیزه بگذارید و به آنها محیط و ابزار را داده و اعتماد کنید . مهمترین فاکتور موفقیت افراد هستند، هر چیز دیگر مانند فرآیند، محیط و مدیریت فاکتورهای بعدی محسوب می-شوند که اگر تاثیر بدی روی افراد می-گذارند، باید تغییر کنند.

-۶ بهترین و موثرترین روش کسب اطلاعات در تیم توسعه، ارتباط چهره به چهره است. در تیم چابک افراد با یکدیگر صحبت می‌کنند. نامه نگاری و مستند سازی فقط زمانی که نیاز است باید صورت گیرد.

-۷ نرم افزار کار کننده معیار اصلی پیشرفت است . پروژه‌های چابک با نرم افزاری که در حال حاضر نیازهای مشتری را پاسخ می‌دهد، سنجیده می‌شوند. میزان مستندات، حجم کدهای زیر ساخت و هر چیز دیگری غیره از نرم افزار کار کننده معیار پیشرفت نرم افزار نیستند.

-۸ فرآیندهای چابک توسعه با آهنگ ثابت را ترویج می‌دهد. حامیان، توسعه دهندگان و کاربران باید یک آهنگ توسعه ثابت را حفظ کنند که بیشتر شبیه به دو ماراتون است یا دوی ۱۰۰ متر. آنها با سرعتی کار می‌کنند که بالاترین کیفیت را ارائه دهند.

-۹ توجه مداوم به برتری تکنیکی و طراحی خوب منجر به چابکی می-گردد . کیفیت بالاتر کلیدی برای سرعت بالا است. راه سریعتر رفتن این است که نرم افزار تا جایی که ممکن است پاک و قوی نگهداریم. بنابراین همه اعضای تیم چابک تلاش می-کنند که با کیفیت-ترین کار ممکن را انجام دهند. آنها هر آشفتگی را به محض ایجاد برطرف می‌کنند.

-۱۰ سادگی هنر بیشینه کردن مقدار کاری که لازم نیست انجام شود، است. تیم چابک همیشه ساده‌ترین مسیر که با هدف آنها سازگار است را در پیش می-گیرند. آنها وقت زیادی روی مشکلاتی که ممکن است فردا رخ دهد، نمی-گذارند. آنها کار امروز را با کیفیت انجام داده و مطمئن می-شوند که تغییر آن در صورت بروز مشکلات در فردا، آسان خواهد بود.

-۱۱ بهترین معماری و طراحی از تیم‌های خود سازمان ده بیرون می‌آید. مدیران، مسئولیت‌ها را به یک فردی خاصی در تیم نمی‌دهند بلکه بر عکس با تیم به صورت یک نیروی واحد برخورد می-کنند. خود تیم تصمیم می-گیرد که هر مسئولیت را چه کسی انجام دهد. تیم چابک با هم روی کل جنبه‌های پروژه کار می-کنند. یعنی یک فرد خاص مسئول معماری، برنامه نویسی، تست و غیره نیستند. تیم، مسئولیتها را به اشتراک گذاشته و هر فرد بر کل کار تاثیر دارد.

-۱۲ در بازهای زمانی مناسب تیم در می-یابد که چگونه می-تواند کاراتر باشد و رفتار خود را متناسب با آن تغییر دهد. تیم

می-داند که محیط دائما در حال تغییر است، بنابراین خود را با محیط تغییر می-دهد تا چابک بماند.

ضرورت توسعه چابک

امروزه صنعت نرم افزار دارای سابقه بدی در تحویل به موقع و با کیفیت نرم افزار است. گزارشات بسیاری تایید می-کنند که بیش از ۸۰ درصد از پروژه‌های نرم افزاری با شکست مواجه می-شوند؛ در سال ۲۰۰۵ موسسه IEEE برآورد زده است که بیش از ۶۰ بیلیون دلار صرف پروژه‌های نرم افزاری شکست خورده شده است. عجب فاجعه-ای؟

شش دلیل اصلی شکست پروژه‌های نرم افزاری

وقتی که از مدیران و کارکنان سوال می-شود که چرا پروژه‌های نرم افزاری با شکست مواجه می-شوند، آنها به موضوعات گسترده ای اشاره می-کنند. اما شش دلیل زیر بارها و بارها تکرار شده است که به عنوان دلایل اصلی شکست نرم افزار معرفی می-شوند:

-۱ درگیر نشدن مشتری

-۲ عدم درک درست نیازمندها

-۳ زمان بندی غیر واقعی

-۴ عدم پذیریش و مدیریت تغییرات

-۵ کمبود تست نرم افزار

-۶ فرآیندهای غیر منعطف و باد دار

چگونه چابکی این مشکلات را رفع می-کند؟

با آنکه Agile برای هر مشکلی راه حل ندارد ولی برای مسائل فوق بدین صورت کمک می-کند:

مشتری پادشاه است!

برای رفع مشکل عدم همکاری کاربر نهایی یا مشتری، Agile مشتری را عضوی از تیم توسعه می-کند. به عنوان عضوی از تیم، مشتری با تیم توسعه کار می-کند تا مطمئن شود که نیازمندها به درستی برآورده می-شوند. مشتری همکاری می-کند در شناسایی نیازمندی-ها، تایید می-کند نتیجه نهایی را و حرف آخر را در اینکه کدام ویژگی به نرم افزار اضافه شود، حذف شود و یا تغییر کند، را می-زند.

نیازمندی‌ها به صورت تست-های پذیرش [۳] نوشته می-شوند

برای مقابله با مشکل عدم درک درست نیازمندی-ها، Agile تاکید دارد که نیازمندیهای کسب شده باید به صورت ویژگی-هایی تعریف شوند که بر اساس معیارهای مشخصی قابل پذیرش باشند. این معیارهای پذیرش برای نوشتن تست-های پذیرش به کار می-روند. به این ترتیب قبل از اینکه کدی نوشته شود، ابتدا تست پذیرش نوشته می-شود. این بدین معنی است که هر کسی باید اول فکر کند که چه می-خواهد، قبل از اینکه از کسی بخواهد آن را انجام دهد. این راهکار فرایند کسب نیازمندی-ها را از بنیاد تغییر می-دهد و به صورت چشم گیری کیفیت برآورد و زمان بندی را بهبود می-دهد.

زمانبندی با مذاکره بین تیم توسعه و سفارش دهنده تنظیم می-شود

برای حل مشکل زمان بندی غیر واقعی، Agile زمان بندی را به صورت یک فرآیند مشترک بین تیم توسعه و سفارش دهنده تعریف می-کند. در شروع هر نسخه از نرم افزار، سفارش دهنده ویژگی‌های مورد انتظار را به تیم توسعه می-گوید. تیم توسعه تاریخ تحویل را بر اساس ویژگی-ها برآورد می-زد و در اختیار سفارش دهنده قرار می-دهد. این تعامل تا رسیدن به یک دیدگاه مشترک ادامه می-یابد.

هیچ چیزی روی سنگ حک نشده است، مگر تاریخ تحویل

برای رفع مشکل ضعف در مدیریت تغییرات، Agile اصرار دارد که هر کسی باید تغییرات را بپذیرد و نسبت به آنها واقع بین باشد. یک اصل مهم Agile می-گوید که هر چیزی می-تواند تغییر کند مگر تاریخ تحویل! به عبارت دیگر همین که محصول به سمت تولید شدن حرکت می-کند، مشتری (در تیم محصول) می-تواند بر اساس اولویت-ها و ارزش-های خود ویژگی-های محصول را کم یا زیاد کرده و یا تغییر دهد. به هر حال او باید واقع بین باشد. اگر او یک ویژگی جدید اضافه کنید، باید تاریخ تحویل را تغییر دهد. به این ترتیب همیشه تاریخ تحویل رعایت می-گردد.

تست-ها قبل از کد نوشته می-شوند و کاملا خودکار هستند

برای رفع مشکل کمبود تست، Agile تاکید می-کند که ابتدا باید تست-ها نوشته شوند و همواره ارزیابی گردند. هر برنامه نویس باید اول تست- را بنویسد، سپس کد لازم برای پاس شدن آن را. همین که کد تغییر می-کند باید تست-ها دوباره اجرا شوند. در این راهکار، هر برنامه نویس مسئول تست-های خود است تا درستی برنامه از ابتدا تضمین گردد.

مدیریت پروژه یک فعالیت جداگانه نیست

برای رفع مشکل فرآیندهای غیر منعطف و باددار، Agile مدیریت پروژه را درون فرآیند توسعه می-گنجاند. وظایف مدیریت پروژه بین اعضای تیم توسعه تقسیم می-شود. برای مثال هر ۷ نفر در تیم توسعه نرم افزار (متدلوژی اسکرام) زمان تحویل را با مذاکره تعیین می-کنند. همچنین کد برنامه به صورت خودکار اطلاعات وضعیت پروژه را تولید می-کند. به عنوان مثال نمودار burndown ، تست-های انجام نشده، پاس شده و رد شده به صورت خودکار تولید می-شوند.

به کار گیری توسعه چابک

یکی از مشکلات توسعه چابک این است که شما اول باید به خوبی آن را درک کنید تا قادر به پیاده سازی درست آن باشید. این درک هم باید کلی باشد (مانند Scrum و XP) و هم جزئی (مانند TDD و جلسات روازنه). اما چگونه باید به این درک برسیم؟ کتاب-ها و مقالات انگلیسی زیادی برای یادگیری توسعه چابک و پیاده سازی آن در سازمان وجود دارند، ولی متاسفانه منابع فارسی کمی در این زمینه است. هدف این کتاب رفع این کمبود و آموزش عملی توسعه چابک و ابزارهای پیاده سازی آن است.

برای این یک توسعه دهنده چابک شوید، باید به مهارت-های فردی و تیمی چابک برسید. در ادامه این مهارت-ها معرفی می-شوند.

مهارت-های فردی

قبل از هر چیز شما باید یک برنامه نویس باشید و مقدمات برنامه نویسی مانند الگوریتم و فلوچارت، دستورات برنامه نویسی، کار با متغیرها، توابع و آرایه-ها را بلد باشید. پس از تسلط به مقدمات برنامه نویسی می-توانید مهارت-های برنامه نویسی چابک را فرا بگیرید که عبارتند از:

- برنامه نویسی شیءگرا

- توسعه تست محور

- الگوهای طراحی

در ادامه نحوه کسب این مهارت-ها بیان می-شوند.

برنامه نویسی شیءگرا

اساس طراحی چابک بر تفکر شیءگرا استوار است. بنابراین تسلط به مفاهیم و طراحی شیءگرا ضروری است.

توسعه تست محور

مهمترین و انقلابی‌ترین سبک برنامه نویسی از دهه گذشته تا به امروز، توسعه یا برنامه نویسی تست محور است. این سبک بسیاری از ارزش‌های توسعه چابک را فراهم می-کند و یادگیری آن برای هر توسعه دهنده چابک ضروری است.

الگوهای طراحی

الگوهای طراحی راه حل-های انتزاعی سطح بالا هستند. این الگوها بهترین تکنیک-های [۴] طراحی نرم افزار هستند و بسیاری از مشکلاتی که در طراحی نرم افزار رخ می-دهند با استفاده از این الگوها قابل حل هستند.

مهارت-های تیمی

انجام پروژه نرم افزاری یک کار تیمی است. شما پس از یادگیری مهارت-های فردی باید خود را آماده حضور در تیم توسعه چابک کنید. برای این منظور باید با مهارت تیمی مانند آشنایی با گردشکار تولید نرم افزار، حضور موثر در جلسات، قبول مسئولیت-ها و غیره آشنا شوید.

اسکرام

تمامی مهارت-های تیمی توسعه چابک توسط اسکرام آموزش داده می-شوند. اسکرام فریم ورکی برای توسعه چابک است که با تعریف فرآیندها، نقش-ها و آرتیفکت-های مشخص به تیم-های نرم افزاری کمک می-کند تا چابک شوند.

[۱] Artifact : خروجی یک فرآیند است. مثلا خروجی طراحی شیءگرا، نمودارهای UML است.

[۲] Agile Alliance

[3] Acceptance Tests

[4] Best Practice

اطلاعات بیشتر در http://AgileDevelopment.ir

نظرات خوانندگان

نویسنده: ایمان
تاریخ: ۱۳۹۲/۱۰/۰۱ ۲۳:۵۸

شاید آدرس زیر هم به کار بیاد

Imanagement.co

فیلم‌های آموزشی رایگان راجع به مدیریت پروژه‌های نرم افزاری به شیوه اجایل

عنوان: **Pro Agile .NET Development With Scrum - قسمت اول**
نویسنده: غلامرضا ربال
تاریخ: ۱۳۹۴/۰۵/۱۷ ۹:۳۵
آدرس: www.dotnettips.info
گروه‌ها: Agile, Scrum

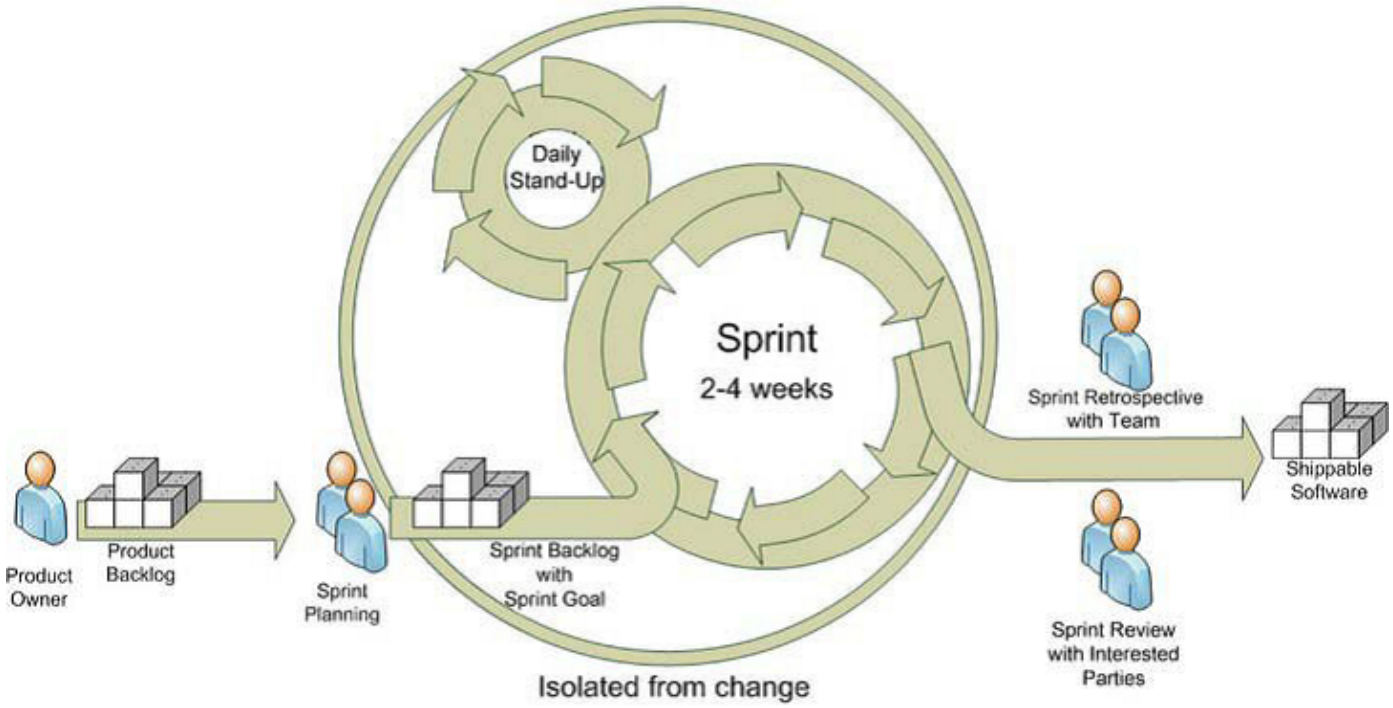
با همکاری آقایان سید مجتبی حسینی و محمد شریفی طی یک سری مقالات سریالی قصد داریم ترجمه آزادی از کتاب Pro Agile .NET Development With Scrum نوشته Jerrel Blankenship و Matthew Bussa ، داشته باشیم.
با توجه به اینکه در سایت جاری مطالب قسمت اول کتاب پوشش داده شده است، ما هم دوباره کاری نکرده و میتوانید از این مقاله استفاده کنید.

مدیریت پروژه‌های چابک با اسکرام

در این فصل با روشها و ماهیت تکرارپذیر اسکرام آشنا می‌شوید که استخوان‌بندی فرآیندی را تعریف می‌کند که دربردارندهٔ مجموعه‌ای از نقشها و فعالیتهایی است که همگی بر پشتیبانی از تیم مسؤول تولید محصول، تمرکز می‌کنند.
مطالعهٔ موردی بخش دوم این کتاب از شیوهٔ اسکرام به نحوی پیروی می‌کند که قادر به دیدن اجرایی عملی از تمام ویژگیهای کلیدی‌ای که در این فصل از آنها بحث می‌شود، خواهید بود و به شما کمک می‌کند تا مزیت‌های این شیوه را به خوبی درک کنید.

اسکرام چیست؟

اسکرام رویکردی تکرارپذیر جهت توسعهٔ نرم‌افزار است که بصورت تنگاتنگی با اصول و بیانیهٔ چابک هم‌سو شده است. اسکرام از دنباله‌ای از بلاکهای زمانی به نام اسپرینت ساخته شده است که بر ارائهٔ محصولات کارآمد تمرکز می‌کند. یک اسپرینت نوعاً از دو تا چهار هفته به طول می‌انجامد و با هدف یا موضوعی که واضح کنندهٔ اسپرینت است، تعریف شده است.
اسپرینتها نسبت به تغییرات ایزوله شده‌اند و بدون هیچ اختلالی، تیم توسعه را بر ارائهٔ محصولی کارآمد، متمرکز می‌سازند. کارها در Product Backlog (لیستی از کارهای کلی یک پروژه است که باید آن را بر اساس درجه اهمیت، دسته بندی نمود) اولویت‌بندی شده که توسط صاحب محصول مدیریت می‌شود. قبل از وقوع هر اسپرینت، یک ویژگی از Product Backlog انتخاب شده و تیم توافق می‌کند که در انتهای آن اسپرینت، آن ویژگی را ارائه کند.
برای آنکه همه چیز بخوبی پیش برود، یک نفر به عنوان ScrumMaster (که وظیفه نگهداری و حفظ فرآیند را برعهده دارد) تعیین می‌شود تا اطمینان حاصل شود که هیچ مانعی باعث جلوگیری از ارائهٔ ویژگیهایی که تیم توسعه مد نظر قرار داده، نشود. جلسات سرپایی روزانه به تیم کمک می‌کند تا دربارهٔ هر مشکلی که مانع کار است، گفتگو کنند. مرور هر اسپرینت در انتهای آن به ارتقای فرآیند کمک می‌کند.
شکل 2-1 نمایش گرافیکی روش اسکرام است که حاوی همهٔ نقشها و فعالیتها و خروجیهای اسکرام بوده که در ادامه، بیشتر دربارهٔ آنها خواهید خواند.



شیوه‌های برنامه محور در مقابل شیوه‌های ارزش محور

هنگام ملاحظهٔ تفاوت میان شیوهٔ آبشاری و شیوهٔ چابک، نیاز است تا به هستهٔ مرکزی هر روش نگریست. یکی از شیوه‌ها از نقشه‌ای برگرفته شده که در ابتدای پروژه ایجاد شده است و شیوهٔ دیگر از ارزشی برگرفته شده که شما به مشتری می‌دهید.

شیوهٔ آبشاری (برنامه محور)

به شیوهٔ آبشاری می‌توان به منزلهٔ شیوه‌ای برنامه محور در توسعهٔ نرم‌افزار نگریست. در گذشته، این شیوهٔ توسعه بسیار مورد استفاده بود، نه به این دلیل که بهترین شیوهٔ توسعهٔ نرم‌افزار بود، بلکه به این دلیل که تنها شیوهٔ شناخته شده بود.

پروژه‌ای که شیوهٔ آبشاری را به کار می‌برد با ریسک بسیار بالایی مواجه بود؛ به این دلیل که همه چیز در ابتدای پروژه طرح‌ریزی می‌شد. تمام نیازمندیها و جستجوها و تعیین بازهٔ کاری قبل از آنکه حتی یک خط کد نوشته شود، جمع‌آوری می‌شد. مشتریان باید همهٔ آنچه را که از سیستم انتظار داشتند، در ابتدای امر می‌دانستند. در زمانی که مشتریان دقیقاً نمی‌دانستند که چه می‌خواهند اما باید تمام جزئیات نیازهای خود را تعریف می‌کردند و در یک وهله باید جزئیات کار را تعیین می‌کردند و تا آخر نیز نمی‌توانستند آن را تغییر دهند؛ حتی اگر بعداً متوجه می‌شدند که نیازشان تغییر کرده است.

این رویکرد سرانجام پروژه را، حتی قبل از آنکه شروع شود، با شکست مواجه می‌کرد. کل فرآیند به سمت مشکلاتی هدایت می‌شد که تا پایان پروژه نیز پنهان می‌ماندند. زیرا مشتری همهٔ نکات جزئی کار را مدنظر قرار نداده بود و راهی برای تغییرات مورد نیاز وجود نداشت. گاهی انجام تغییر مستلزم هزینهٔ بسیار بالایی بود. در این گونه پروژه‌ها دامنهٔ پروژه دچار تغییرات می‌شد؛ توسعه‌دهنده از مسائلی که مشتری درصدد حل آنها بود سردرنمی‌آورد و به همین ترتیب مشتری.

توسعهٔ برنامه محور به مانند روند پرش حلقه‌ای است: شما ابتدا جستجو می‌کنید و یک مرتبه از میان آن حلقه پریده و وارد حلقهٔ جمع‌آوری نیازمندیها می‌شوید و از آنجا وارد حلقهٔ طراحی می‌شوید. از یک حلقه نمی‌توانید عبور کنید مگر اینکه از حلقهٔ پیشین آن پریده باشید و با یک مرتبه، عبور از یک حلقه برگشتن به آن حلقه ممکن نیست. حتی اگر نیاز باشد چنین کاری انجام شود. ممکن نیست که اندکی از هر کاری را انجام داده و برای اطمینان از مسیر درست، قدری متوقف بمانید. فرآیند آبشاری فراهم کنندهٔ بستری نیست که در آن توسعه دهند بتواند به مشتری خود بگوید: «مایل هستم که کاری را که تاکنون انجام داده‌ام، به شما نشان دهم تا ببینید که آیا با آنچه شما می‌خواهید منطبق است یا خیر».

معمولاً در انتهای پروژه است که مشکلات بزرگی بروز پیدا می‌کنند که نسبتاً خیلی دیر است. این مورد منجر به آن می‌شود که چند تیم به کار وارد شده و افراد بیشتری در پروژه استفاده شوند؛ به این امید که پروژه سریعتر به اتمام برسد و البته چنین نتیجه‌ای به ندرت اتفاق می‌افتد. در نتیجه بخشهایی از پروژه باید کنار گذاشته شوند؛ یعنی یا حدود پروژه محدودتر شود، یا آزمودن آن حذف شود یا هردو.

شیوهٔ اسکرام (ارزش محور)

اسکرام به عنوان شیوه‌ای ارزش محور در توسعهٔ نرم‌افزار مورد توجه قرار می‌گیرد. اسکرام به چند دلیل تغییر چشم‌گیری نسبت به شیوهٔ آبشاری داشته‌است. اسکرام به جای آنکه در ابتدا به جمع‌آوری نیازمندیهای مورد نیاز برای هر ویژگی مد نظر پروژه بپردازد و به جای آنکه همهٔ طراحی‌های خود را مبتنی بر این نیازمندیها کامل کرده و سپس به کدنویسی برنامه مبتنی بر این طرحهای از اول مشخص شده بپردازد؛ به توسعهٔ تکرارپذیر و افزایشی می‌نگرد.

اسکرام تماماً معطوف به مسیرهایی جزئی در حل مسأله و ارزیابی مجدد آن مسأله پس از طی هر مسیر است.

بلاکهای جزئی با عنوان اسپرینت

ویژگیهای جزئی

تیم‌های کوچک

بلاکهای زمانی کوچک بیانگر چگونگی کار بر روی حل مسأله توسط تیم توسعه است. به هر اسپرینت می‌توان به صورت یک پروژهٔ آبشاری کوچک نگریست. زیرا در هر اسپرینت شما همهٔ کارهایی را که به طور عادی در یک پروژهٔ آبشاری انجام می‌دهید، اجرا می‌کنید با این تفاوت که فقط در مقیاسی کوچک‌تر آن را انجام می‌دهید. در هر اسپرینت، شما یک ویژگی را انتخاب کرده و نیازمندیهای آن ویژگی را جمع‌آوری کرده و به طراحی آن ویژگی مبتنی بر نیازمندیهای به دست‌آمده پرداخته و سپس کدنویسی کرده و آن خصیصه را با توجه به طراحی صورت گرفته، تست می‌کنید. شما در اسکرام برخلاف روش آبشاری، تلاش نمی‌کنید که همه چیز را پیشاپیش طراحی کنید. بلکه شما چیزی را انجام می‌دهید که نیاز است انجام شود. هدف هر اسپرینت انجام ارتقایی

(افزایشی) برای رسیدن به پروژهٔ نهایی است؛ اما افزایشی که به طور بالقوه قابل ارائه است.

حال چگونه می‌توان در هر اسپرینت تعداد زیادی پروژه‌های آبشاری را انجام داد، در حالی که قبلاً به سختی یک پروژهٔ آبشاری قابل انجام بود؟ جواب، انجام اسپرینتهایی با ویژگیهای کوچک است. ویژگیهای جزئی، قطعاتی از پروژه هستند که تلاش می‌کنند مسألهٔ خاصی را برای مشتری حل کنند. آنها درصدد این نیستند که کل برنامه را ایجاد کنند. ویژگیهای مدنظر یک پروژه به تکه‌های کوچکتری شکسته می‌شوند که هنوز قادر به تامین ارزش برای مشتری بوده و می‌توان آنها را به سرعت انجام داد. با هرچه بیشتر شدن این ویژگیهای کامل شده در پروژه، مشتری کم کم با نمای کامل برنامه مورد نظر مواجه شده و آن را ملاحظه می‌کند.

همه‌ی این موارد توسط یک تیم کوچکی از توسعه‌دهندگان، تست‌کننده‌ها و طراحانی که صرفاً به انجام پروژه مشغول هستند، انجام می‌شود. این تیم، یک تیم با قابلیت‌هایی چندگانه است که هر عضو آن با انجام تمام کارهای تیم آشناست. هر عضوی از آن ممکن است که در همه چیز بهترین نباشد؛ اما هرکس می‌داند که چگونه یک کار ضروری را برای تکمیل پروژه انجام دهد. نگریستن به آنها به عنوان یک تیم SEAL که هر عضو آن می‌داند که چگونه هرچیز مورد نیاز را انجام دهد، اما برای هرکاری کارشناسان مخصوص آن کار وجود دارد.

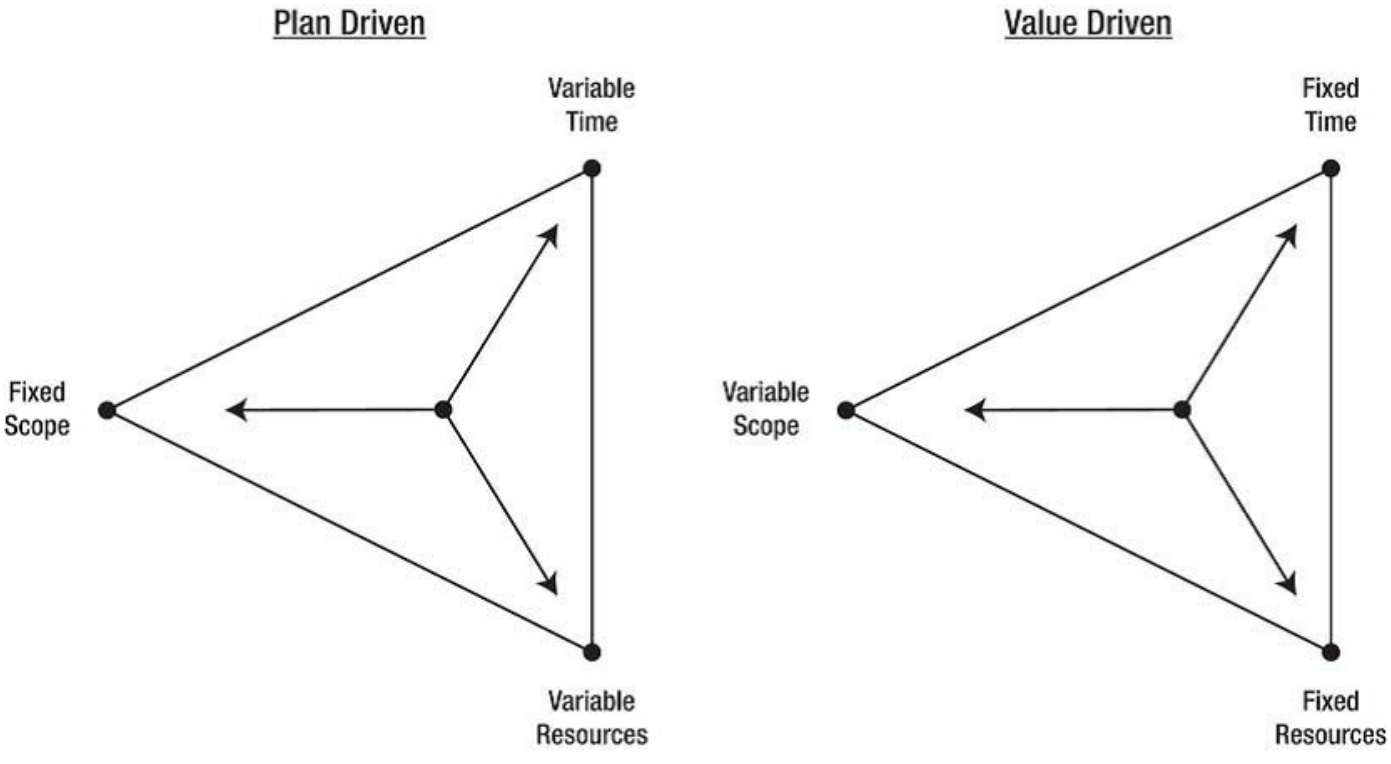
با انجام این کار در سطوح جزئی، مسائل این سطوح جزئی تا حدی شبیه مسائلی هستند که در انتهای پروژه در شیوهٔ آبشاری رخ می‌دهند. در واقع اسکرام به گونه‌ای کار می‌کند که بتواند تا آنجا که ممکن است سریعاً مشکلات و مسائل را نشان دهد. مشکلات قابل پنهان شدن نیستند؛ چراکه پروژه به سطوحی کوچک و قابل مدیریت، تجزیه شده است. هنگامیکه مشکلی بروز پیدا می‌کند، تا وقت پیدا شدن راه حل و حل شدن آن، موجبات دردسر تیم را فراهم می‌کند و آنها نمی‌توانند از مسأله چشم‌پوشی کنند، چون برای همه قابل رؤیت است.

نکتهٔ بسیارمهمی را باید دربارهٔ اسکرام فهمید و آن اینکه اسکرام مشکلات را هرچه زودتر، به تیم نشان می‌دهد؛ اما آنها را حل نمی‌کند.

اسکرام نه تنها ویژگیهایی را برای نمایش به مشتری توسط تیمهای فروش و بازاریابی تولید می‌کند، بلکه راه‌حلهایی را نیز به مشتری ارائه می‌دهد. چنین امری با اولویت‌بندی خصوصیت‌ها مطابق نیاز و خواسته‌های مشتری، صورت می‌گیرد. اگر مشتری‌ای تصور کند که ویژگی A باید از ویژگی B بسیار مهم‌تر باشد و توسعه دهنده، وقت زیادی را بر سر ویژگی B قبل از ویژگی A صرف کند، نمی‌تواند به نیاز مشتری به نحو مطلوبی، پاسخ‌گو باشد.

عوامل ثابت در مقابل عوامل متغیر

سه عامل یا قید کلیدی، برای هر پروژهٔ نرم‌افزاری وجود دارند: زمان، منابع و محدودهٔ پروژه. متأسفانه در یک زمان، هر سه عامل قابل جمع نیست. طبق شکل مثلثی زیر، در هر زمان می‌توان بر روی تاثیرات دو عامل کار کرد و آن دو عامل اتفاقی را که رخ می‌دهد، بر سومی دیکته می‌کنند.



در مدل توسعهٔ برنامه محور، حیطه و منابع پروژه، معمولاً عوامل ثابتند و زمان عامل متغیر است. در این حالت حیطهٔ پروژه بر منابع و زمان حاکم است. این حالت تا زمانی خوب است که شما در میانهٔ پروژه قراردارید. اما به مرور، رشد حیطهٔ پروژه، چهره نامطلوب کار را نمایان می‌سازد. در این هنگام محدودهٔ پروژه، گسترش خواهد یافت در حالی که نه منابع و نه زمان، متناسب با چنین تغییری، قابل تغییر نیستند. در این هنگام شما افراد بیشتری را به پروژه وارد می‌کنید، به این امید که به نتیجهٔ مناسبی در انتهای کار دست یابید.
در مدل توسعهٔ ارزش محور، منابع و زمان در مثلث ثابتند. شما از ابعاد تیمتان و سرعت انجام کارشان در اسپرینتهای قبلی آگاهید. در این حالت محدودهٔ پروژه در مثلث فوق، عنصر متغیر می‌شود. به عبارت دیگر منابع پروژه و زمان، تعیین‌کنندهٔ محدودهٔ پروژه هستند.

محصولات اسکرام

اسکرام سه خروجی دارد:

product backlog : مجموعه‌ای اولویت بندی شده از نیازمندی‌های سطح بالای سیستمی که در نهایت بایستی تحویل داده شود.

sprint backlog : مواردی از product backlog که قرار است در یک sprint انجام شوند.

نمودار burn-down :هدف نمودار burn-down، نمایش روند پیشرفت پروژه به صورت نموداری به اعضای تیم توسعه است که حاوی اطلاعاتی دربارهٔ کل زمان انجام کار، زمان تخمین‌زده شده، مقدار کارانجام شده و عقب‌ماندگی‌های پروژه است.

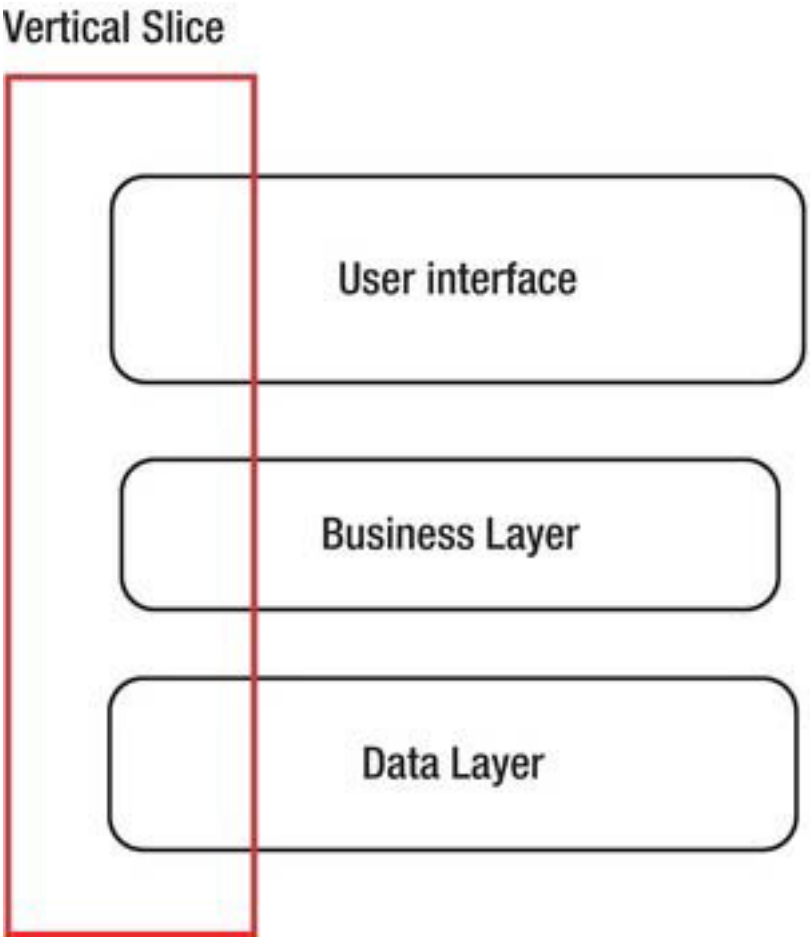
این خروجی‌ها، محصولات فعالیت‌های اسکرام هستند و به تیم در جهت‌یابی و شفافیت کار کمک می‌کنند. افزون بر این خروجی‌های اصلی خروجی‌های فرعی‌ای نیز از قبیل معیار پذیرش (الزاماتی که باید در حل یک مسأله برآورده شود تا بتوان آن را کامل شده تلقی کرد) وجود دارد.

Product Backlog

product backlog لیستی از همه کارهای باقی‌مانده در یک پروژه است که باید انجام شوند. این لیست نمایانگر نیازمندیها و خواسته‌های مشتری است. در قلب این لیست «داستان کاربر (user story)» یعنی مؤلفهٔ کلیدی اسکرام قرار دارد. این مؤلفه تعیین‌کنندهٔ ملاک افزایش ارزش در نزد مشتری بوده و آن چیزی است که توسعه دهنده تلاش می‌کند، ارائه نماید و توسط صاحب محصول (product owner) (یعنی کسی که نسبت به افزودن یا حذف داستان کاربر (user story)ها به لیست، پاسخ‌گو است) مدیریت می‌شود. product backlog به طور دائم توسط صاحب محصول و مشتری اولویت‌بندی می‌شود. این اولویت‌بندی دائمی امری کلیدی برای اسکرام است. این امر تضمین می‌کند که داستان کاربر (user story) که تعیین‌کنندهٔ بیشترین ارزش برای مشتریست، در صدر product backlog قرار گرفته باشد. با افزوده شدن یک داستان کاربر (user story) این مورد با سایر داستان‌های کاربر (user storyهای) پیشین مقایسه شده تا مشخص شود که در چه سطح ارزشی‌ای از نظر مشتری قراردارد. در طول یک اسپرینت، داستان کاربر (user storyها) را می‌توان به اسپرینت اضافه کرد. اما تا کامل شدن اسپرینت جاری، به تیم توسعه نشان داده نمی‌شود.

User Stories

همانطور که خاطرنشان شد، product backlog چیزی بیش از یک لیست اولویت‌بندی شده از داستان‌های کاربر (user storyها) نیست. یک داستان کاربر (user story)، یک کارت است که ارزش اضافه‌ای را برای مشتری توصیف می‌کند. داستان کاربر (user story) برای توسعه‌دهنده به منظور بیان ارزشی اضافه نوشته می‌شود. نکتهٔ کلیدی یک داستان کاربر (user story) خوب، این است که داستان کاربر (user story) بخشی عمودی از لیست است و بخش افقی ویژگی‌ای است که فقط به یک سطح، مانند سطح بانک اطلاعات یا سطح رابط کاربری اثر می‌گذارد. به عبارت دیگر قطعهٔ عمودی تمام سطوح را آن گونه که در شکل 3-2 نشان داده شده، متاثر می‌سازد. این کوچکترین مقدار کاری است که تمام سطوح یک محصول را تحت تاثیر قرارداده و برای مشتری ارزش ایجاد می‌کند. با نوشتن داستان‌های کاربر (user storyها) به گونه‌ای که در بخشهای عمودی جایز است، می‌توان قابلیت پایه‌ای را در اولین داستان کاربر (user story) ایجاد کرده و سپس به سادگی قابلیتی به این ویژگی به عنوان نیازهای مشتری اضافه کرد.



یک شیوهٔ اطمینان از اینکه داستان کاربر (user story) فایدهٔ قطعهٔ عمودی بودن در یک سیستم را داراست این است که مطمئن شویم با «INVEST» منطبق است. «INVEST» عبارت است از مخفف:
مستقل (Independent): باید خودبسنده باشد و به سایر داستانها وابسته نباشد.

قابل مذاکره (Negotiable): داستانهای کاربری که بخشی از یک اسپرینت هستند همیشه قابل تغییر و بازنویسی هستند.

با ارزش (Valuable): یک داستان کاربر باید به کاربر نهایی، ارزشی را ارائه دهد.

قابل برآورد (Estimable): همیشه باید بتوان اندازهٔ داستان کاربر را تخمین زد.

اندازهٔ مناسب (Sized appropriately): داستانهای کاربر نباید آن قدر بزرگ باشند که تبدیلشان به یک طرح یا وظیفه یا امر اولویت‌بندی شده با درجهٔ مشخصی ممکن نباشد.

قابل آزمون (Testable): داستان کاربر یا توصیفات مربوط به آن باید اطلاعات ضروری برای آزمودن آن را فراهم کنند.

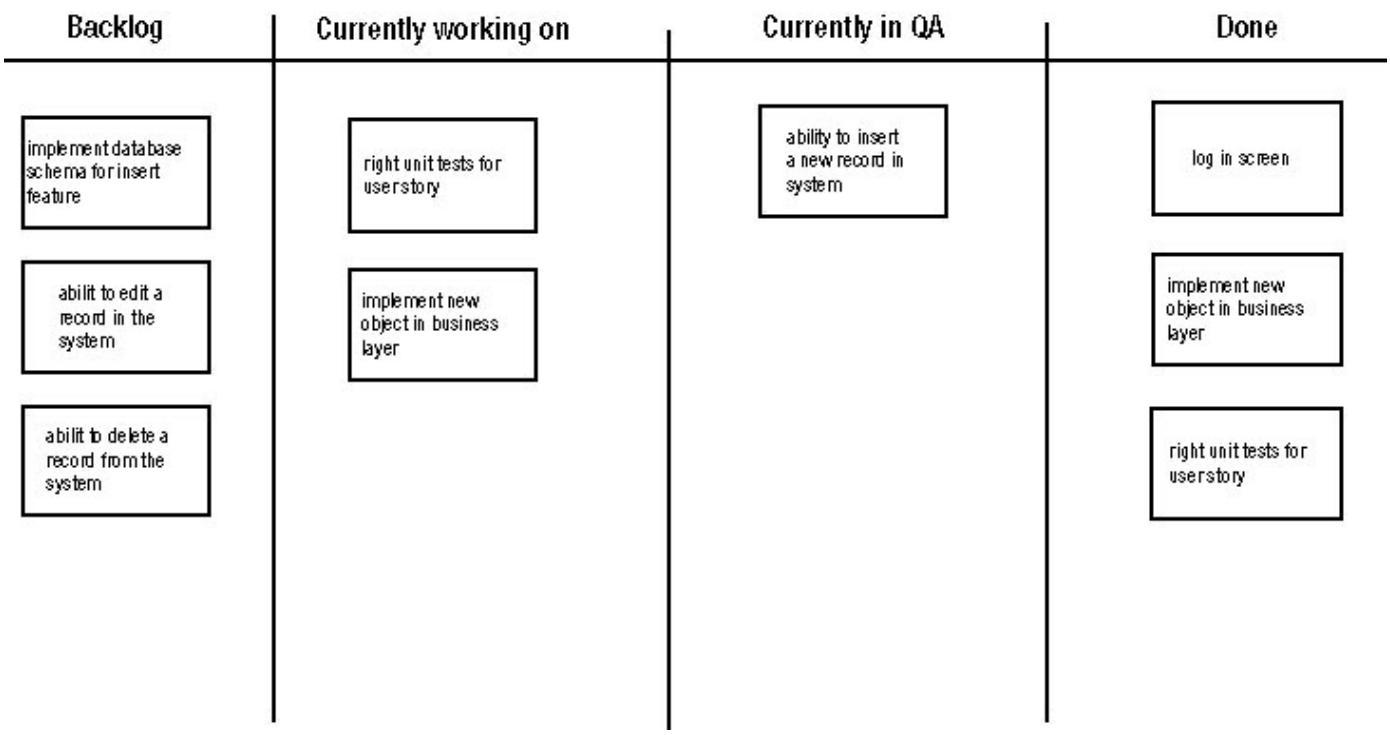
تعیین اندازهٔ backlog

تعیین اندازهٔ product backlog عبارت است از اندازه‌گیری سرعتی که تیم اسکرام می‌تواند مؤلفه‌های آن را ارائه کند. افراد در تخمین کار خوب عمل نمی‌کنند. همگی می‌دانیم که در تخمین دقیق اینکه چقدر طول می‌کشد تا یک کار را به طور کامل انجام دهیم، تا چه اندازه بد عمل می‌کنیم. تا کنون چند مرتبه این اتفاق افتاده است که از کسی بشنویم یا به خودمان بگوییم که 80 درصد کار را انجام داده‌ام و 20 درصد باقی‌ماندهٔ آن در یک ساعت انجام خواهد شد. اما هنوز بعد از دو روز انجام نشده است. افراد به طور

طبیعی بد تخمین می‌زنند.
ما ممکن است در تخمین زدن خوب نباشیم؛ اما در مقایسه کردن اشیاء با یکدیگر عالی هستیم. به عنوان مثال قادریم که با نگاه انداختن به دو دستور پخت غذا تشخیص دهیم که کدام‌یک پیچیده‌تر از دیگری است؛ بدون آنکه تخصصی در آشپزی داشته باشیم. به دو چیز نگاه می‌کنیم و تشخیص می‌دهیم که کدام‌یک بزرگتر از دیگری است. تخمین اندازهٔ backlog تماماً یعنی تصمیم‌گیری دربارهٔ پیچیدگی و مقدار کار لازم، نه اینکه چقدر طول می‌کشد تا این کار انجام شود. تخمین اندازه با تخمین برابر نیست.
ممکن است بپرسید که چگونه می‌توان زمان انجام برخی چیزها را اندازه گرفت؟ مدیری را در نظر بگیرید که می‌خواهد بداند چقدر طول می‌کشد تیم شما یک widget را تولید کند. شما می‌توانید تخمین زمان کامل شدن widget را از پیچیدگی widget تخمین بزنید. شما می‌توانید وقتی که تیم از یک اسپرینت فارغ شد، به آن اسپرینت نگاه کرده و محاسبه کنید که چقدر طول می‌کشد تا کار کامل شود. فقط پیچیدگی یک وظیفه‌ی مورد توجه تیم است.
اجازه دهید برای توضیح بهتر چگونگی تخمین مقدار کار مورد نیاز برای کامل شدن کار، این مسأله را با رنگ‌آمیزی خانه‌تان مقایسه کنیم. شما به فروشگاه رنگ فروشی رفته و چند سطل رنگ را برای رنگ‌آمیزی خانه می‌خرید. سپس از سه پیمانکار می‌خواهید که انجام این کار را برای شما تخمین بزنند. اولین پیمانکار به خانه‌ی شما آمده و دور خانه قدم زده و به سطلهای رنگی که خریده‌اید نگاه کرده و می‌گوید که وی با یک نردبان زنگ زده و برس‌های دستی و پسرکی لاغراندام به عنوان دستیارش، این کار را در ظرف دو روز انجام می‌دهد.
دومین پیمانکار دور خانه قدم زده و به سطلها نگریسته و می‌گوید که به تازگی نردبان و برس‌هایی خریداری کرده است و تیم محلی فوتبال در آخر هفته به وی کمک خواهند کرد. با این دستیاران و تجهیزات جدید، انجام این کار فقط یک روز به طول می‌انجامد.
سومین پیمانکار دور خانه قدم زده و به رنگها نگریسته و می‌گوید که وی صاحب یک دستگاه مکانیکی رنگ‌آمیزی است که باعث می‌شود انجام این کار حدود یک ساعت وقت بگیرد.
شما دربارهٔ این ماجرا و سه نوع تخمین از رنگ‌آمیزی خانه، چگونه فکر می‌کنید در صورتی که در هیچ کدام از این سه وضعیت، نه ابعاد خانه تغییری کرده است و نه مقدار رنگی که شما خریداری کرده‌اید. نکتهٔ این داستان در این است که بهترین چیزی که شما می‌توانید انجام دهید تخمین مدت زمان انجام کار نیست؛ بلکه به جای آن باید مقدار تلاشی را که منجر به اتمام کار خواهد شد، تخمین زد. با تخمین مقدار کار می‌توان مدت زمان انجام کار را به دست آورد.

Sprint Backlog

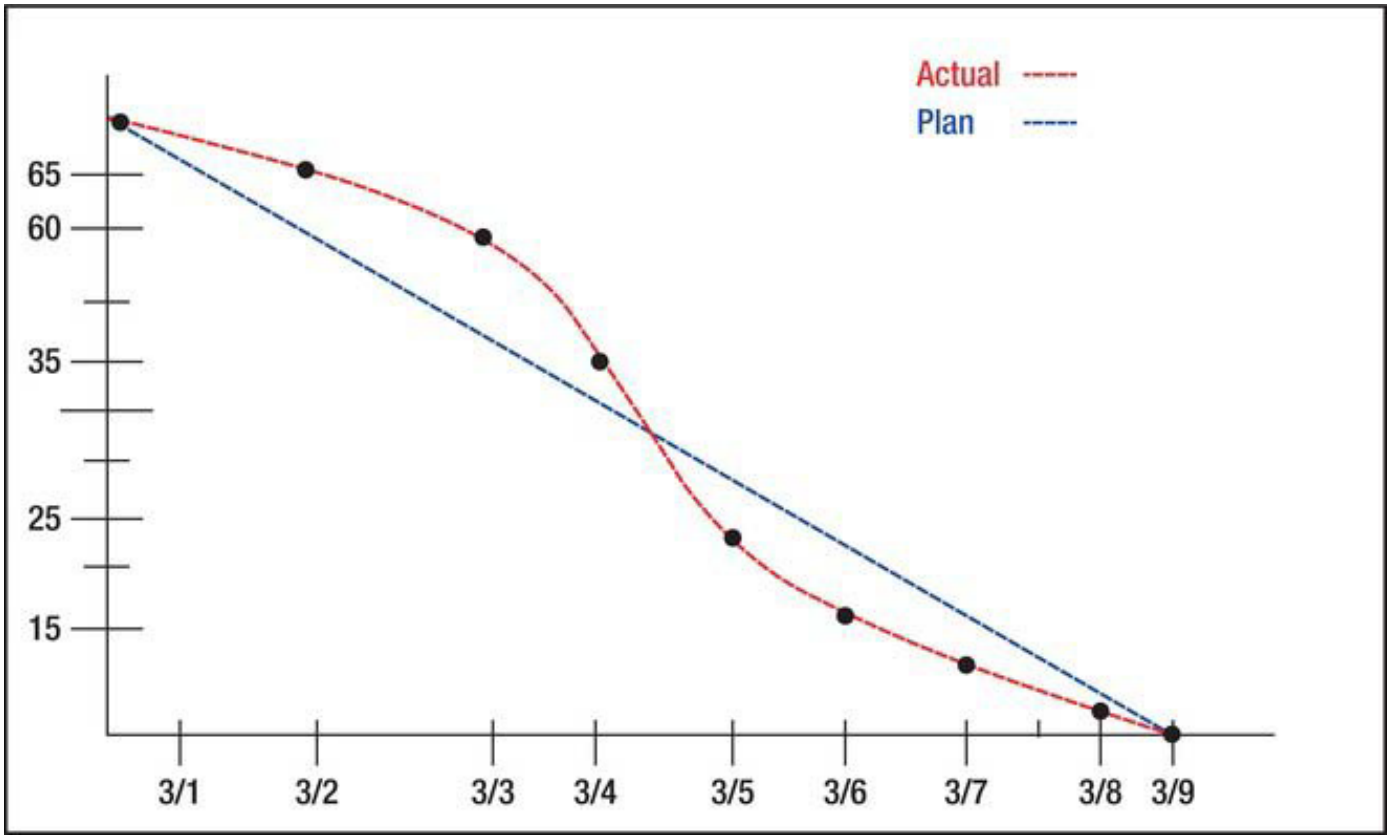
sprint backlog لیستی از همهٔ کارهای باقی‌مانده در یک اسپرینت است و باید توسط تیم انجام شود. sprint backlog زیرمجموعه‌ای از product backlog است. product backlog همهٔ داستانهای کاربران را که برای product مانده لیست می‌کند؛ اما sprint backlog حاوی همه‌ی داستان‌ها و وظایف باقی‌مانده برای اسپرینت است. نوعاً هنگامی که یک داستان کاربر برای یک اسپرینت انتخاب می‌شود، تیم، آن داستان کاربر را به تعدادی وظیفه تقسیم می‌کند.
یک وظیفه، تکهٔ کوچکی از داستان کاربر است که توسط هر عضو تیم قابل انجام است. مثلاً وظایفی از قبیل اجرای تغییرات بر روی بانک اطلاعاتی مورد نیاز یک داستان کاربر یا وظیفهٔ اجرای UI برای داستان کاربر. وظایفی که بر روی تابلوی وظایف – که با عنوان Kanban (معادل ژاپنی بیلبورد) نیز شناخته می‌شود- برای همهٔ تیم قابل رؤیت است. سایر مؤلفه‌های روی این تخته به همان ترتیب، حاوی اطلاعاتی دربارهٔ قرار ملاقاتهای جمع‌آوری نیازمندیها، کنترلهای بازبینی، تحقیقات، آزمون، طراحی و مراحل کد نویسی هستند. شکل 4-2 یک مثال را نشان می‌دهد.
اعضای تیم یک کارت از تخته برداشته و در طول اسپرینت اقدام به انجام وظیفه‌ای که روی کارت توصیف شده، می‌نمایند. در خلال مدتی که تیم بر روی وظایف کار می‌کند، سایر وظایف بروز پیدا کرده و تخمینهای اصلی مجدداً تنظیم می‌شوند. همهٔ اعضای تیم، در قبال به روز رسانی تابلو بر طبق اطلاعات جدید مقید خواهند بود.



sprint backlog اطلاعات مورد نیاز نمودار burn-down را فراهم می‌کند. در پایان هر اسپرینت، sprint backlog خالی می‌شود. هر آیتم باقی مانده‌ای در backlog به product backlog برگردانده شده و مجدداً در کنار سایر داستانهای کاربری موجود در product backlog بعلاوهٔ داستانهای کاربری تازه وارد شده، اولویت‌بندی می‌شود.

Burn-down chart

نمودار burn-down شیوه‌ای بصری برای دنبال کردن چگونگی پیش‌روی یک اسپرینت است. این نمودار کار باقیماندهٔ اسپرینت را در هر روز، به صورت گرافیکی همانند شکل 5-2 نشان می‌دهد. معمولاً این نمودار در یک محیط عمومی نمایش داده می‌شود تا هرکسی بتواند آن را ببیند. این کار به ارتباطات میان اعضای تیم و هرکس دیگری در سازمان کمک می‌کند. این نمودار همچنین می‌تواند به عنوان نشانگر وجود یک مسأله در اسپرینت عمل کند که تیم ممکن است بخاطر آن نتوانند به تعهد خود عمل کنند.



معیار پذیرش

اگرچه sprint backlog ، product backlog و نمودار burn-down بخشهای اصلی اسکرام هستند، معیار پذیرش خروجی جانبی بسیار مهمی از فرآیند اسکرام است. بدون معیار پذیرش خوب، یک پروژه محکوم به شکست است.

معیار پذیرش ضرورتاً شفاف کنندهٔ داستان است. چنین معیاری مجموعه‌ای از گامهای مختلف را در اختیار توسعه دهنده می‌گذارد که پیش از آنکه کار تمام شده تلقی شود، باید انجام دهد. معیار پذیرش، توسط صاحب محصول (product owner) به کمک مشتری ایجاد می‌شود. این معیار انتظار از داستان کاربر را تنظیم می‌کند. استفاده از این معیار درجای خود نقطهٔ شروع خوبی برای نوشتن تستهای خودکار یا حتی توسعهٔ آزمون محور توسط توسعه‌دهنده است. بدین طریق، توسعه‌دهنده چیزی را تولید می‌کند که مشتری بدان نیاز داشته و آن را می‌خواهد.

دیگر مزیت معیار پذیرش وقتی آشکار می‌شود که یک ویژگی در طول یک اسپرینت کامل نشده و نیاز است تا از اسپرینتها خارج شود. در چنین موردی تیم می‌تواند معیار پذیرش را به عنوان ابزاری به کار گیرد تا بفهمد که داستان کاربر چگونه به قطعات کوچکتری تقسیم شود تا کماکان ارزشی را برای مشتری فراهم کرده تا بتواند در یک اسپرینت کامل شود.

نقش‌های اسکرام

اسکرام بین افرادی که نسبت به پروژه متعهد هستند و افرادی که فقط ذینفع محسوب میشوند، تمایز قابل توجه‌ای قائل است.
مشهورترین روش برای توضیح این مفهوم تعریف حکایت "Pig & Chicken" میباشد؛ یک خوک و یک جوجه در حال قدم زدن بودند که، یک دفعه جوجه به خوک گفت که: "چرا یک رستوران افتتاح نکنیم؟" خوک هم نگاهی به جوجه کرد و گفت:"ایده خوبی است ، اسم آن را چه بنامیم؟" جوجه کمی در مورد این مسئله فکر کرد و گفت:"چرا اسمش را 'گوشت ران خوک و تخم مرغ ها' نگذاریم؟"
خوک جواب داد:"فکر نمیکنم جالب باشد چون من متعهد خواهم بود به کار، ولی تو فقط درگیر کار خواهی بود".
بنابراین Pigs همان افراد متعهد به پروژه هستند که وظیفه ساخت، تست، گسترش و توزیع را ایفا میکنند. Chickens در طرف دیگر همان افرادی هستند که کمتر به پروژه تعهد دارند. این افراد همان stackeholderها و یا ذینفعانی هستند که از پروژه منفعت

می‌برند، اما در مقابل تحویل پروژه مسئول و پسخگو نیستند.

Pig Roles

نقش‌های عنوان شده در زیر جز نقش‌های Pig هستند که تیم اسکرام را نیز تشکیل میدهند:

Scrum Master

Product Owner

Delivery Team

Scrum Master

اگر تیم را موتور پروژه‌ی اسکرام در نظر بگیریم، اسکرام مستر روغنی است که موتور را در حال اجرا نگه می‌دارد. او مسئول این است که مطمئن شود فرآیند اسکرام تفهیم شده و دنبال میشود. اسکرام مستر تسهیل کننده‌ی جلسات تیم و حذف موانعی است که امکان دارد تیم در دوره‌ای از انجام کار خود با آن مواجه شد. او مطمئن خواهد بود که هیچ مانعی به عنوان بازدارنده از رسیدن به اهداف تیم در مقابل آنها وجود ندارد و تیم را از حواس پرتی‌های خارجی ایزوله نگه می‌دارد تا مطمئن شود اعضای تیم دقیقا کاری را به آنها سپرده شده است انجام میدهند. اسکرام مستر با بخش‌های مختلف تیم، از صاحبان محصول گرفته تا تست کنندگان وذینفعان کسب و کار در تعامل است، تا مطمئن شود که تمام اعضای تیم برای پروژه مفید هستند و تمام دست آوردهای مشترک در اسپرینت را به اشتراک میگذارند. اسکرام مستر را یک مدیر پروژه معمول فرض نکنید؛ چون نقشی که او ایفا میکند بیشتر از نقش یک مدیر پروژه است. مشخصه کلیدی اسکرام مستر "رهبر خدمتگزار است". او رئیس تیم نیست ولی به تیم کمک میکند تا به چیزی که نیاز دارند در این اسپرینت دست یابند. اسکرام مستر به تیم کمک میکند تا کار را به سمتی که خروجی با ارزشی برای مشتری دارد، متمایل کنند. زمانی که مسئله‌ای در داخل تیم بوجود آید، به اسکرام مستر انتقال داده میشود تا این تضاد را مدیریت کند. مواقعی هم وجود دارند که اسکرام مستر در نقش فرمانروا، ایفای نقش میکند. وقتی که یکی از مسئولیت‌های اسکرام مستر مطمئن شدن از این است که تمام روشهای اسکرام توسط اعضای تیم دنبال میشوند، هرمسئله و حمله‌ای در برابر چارچوب اسکرام باید توسط اسکرام مستر رفع شود. این شانس خوش و اینچنین اتفاقی به ندرت خواهد افتاد.

Product Owner

صاحب پروژه یا محصول، مسئول مشخص کردن مشتری و افزایش مقدار کاری است که تیم انجام میدهد. او با مشتریان برای مشخص کردن اینکه خواسته آنها چیست، جلسه تشکیل داده و این نیازها را اولویت بندی میکنند و تیم هم بر روی آیتم هایی با بیشترین ارزش برای مشتری، کار خواهد کرد. صاحب محصول همچنین product backlog را مدیریت کرده و تنها شخصی است که میتواند user storyها را برای انتقال به اسپرینت، اولویت بندی کند. مسئولیت‌های صاحب محصول در طول اسپرینت از سری مسئولیت‌های نقش Pig به سری مسئولیت‌های نقش Chicken تغییر میکند. یکی دیگر از نقش‌های حیاتی او این است که به عنوان نماینده‌ی مشتری، برای تیم است. صاحب محصول خیلی شبیه به اسکرام مستر میباشد ولی در این بین یک تفاوت اصلی در طبیعت نقش‌های آنها وجود دارد: اسکرام مستر به دنبال بهترین جذابیتهای مورد علاقه تیم در یک اسپرینت بوده؛ در حالی که صاحب محصول به دنبال بهترین جذابیتهای مطلوب مشتری در یک اسپرینت است . در یک تیم اسکرام، صاحب محصول، نقشی است که نمیتوان آن را دست کم گرفت. اگر صاحب محصول کسی باشد که نتواند به طور دقیق نیازهای مشتری را به تصویر بکشد، در نتیجه پروژه شکست خواهد خورد.

صاحب محصول کلید اجرایی یک محصول است که ارزشی را برای مشتری و موفقیتی را برای تیم به ارمغان می‌آورد.

Delivery Team

تیم تحویل، گروهی است از افراد که مسئول ارائه واقعی محصول هستند. این تیم معمولا شامل دو تا ده نفر از افراد و همچنین ترکیبی از برنامه نویسان، تست کننده‌ها، طراحان محصول نهایی و اعضایی از سایر نظام‌های ضروری، میباشد. تیم برای انتقال user story و وظایف مرتبط با آن به مرحله بعد بر روی تخته Kanban، تا مرحله‌ی اتمام، بر روی اسپرینت‌ها کار میکند. مشخصه‌ی کلیدی تیم تحویل این است که آنها به صورت یک واحد خود سازمانده می‌باشند. هیچ رهبری در جمع آنها وجود ندارد و همه به صورت گروهی تصمیم میگیرند که در هر اسپرینت به انجام چه چیزی میتوانند متعهد شوند. اعضای تیم بار دیگر تصمیم خواهند گرفت که چه ابزاری برای موفقیت پروژه نیاز دارند. چنین سطحی از استقلال در متدولوژی آبشاری بی‌سابقه است! تیم تحویل برای بهینه سازی انعطاف پذیری و بهره وری در نظر گرفته شده‌اند.

تیم اسکرام ترکیبی از افرادی است با توانمندیهای گوناگون که هرکدام باید با تمام چشم اندازهای محصول در مراتب مختلف آشنا

باشند. هریک از اعضای تیم به تنهایی در همه‌ی مباحث نرم افزار ماهر نیستند، اما هر یک از آنها دانش عمومی در همه مباحث را دارند و در قسمت کلی از مفاهیم محصول هم متخصص هستند. تیم تحویل به همراه اسکرام مستر و صاحب محصول، برای تکمیل user storyها و به سرانجام رساندن هر اسپرینت، باهم کار میکنند. اسکرام مستر با آمادگی به دنبال بهترین جذابیتهای مورد علاقه تیم در یک اسپرینت است؛ در حالیکه صاحب محصول با آمادگی به دنبال بهترین جذابیتهای مطلوب مشتری در یک اسپرینت است. با وجود این دو نقش، تیم میتواند محصولی که مشتری میخواهد را بسارد.

فعالیت‌های اسکرام

شامل فعالیت‌هایی که در مرکز کانونی اسکرام و در سراسر طرح ریزی، بررسی و نشست‌ها و جلسات میباشد.

Sprint Planning

قبل از شروع هر sprint، جلسه طرح ریزی برای مشخص کردن اینکه کدام امکان و ویژگی در این sprint قرار بگیرد، برگزار میشود. ویژگی‌ها و امکانات از لیست pb ای (product backlog) که توسط صاحبان (یا صاحب) محصول اولویت بندی شده است، انتخاب خواهند شد. برای بار اول که این نشست و جلسه برای یک پروژه برگزار شود، pb ساخته می‌شود. شما می‌توانید این قسمت را 0 sprint در نظر بگیرید. user stories (گزارشات کاربر) انتخاب شده توسط صاحب محصول، برای قرار گرفتن در print ، به تیم داده میشود و آنها از طریق یک ابزار کاری بنام Planning Poker، گزارشات مذکور را برای نشان دادن پیچیدگی یک گزارش وابسته به گزارشات دیگر در گروه گزارشات، تغییر اندازه و تغییر حجم میدهند. بار دیگر user story هایی که به اندازه هستند، توسط تیم به وظیفه‌هایی قابل نسبت دادن به یک فرد تبدیل میشوند و یک زمان تخمینی که نشان دهنده‌ی زمان اتمام برای هر وظیفه است، برای هر وظیفه در نظر گرفته میشود. بار دیگر که تمام این کارها انجام شد ، اعضای تیم به لیست کامل کارهایی که برای sprint در نظر گرفته شده‌اند، نگاه خواهند کرد و اگر بتوانند تا اتمام sprint کار را تمام کنند، تصمیم خواهند گرفت که آن را انجام دهند. این تصمیم گیری به صورت زیر است:

به وسیله 5 انگشت قرار است نظرات خود را ارائه دهند؛ به طوری که اگر عضوی دست خود را با یک انگشت بالا ببرد، بدین معنی است که او درطرح پیشنهادی خیلی تردید دارد و اگر دستی با 5 انگشت بالا رود، به این معنی است که این عضو، به شدت به طرح پیشنهادی مطمئن است. اگر هیچ دستی با تعداد انگشت 1 یا 2 از بین دست‌های بالا رفته دیده نشود، لذا تیم به انجام آن کار در sprint جاری متعهد خواهد شد. ولی اگر دستی با تعداد انگشت 1 یا 2 از بین دست‌های بالا رفته دیده شود، در آن صورت اعضای تیم در مورد دلیل رای عضوی که رایی با ارزش 1 یا 2 داده است، بحث خواهند کرد و با دیگر اعضای تیم برای تحویل گزارشات و وظایف موجود در اسپرینت، متعهد میشوند.

sprint backlog از گزارشات کاربر و وظایفی که باید در sprint تکمیل شوند، ساخته شده است. تمام اعضای تیم در کنار اسکرام مستر و صاحبان محصول در نشست برنامه ریزی اسپرینت درگیر هستند. بار دیگر جلسه برنامه ریزی متشکل از اعضای تیم و بدون صاحبان محصول برای بحث در مورد طراحی سطح بالای سیستم برگزار خواهد شد.

planning poker

planning poker یک بازی است که اعضای تیم را تشویق میکند تا ارزیابی درستی در مورد پیچیدگی گزارش کاربری (user story) که در ارتباط با سایر گزارشات (stories) است، داشته باشند. ابزارهای مورد نیاز برای این بازی خیلی ساده هستند: شما میتوانید از دست خود استفاده کنید؛ یا حتی میتوانید مجموعه کارت‌های Planning Poker را برای انجام بازی، خریداری کنید. برای انجام این بازی، صاحب محصول، گزارش کاربر (user story) را خوانده و برای تیم توضیح خواهد داد. تیم برای پرسیدن سوال در باره این گزارش کاربر، آزاد است. وقتی که تمام سوالات پاسخ داده شدند، اسکرام مستر از اعضای تیم خواهد خواست که یک عدد را به صورت خصوصی که به بهترین شکل پیچیدگی user story را ارئه میکند، تعیین کنید. توجه داشته باشید برای اینکه این انتخاب به صورت سهوی تحت تاثیر انتخاب سایر اعضا نباشد، باید برای دیگران آشکار نشود. بار دیگر اسکرام مستر از همه میخواهد تا شماره‌های خود را برای همه آشکار کنند. اگر تمام اعضای تیم، یک شماره یکسان را تعیین کرده باشند، آن شماره به user story مورد نظر نسبت داده شده و همه سراغ user story بعدی میروند.

اگر شماره‌ها باهم یکسان نباشند، عضوی با بیشترین و کمترین شماره، انتخاب شده و از آنها خواسته میشود دلیل تعیین شماره خود را شرح دهند. بعد از بحث، یک راند دیگر از بازی بین اعضایی که یک شماره را برای user story انتخاب کرده‌اند، انجام میشود. این کار تا زمانیکه تیم در یک شماره اتفاق نظر داشته باشند، ادامه خواهد داشت. به طور متوسط برای رسیدن به یک شماره یکسان، بیشتر از 3 راند طول نخواهد کشید. اگر بعد از 3 راند باز هم به شماره‌ای که همه با آن موافق هستند، دست نیابند، ما به اسکرام مستر پیشنهاد میکنیم که میانگین را انتخاب کرده و سراغ user story بعدی بروند.

Daily Stand Ups

در طول یک اسپرینت، تیم، اسکرام مستر و صاحب محصول، برای حضور در جلسات روزانه که یکبار در هر روز و در یک مکان و زمان یکسان برای بحث در مورد موضوع‌هایی که موجب مانع از اتمام کار میشوند، متعهد میشوند. در جلساتی که برگزار میشود، همه به صورت ایستاده بوده و زمان آن بیشتر از 15 دقیقه طول نخواهد کشید. هرکسی که به نوعی ذینفع در پروژه هستند، برای

حضور در جلسات دعوت میشوند. هر چند فقط افرادی که رده بندی شده‌اند، اجازه صحبت در این جلسات را خواهند داشت. در این جلسات، هر عضو تیم به 3 سوال زیر پاسخ خواهد داد:
شما چه چیزی را از دیروز تا حالا انجام داده‌اید؟

شما چه برنامه‌ای برای امروز دارید؟

آیا شما مشکل دیگری که مانع رسیدن به هدفتان باشد، ندارید؟ چه جریانی باعث ایجاد این موانع شده‌اند؟ آیا میتوان مانع را حذف کرد یا باید تشدید شود؟

Sprint Review

جلسه "بررسی اسپرینت" در پایان اسپرینت برگزار میشود. هدف از آن ارائه گزارشات کاربری (user stories) هست که در طول اسپرینت تکمیل شده‌اند. تیم، صاحب محصول و اسکرام مستر به همراه سایر ذینفعان، مخصوصا مدیران و مشتریان، در این جلسه حضور خواهند داشت. این بررسی شامل یک دموی غیررسمی از نرم افزار توسعه داده شده در اسپرینت، میباشد. این جلسه دموی محصول، فرصتی است برای مشتری تا بازخوردهای خود از محصول را به تیم توسعه انتقال دهند. هدف اصلی از این بازنگری، نمایش محصول با کارکرد واقعی است. این جلسه با اصل "بالاترین اولویت ما عبارت است از راضی کردن مشتری با تحویل سریع و مداوم نرم افزار با ارزش" چابک در یک راستا میباشد.

عنوان: Pro Agile .NET Development With Scrum - قسمت دوم
نویسنده: غلامرضا ربال
تاریخ: ۱۳۹۴/۰۶/۰۴ ۱۱:۴۵
آدرس: www.dotnettips.info
گروه‌ها: Agile, Scrum

داستان‌های کاربر

توسعه‌دهندگان، ویژگی‌های مورد نظر پروژه را با جمع‌آوری نیازمندی‌ها، در قالب داستانهای کاربر احصاء می‌کنند و به هرکدام متناسب با پیچیدگی‌اش امتیازی اختصاص می‌دهند. با لیستی از داستان‌های دارای ابعادی مشخص و بودجه و زمان مورد نیاز برای هرکدام، مشتریان قادر به این انتخابند که کدام ویژگی‌ها در تکرار (iteration) بعدی باقی بماند. مشخص‌کردن بودجه و زمان، یعنی تعیین حجم کاری که تیم توسعه برای انجام آن ویژگی، نیاز می‌داند. برآورد بودجهٔ مورد نیاز تکرار اول به صورت تجربی خواهد بود و ممکن است این تخمین در ابتدا نادرست باشد؛ اما با شروع تکرار بعدی درست خواهد شد. در پایان هر تکرار، امتیازات به دست آمده از داستان‌های کامل شده را جمع کنید. مجموع این امتیازات، نشانگر سرعت شما خواهد بود. این سرعت شاخص خوبی جهت چگونگی بودجه‌بندی مرحلهٔ بعد است. هنگامیکه امتیازات جمع‌آوری شده به حد مطلوبی رسید، «سرعت پیشرَوی»، شاخص مناسب دیگری برای بودجه‌بندی است که عبارت است از متوسط سرعت سه تکرار آخر.

با این کار شما به دیدگاه مناسبی از فاز برنامه‌ریزی دست پیدا می‌کنید. حال اجاز دهید نگاه دقیق‌تری به شیوه‌های برنامه‌ریزی داشته باشیم.

برنامه‌ریزی (planning game) دو فاز دارد: فاز شناسایی و فاز برنامه‌ریزی. در فاز شناسایی، توسعه‌دهندگان و مشتریان را دور هم جمع می‌کنند تا دربارهٔ نیازمندیهای سیستم در حال طراحی، گفتگو کنند. به خاطر داشته باشید که این کار تا وقتی انجام می‌شود که به ویژگی‌هایی (features) کافی برای شروع انجام کار برسیم و البته واضح است که چنین لیستی از ویژگی‌های احصاء شده، هرچقدر هم که تلاش شود، کامل نخواهد بود. مشتریان اغلب اوقات، خواسته‌ی خود را یا نمی‌دانند یا نمی‌توانند به خوبی توضیح دهند. بنابراین معمولاً این لیست به مرور تغییر می‌کند. در ضمن آنکه برخی ویژگیها دقیق‌تر می‌شود، مواردی نیز ممکن است به لیست افزوده شوند یا حتی می‌توان برخی ویژگی‌های نامربوط را از لیست حذف کرد. در مرحلهٔ شناسایی، ویژگی‌ها به داستانهای کاربر تجزیه شد و ثبت می‌شوند.

یک داستان کاربر عبارت است از توصیفی کوتاه از یک ویژگی که نمایانگر یک واحد ارزش کسب و کار برای مشتری است. داستانهای کاربر از زبان کاربر بیان شده‌اند و قالب نوشتاری زیر را دارند:

به عنوان «نوع کاربر»، من می‌خواهم «یک فعل» تا «منفعتی برای کسب و کار»

یا به صورت:

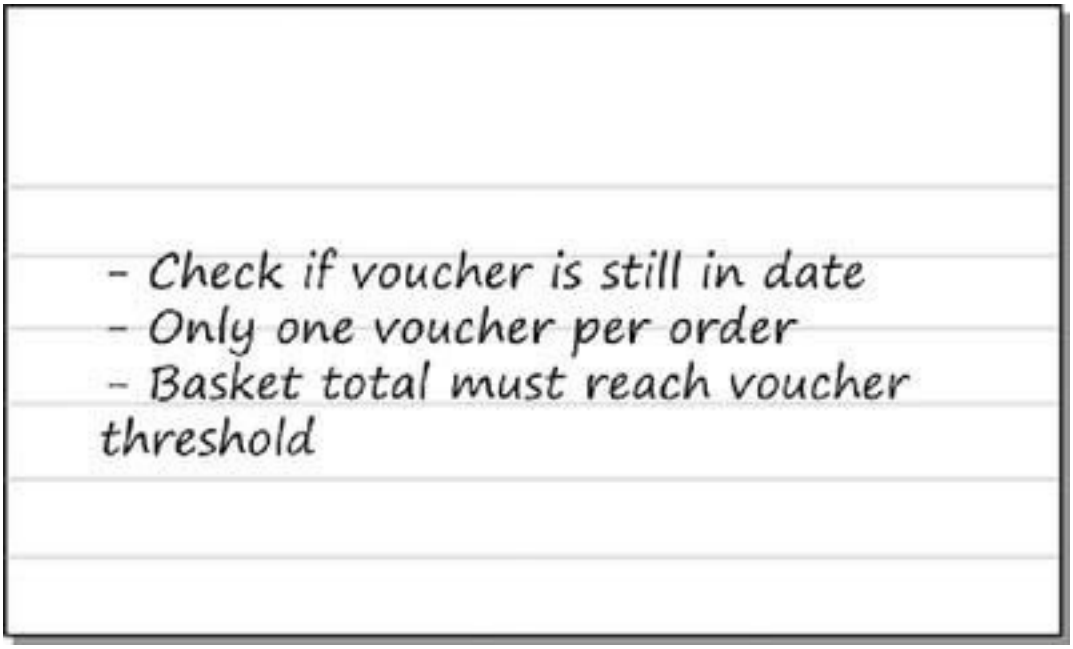
به منظور «یک دلیل» به عنوان «نقش کاربر» من می‌خواهم «یک فعل»

داستانهای کاربر معمولاً در جلسه‌ی گفتگو با مشتری بر روی کارت‌های راهنما نوشته شده و در آن از واژگان و ادبیاتی استفاده می‌شود که برای مشتری قابل فهم باشد. ممکن است چنین بیاندیشید که ثبت نیازمندی‌ها، خلاف مزیت‌های چابک‌سازی است؛ چرا که تولید نرم‌افزار کارآمد و چابک مبتنی بر مستندسازی گسترده و فراگیر خواهد بود. در واقع، داستان‌های کاربر به طور ساده فقط یادآورندهٔ جزئیات بیشتری از گفتگوی انجام شده‌اند که به عمد به‌صورت کوتاه و دقیق نوشته شده‌اند. فهم دقیق‌تر جزئیات کار، مستلزم ارتباط بیشتر میان توسعه‌دهندگان و مشتری است. در واقع همسو با این اصل چابک که می‌گوید: «مؤثرترین و کارآمدترین شیوهٔ انتقال اطلاعات در میان تیم توسعه و به خارج از آن، گفتگوی چهره به چهره است.»

هنگام احصاء ویژگی‌های پروژه تحت عنوان داستان‌های کاربری، از اصول INVEST (که پیش‌تر گفته شد) جهت کنترل مناسب بودن این داستانها استفاده کنید. شکل 2-3 مثالی از یک داستان کاربر را که توصیف‌کنندهٔ ویژگی «افزودن یک بن تخفیف به سبد خرید»

است، نشان می‌دهد. «تخفیف گرفتن»، یک منفعت کسب و کار است برای عامل (actor) اصلی، یعنی مشتری. «یک بن تخفیف به سبد بیفزا» نام فرآیند یا «use case» مربوط است.

از معیار پذیرش (acceptance criteria) نیز می‌توان در هنگام تولید داستان‌ها استفاده کرد. معیار پذیرش را می‌توان در پشت کارت داستان، آن طور که در شکل 3-3 نشان داده شده است، نوشت. استفاده از طرف مقابل کارت این اجازه را می‌دهد که اعضای تیم و مشتریان، اطلاعات خودشان را در یک جا جمع کنند.



معیار پذیرش همچنین به تشخیص جزئیات بیشتر یا شناسایی وابستگی‌ها کمک می‌کند. مثلاً در شکل 3-3 تعریف «in date» چیست و چه چیزی حدود یک بن تخفیف را مشخص می‌کند؟ معمولاً باید حداقل سه معیار پذیرش وجود داشته باشد. در فصل بعد در یک مطالعهٔ موردی، مطالب بیشتری را دربارهٔ داستانهای کاربر خواهید آموخت.

هنگامیکه تیم و مشتریان حس‌کنند که حدود 75 درصد از ویژگی‌های اصلی احصاء شده است، توسعه‌دهندگان ابعاد داستان‌ها را تخمین زده و آنها را برای اولویت‌بندی توسط مشتری آماده می‌کنند.

تخمین

شکی در آن نیست که تخمین‌زدن کار سختی است. تخمین‌زدن هم دانش است هم هنر. تخمین‌زدن در یک پروژهٔ تازه شروع شده، بسیار سخت است زیرا مجهولات بسیاری در آن وجود دارد.

یکی از روش‌های تخمین گروهی، روش «Planning Poker» نام دارد. در این روش همه‌ی اعضای فنی تیم، متشکل از توسعه‌دهندگان نرم‌افزار، تحلیل‌گران، متخصصان امنیت و زیرساخت، مشارکت می‌کنند. نقش مشتری در این حالت پاسخ‌گویی به سؤالات احتمالی اعضای تیم است تا ایشان بهتر بتوانند تخمین بزنند.

شیوهٔ انجام کار به این صورت است که عضوی از تیم، یک داستان کاربر را برداشته و آن را برای تیم توضیح می‌دهد. تیم دربارهٔ آن ویژگی با مشتری گفتگو کرده تا جزئیات بیشتری را دریابد. وقتی که تیم به درک خوبی از آن رسید، رأی‌گیری آغاز می‌شود. هر عضو تیم با یک کارت، از مجموعه‌ای ازکارتهایی با شماره‌های 0، 1 ، 2، 3، 5، 8، 13، 20، 40 و 100 رأی خود را اعلام می‌کند.

تیم باید از داستانی شروع کند که نسبتاً کوچک و ساده باشد. این داستان به عنوان مبنا انتخاب می‌شود. هر تخمین داستان کاربر، باید به نسبت این داستان کوچک انجام شود. اگر داستان مبنا به خوبی انتخاب نشود، بقیهٔ تخمین‌ها نادرست خواهد بود.

اگر همه‌ی اعضای تیم به یک صورت رأی دهند، آن رأی، تخمین آن داستان خواهد شد. اگر اختلاف آراء وجود داشت، ناظر یعنی کسی که رأی نمی‌دهد، از افرادی که بالاترین و پایین‌ترین امتیاز را داده‌اند، می‌خواهد که علل خود را توضیح دهند. سپس تیم مجدداً گفتگو کرده و دوباره رأی‌گیری می‌کند. طبق تجربه، خوب است که زمان معقولی، برای هر گفتگو در نظر گرفته شود.

اگر تخمین یک داستان به دلیل فقدان دانش فنی، بسیار سخت بود، مناسب است که این داستان کنار گذاشته شود و داستان دیگری برای برطرف کردن مشکل ناآشنایی با دانش فنی مورد نظر فراهم شود. بدین ترتیب تیم توسعه در موقعیت بهتری می‌تواند نسبت به داستان جدید تخمین بزند.

داستان‌هایی که بیش از یک هفته کار نیاز داشته باشند با عنوان داستانهای حماسی (epic stories) شناخته می‌شوند و معمولاً برای تخمین بسیار بزرگ هستند. در واقع، این داستان‌ها به چند داستان کوچک‌تر که قابل فهم‌تر و به آسانی قابل تخمین باشند، تجزیه می‌شوند. این بدان معناست که ایجاد یک داستان کاربر از تعداد انبوهی ویژگی موجب کاهش کارآیی خواهد شد.

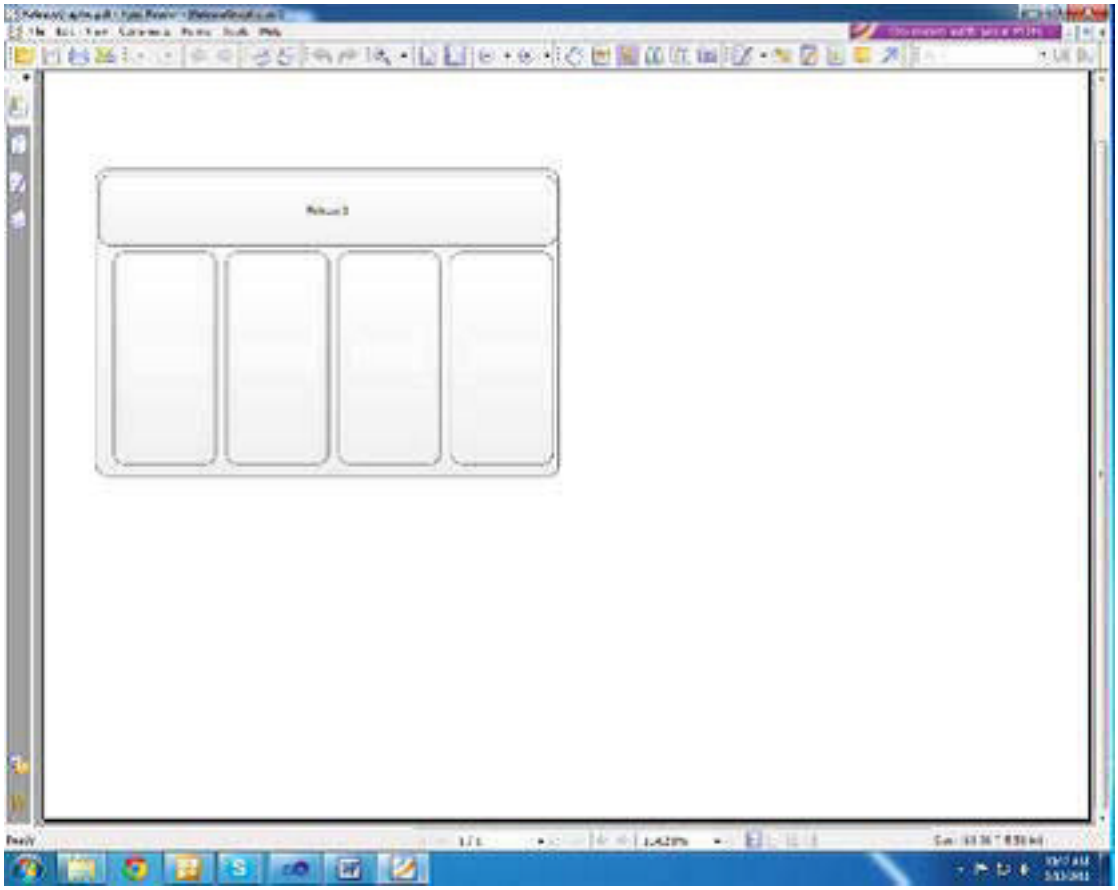
تخمین در تیمی که افراد آن تاکنون با همدیگر سابقهٔ همکاری نداشته باشند، خیلی پایین یا خیلی بالاست. اما با استمرار هر تکرار و تجربه و دانش بیشتر افراد، تخمین داستان‌ها بهتر می‌شود.

استفاده از ابزار Planning Poker مزایای بسیاری دربردارد. دقت تخمین بالا می‌رود؛ زیرا مسأله از منظر تخصص‌های گوناگون مورد بررسی قرار گرفته است. همچنین به تیم کمک می‌کند که هم رأی شوند و گفتگو میان اعضاء را تسهیل می‌کند. پس از آنکه داستان‌ها تخمین زده شدند، مشتری و صاحب محصول با تیم توسعه در تولید چگونگی انتشار نسخه‌ها، همکاری می‌کنند.

برنامه انتشا ر

اگرچه کدهای قابل ارسال، قابلیت انتشار در پایان هر تکرار را دارند، اما یک پروژه XP در چند سری منتشر شده است. یک نسخهٔ منتشرشده، متشکل از تعداد مناسبی داستان برای عرضهٔ ارزش کسب وکاری است که به کوچک نگه داشتن آن کمک می‌کند.

بسیار مناسب است که یک موضوع یا هدف خاص را در ضمن هرنسخهٔ انتشار، مد نظر قرار داد تا کمک کند که هر نسخهٔ انتشار بر برخی ارزشهای کسب و کاری متمرکز شده و آن را هدایت کند. معمولاً یک نسخهٔ انتشار، متشکل از چهار تکرار است؛ همانطور که در شکل 4-3 نشان داده شده است.



در برنامه‌ریزی نسخه‌های انتشار، طول یک تکرار نیز تعیین می‌شود که معمولاً بین دو تا چهار هفته است. مطابق تجربه، اگر محیط کار شما دچار بی‌نظمی و اختلالات دائمی است، می‌توانید دورهٔ تکرار را به یک هفته محدود کنید.

یکی از پروژه‌هایی که ما بر روی آن کار می‌کردیم، برنامه‌ای بود که نگهداری آن بسیار سخت و فوق‌العاده ناپایدار بود. مشتری مکررا با تیم تماس گرفته و اشکالات بحران‌ساز و ایراداتی را که مخل برنامه بودند، گزارش می‌کرد. در ابتدای کار دوره، تکرار ما هفتگی بود. به همین دلیل چون حلقهٔ بازخوردگیری‌مان کوچک بود، می‌توانستیم بر پایدارسازی پروژه در هر دوره کاری تمرکز کنیم. هنگامی که محصول به پایداری مناسب‌تری رسید و تماس‌های مشتری کم شد، قادر شدیم تا در هر دوره، دقت بیشتری بر روی مسائل به خرج دهیم.

اگر قصد دارید به صورت دقیق بر روی حلقهٔ بازخورد متمرکز شوید، دوره‌ی تکرار یک هفته‌ای، مدل خوبی است. اما این مدل سربار زیادی را به دلیل ضرورت تقسیم داستانهای کاربر باید به بخش‌های کوچک‌تری تا آن اندازه که در یک دوره تکمیل شوند، بر پروژه تحمیل می‌کند. در ادامه خواهیم گفت که هر تکرار شامل برنامهٔ ملاقات و بازبینی نیز هست.

بعد از مدتی که تیم با فرآیند کار آشناتر شد و نوبت به مشکلات با اولویت کم‌تر رسید، می‌توان دورهٔ تکرار را دو هفته‌ای در نظر گرفت. اما اگر پروژه به گونه‌ای است که ویژگی‌های بزرگ‌تر را نمی‌توان به موارد کوچک‌تری که قابل انجام در دوره‌های یک هفته‌ای باشد، تجزیه کرد و تیم هنوز در حال یادگیری است، دوره‌های بلندمدت‌تر قابل پذیرش است.

مشتری با توجه به طول دورهٔ تکرار و بودجهٔ داستان آغازین، انتخاب می‌کند که کدام داستان در هنگام انتشار نسخهٔ اوّل، در تکرار اوّل کامل شود.

این مشتری است که داستان‌ها را به گونه‌ای اولویت‌بندی می‌کند تا مشخص شود که کدام‌یک بیشترین ارزش کسب و کار را فراهم می‌کند. از آنجایی که مشتری مسؤول داستانهای کاربر است، تیم باید به وی توضیح دهد که داستانهایی وجود دارند که صرفاً باید به جهت دلایل فنی ایجاد شوند.

معمولاً باید به داستانهای کاربری‌ای که مستلزم ریسک بالا بوده یا دربرگیرندهٔ مجهولات زیادی باشند، بیش از یک یا دو تکرار اختصاص داد.

برنامهٔ تکرار

مشتری داستان‌هایی را که می‌خواهد در تکرار باشند، انتخاب می‌کند. برای هر داستان کاربر، مجموعه‌ای از معیارهای پذیرش، تعریف شده است. همان طور که متوجه شده‌اید ما در هر فاز، وقت بیشتر و بیشتری را صرف جمع‌آوری جزئیات هر داستان کاربر کرده و بصورت عمیق‌تری در آن غور می‌کنیم. این کار مفید است، زیرا اگر یک داستان کاربر ایجاد شده در ابتدای پروژه، ممکن است بعداً به عنوان داستانی کم اهمیت یا غیر مهم دیده‌شود و بدون آنکه وقت خاصی برای آن صرف شده باشد، کنار گذاشته شود. اما اگر در ابتدای کار وقت زیادی صرف دقیق‌تر کردن داستان‌های کاربر شود و بعداً بعضی از آنها کنار گذاشته شوند، در واقع وقت تلف شده است. بنابراین دقیق‌تر کردن یک داستان در جایی که مورد نیاز است، باید اتفاق بیفتد. در سطح برنامهٔ تکرار، مجموعه‌ای از معیارهای پذیرش را برای هر داستان کاربر تعریف می‌کنیم. معیار پذیرش به توسعه‌دهنده کمک می‌کند تا بداند که یک داستان کاربر به طور کامل انجام می‌شود. این معیارها به صورت مؤلفه‌هایی از بافرض/هنگامی که/درنتیجه، نوشته می‌شود.

مثالهای زیر چگونگی انجام این کار را توصیف می‌کند:

عنوان ویژگی : افزودن کالایی به سبد

به عنوان یک مشتری می‌خواهم بتوانم کالایی را به سبدم اضافه کنم؛ به نحوی که قادر باشم به خرید خود ادامه دهم.

سناریو : سبد خالی

با فرض اینکه یک سبد خالی دارم، در نتیجه جمع تعداد کالایی که برای سفارش در سبد من وجود دارد، صفر است.

سناریو : افزودن یک کالا به سبد

با فرض اینکه یک سبد خالی دارم هنگامی که کالایی با شناسهٔ 1 به سبدم اضافه می‌کنم، در نتیجه جمع کالاهای قابل سفارش در سبدم 1 می‌شود.

سناریو : افزودن کالاهایی به سبد

با فرض اینکه یک سبد خالی دارم، هنگامی که کالایی با شناسهٔ 1 و کالایی با شناسهٔ 2 به سبدم اضافه می‌کنم، در نتیجه جمع کالاهای قابل سفارش در سبدم 2 می‌شود.

سناریو : دو بار افزودن یک کالا

با فرض اینکه یک سبد خالی دارم هنگامی که کالایی با شناسهٔ 1 به سبدم اضافه می‌کنم و هنگامی که کالایی با شناسهٔ 1 را مجدداً به سبدم اضافه می‌کنم، در نتیجه تعداد کالاهای با شناسهٔ 1 در سبد من باید 2 باشد.

سناریو : افزودن یک کالای تمام شده به سبد

با فرض اینکه یک سبد خالی دارم و کالایی با شناسهٔ 2 در انبار وجود نداشته باشد، هنگامی که من کالایی با شناسهٔ 2 را به سبد خودم اضافه می‌کنم، در نتیجه جمع تعداد کالای قابل سفارش در سبد من باید 0 باشد و به کاربر، موجود نبودن آن کالا را هشدار دهد.

یک آزمون پذیرش (acceptance) به زبان متعارف در قوانین کسب و کار نوشته می‌شود. در مثال سبد خرید، این سؤال پیش می‌آید که چگونه می‌توان یک محصول را از سبد کالا، حذف کرد و اگر یک جنس اکنون در انبار نیست و کاربر پیام هشدار دریافت کرده است، در ادامه چه اتفاقی باید بیفتد؟ سناریوها به تیم در کشف ملزومات کسب و کار و تصریح آن‌ها کمک می‌کند.

این سناریوها توسط توسعه‌دهنده به عنوان نقطهٔ شروع آزمونهای واحد در توسعهٔ آزمون محور و رفتار محور استفاده می‌شود. سناریوها همچنین در آزمودن معیارهای پذیرش به توسعه‌دهنده کمک کرده و توسعه‌دهنده و تست‌کننده را قادر می‌سازند که بر روی اتمام داستان اتفاق نظر داشته باشند.

بعد از آنکه سناریوهای معیار پذیرش تعیین شد، تیم توسعه، هر داستان را به تعدادی وظیفه تقسیم می‌کند و وظایف مرتبط به یک داستان، در تابلوی وظایف قرارگرفته و تیم توسعه تخمین‌های خود را در قالب یکی از واحدهای اندازه‌گیری، مثلاً نفرساعت اعلام می‌کند. شکل 5-3 یک تابلوی وظیفه را نمایش می‌دهد.

به عنوان مثال وظایف می‌توانند شامل ایجاد طرح یک بانک اطلاعاتی برای یک داستان یا یکپارچه‌سازی آن با بخشی موجود در سیستم باشند. وظایف شامل مؤلفه‌های فنی مانند تهیهٔ گزارش از زیرسیستم‌ها یا چارچوب مدیریت استثنائات نیز می‌باشد. اغلب این‌گونه وظایف نادیده‌گرفته می‌شود. یک داستان کاربر با وظایف گوناگونی گره خورده است. مثلاً:

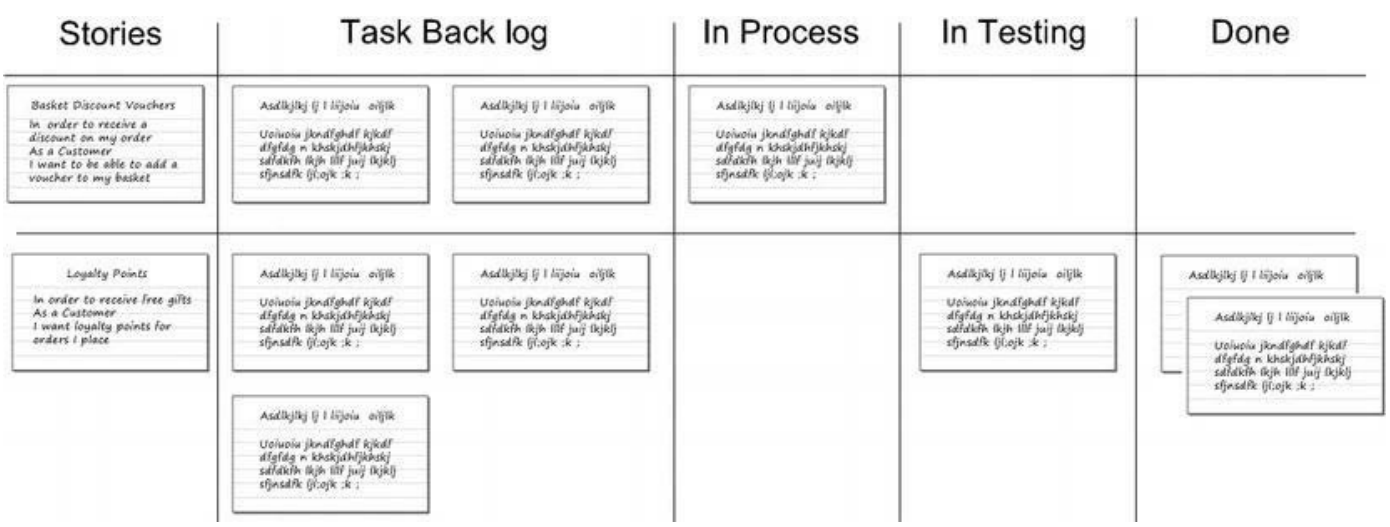
داستان کاربر : به عنوان یک کاربر می‌خواهم بتوانیم یک کاربر را مدیریت کنم.

وظایف زیر از این داستان قابل استخراج است:

طرحی برای بانک اطلاعات جهت ذخیره‌سازی اطلاعات کاربر ایجاد کن.

یک کلاس کاربر، برای مدیریت کاربر از درون برنامه ایجاد کن.

هر عضو تیم می‌تواند بر روی هر وظیفه‌ای که بر روی تخته است، کار کند. هنگامیکه یک عضو گروه، وظیفه‌ای را برمی‌دارد، باید نشانی از خود روی کارت آن وظیفه قراردهد (مثلاً حروف اوّل اسمش) تا بقیهٔ افراد بدانند که وی بر روی آن وظیفه، مشغول به کار است. معمولاً اما نه همیشه، یک توسعه‌دهنده همهٔ وظایف مربوط به یک داستان را برمی‌دارد. این کار بدین معناست که آن توسعه‌دهنده با پشتیبانی تیم، مسؤول اتمام آن کار است.

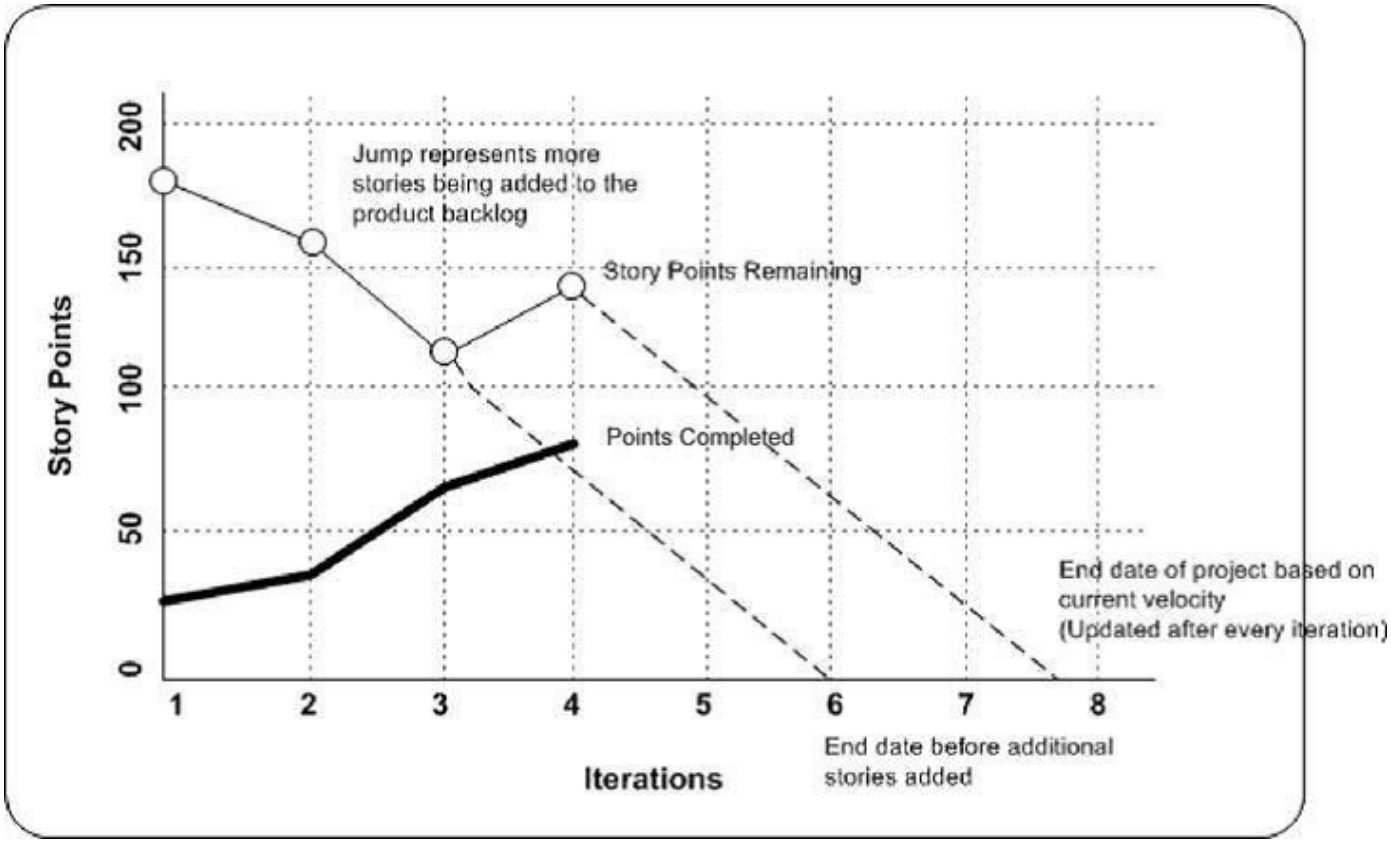


به محض اینکه یک تکرار آغاز شد، داستان‌هایی را که کار بر روی آنها شروع شده است، دیگر نمی‌توان تغییر داد. این مهم است که برنامهٔ تکرار را در حین انجام آن، تغییر ندهید؛ زیرا این کار منجر به سوئیچنگ زمینه (context switching) می‌شود. برای توسعه‌دهندگان سوئیچینگ زمینه، هم به لحاظ زمانی و هم به لحاظ مالی، بسیار پرهزینه است.

به جای آنکه تلاش کنید در ضمن یک تکرار، تغییراتی را ایجاد کنید، مشخص کنید که آیا این کار اضافه، یا داستان اضافه را می‌توان تا تکرار بعدی به تعویق انداخت. مشتریان یا مدیران معمولاً می‌توانند چنین تعویقی را بپذیرند؛ زیرا این پذیرش مستلزم به تاخیر انداختن کار، مثلاً تا یک ماه دیگر نیست. اگر این کار جدید را که اضافه شده است، نمی‌توان به تعویق انداخت، باید ریسک خارج ساختن کدهای موجود و رفتن به سمت کدنویسی برای کارکرد جدید را به همراه تیم بررسی کرد. همچنین تیم باید بداند که اگر کار اضافه‌ای به یک دورهٔ تکرار افزوده شد، بخشی از کارهای این دوره باید به تکرار بعدی موکول شوند. قاعدهٔ کلی این است که اگر چیزهای جدیدی به کار وارد شد و تعویق آن ممکن نبود، باید کارهایی با همان ابعاد یا بزرگتر از تکرار، خارج شود.

سرعت به ما نشان می‌دهد که تیم چه حجم کاری را در طول یک دوره کامل کرده است. از سرعت، در برنامه‌ریزی تکرارهای آتی استفاده می‌شود. برای درک چگونگی سرعت کار از نمودار burn-down استفاده می‌شود. یک نمودار burn-down (شکل 6-3) داستان‌های باقی‌ماندهٔ یک پروژه و داستانهای تکمیل شده را در یک تکرار نمایش می‌دهد. سرعت در پایان هر تکرار محاسبه می‌شود و تعریف آن عبارت است از تعداد داستان‌های تکمیل شده در آخرین تکرار. بر اساس سرعت کنونی و تعداد داستان‌های باقی‌مانده، می‌توان تخمین زد که چقدر طول می‌کشد تا همه‌ی داستانها تکمیل شود. همانند آنچه در شکل 6-3 با خط چین نمایش داده شده است.

نمودار burn-down ابزار خوبی برای فهم آن است که آیا تیم می‌تواند پروژه را در زمان مقتضی به پایان برساند یا خیر و اگر نمی‌تواند، مدیر چگونه باید نسبت به آن تصمیم‌گیری کند. آیا افراد بیشتری باید به پروژه وارد شوند؟ آیا باید از ویژگی‌های مدنظر پروژه کاهش داد، یا باید زمان پایان کار را تغییر داد؟



در طول یک تکرار، هر روز باید گفتگوهایی سرپایی با حضور همهٔ اعضای تیم انجام شود و مشکلاتی که ممکن است باعث به تأخیر افتادن ارائه کار شود، مورد بحث و بررسی قرار گیرد و همچنین تیم، لیست وظایف و تخته آن را به‌روز کرده تا پیشرفت یا موانع آن به وضوح قابل رؤیت باشند.

با تشکر از آقای سید مجتبی حسینی