

آنچه باید درباره فرایند توسعه نرم افزار بدانید!

محمد حسین احمدی

smhahmadi@chmail.ir

معرفی اینجانب

• محمدحسین احمدی

- فارغ التحصیل کارشناسی مهندسی IT و ارشد نرم افزار شریف
- پایان نامه کارشناسی ارشد زیر نظر دکتر رامسین در آزمایشگاه مهندسی
متدولوژی دانشکده مهندسی کامپیوتر
- سابقه برنامه نویسی C++، جاوا و PHP در کار
- چهار سال مسئول اصلاح فرایند توسعه نرم افزار (و مدیر گروه بهره‌وری تولید)
در شرکت امن افزار گستر شریف
- مدیر گروه پارس گیت در شرکت امن افزار گستر شریف

- «فرایند توسعه نرم افزار» یعنی چه؟
- چرا مهم است؟
- آشنایی مختصر با متدولوژی های مشهور
- آشنایی با RUP
- آشنایی با توسعه چابک نرم افزار
- آشنایی با چارچوب اسکرام

منظور از «فرایند توسعه نرم افزار» چیست؟

- هر کاری که انجام می دهیم تا یک نرم افزار تولید شود، به مشتری تحویل داده شود و پس از تحویل نگهداری شود
- از کارهای «بزرگ» مثل برنامه ریزی، جمع آوری نیازمندی ها از مشتری، طراحی، برنامه نویسی، آزمون و تحویل گرفته تا موارد ظاهراً جزئی مثل نحوه نوشتن کد توسط برنامه نویسان، نحوه مرور کد توسط سایر برنامه نویسان، محل نشستن اعضای تیم، چگونگی نوشتن آزمون های خودکار و ... ذیل موضوع «فرایند توسعه نرم افزار» می گنجند.
- اگر فرایند توسعه نرم افزار مدوّن باشد، «متدولوژی» نامیده می شود.

مثال‌هایی از فرایندهای توسعه نرم‌افزار

- مدل آبشاری: نیازمندی‌ها، طراحی، پیاده‌سازی، آزمون، تحویل و نگهداری
- مدل تکراری (iterative): انجام گام‌های فوق در «تکرار»های متوالی
- متدولوژی RUP: ترکیبی از فازهای آبشاری و مدل تکراری
- متدولوژی‌های چابک: اضافه کردن فیچرهای جدید به ترتیب و با کیفیت بالا به نرم‌افزار و تحویل مکرر به مشتری
- فرایند دلبخواهی: سه برنامه‌نویس دور هم جمع می‌شوند، بدون هیچ برنامه‌ریزی یا طراحی خاصی گد می‌زنند و سعی می‌کنند بعد از مدتی آن را به مشتری بفروشند

مثال‌هایی از تصمیم‌های فرایندی در توسعه نرم‌افزار (۱)

- تحقیقات شرکت IBM نشان داده Formal Technical Review به شدت باعث افزایش کیفیت محصول پروژه‌های نرم‌افزاری می‌شود.
- در مدیریت پروژه به سبک PMBOK، استخراج «ساختار شکست کار» (WBS) در هنگام برنامه‌ریزی توصیه می‌شود.
- بسیاری از تیم‌ها از Git Workflow برای مدیریت کد استفاده می‌کنند.
- استفاده از UML برای مدل‌سازی نیازمندی‌ها و طراحی در بسیاری از پروژه‌های نرم‌افزاری رایج است.
- بسیاری از مدیران پروژه کم‌تجربه تلاش می‌کنند از تماس مستقیم توسعه‌دهندگان با مشتریان جلوگیری کنند.

مثال‌هایی از تصمیم‌های فرایندی در توسعه نرم‌افزار (۲)

- در روش‌های چابک توصیه اکید می‌شود که تمام اعضای تیم در یک محل بنشینند.
- در روش‌های سنتی مدیریت پروژه، تلاش می‌شود نیازمندی‌های پروژه در ابتدای پروژه با کارفرما نهایی شود.
- در روش‌های نوین «تحويل پیوسته» (Continuous Delivery) تمام آزمون‌ها به طور خودکار انجام می‌شوند و کد برنامه‌نویسان در چند دقیقه در صورت پاس کردن تمام تست‌ها، به طور خودکار در سیستم اصلی نصب می‌شود.
- برای ثبت و حفظ دانش اعضای یک تیم نرم‌افزاری، استفاده از «ویکی» بسیار رایج است.
- در روش‌های سنتی مدیریت پروژه، وظیفه هر عضو تیم را مدیر پروژه تخصیص می‌دهد و پیگیری می‌کند.

چرا فرایند توسعه نرم افزار؟

- بعد از مسائل تجاری و مهارت و تجربه افراد درگیر، موفقیت یک محصول یا پروژه نرم افزاری را فرایند توسعه آن نرم افزار مشخص می کند.
- یک فرایند توسعه نرم افزار خوب می تواند بازدهی یک تیم را از نظر سرعت و کیفیت نرم افزار خروجی چندین برابر افزایش دهد.
- فرایند توسعه نرم افزار روی رضایت مشتری و ذی نفعان، کیفیت خروجی، رضایت و انگیزه توسعه دهندگان، سرعت کار و قابل پیش بینی شدن کار توسعه نرم افزار تأثیر می گذارد.

فرایند توسعه نرم افزار به خصوص چه زمان هایی مهم می شود؟

- وقتی نیازی که قرار است پاسخ داده شود، ابهام و عدم قطعیت بیشتری دارد
- وقتی تعداد اعضای تیم رو به افزایش می گذارد
- وقتی محصول بزرگ تر و پیچیده تر می شود
- (در صورتی که هیچ یک از موارد فوق صدق نمی کنند) وقتی تیم مهارت و تجربه کمتری دارد

پروژه‌های نرم‌افزاری معمولاً شکست می‌خورند

- آمار گزارش CHAOS از Standish Group نشان می‌دهد

- فقط ۱۶٪ از پروژه‌های نرم‌افزاری موفق هستند

- ۵۳٪ با مشکلات (در محدوده، کیفیت، زمان یا هزینه) به پایان می‌رسند

- ۳۱٪ کلاً قبل از اتمام لغو می‌شوند

- هر پروژه به طور متوسط دو برابر زمان تخمینی اولیه طول می‌کشد

- آمار موفقیت پروژه‌های بزرگ آی تی در یک دهه: ۶/۴٪

(ComputerWorld, 2013)

مشکلات رایج در پروژه‌های نرم‌افزاری (۱)

- مشکلات کیفی آخرین لحظه معلوم می‌شود
 - مشتری: «ولی این چیزی نیست که ما می‌خواستیم»
 - مسئول کنترل کیفیت: «پنجاه تا باگ داریم»
 - «ویژگی ایکس کار نمی‌کند، نمی‌دانم چرا»
- رفع مشکلات کیفی قابل برنامه‌ریزی نیست!
 - چون
 - تعداد و حجم مشکلات کیفی از قبل قابل پیش‌بینی نیست
 - تخمین زمان مورد نیاز برای فهمیدن علت باگ تقریباً غیرممکن است
 - رفع باگ ممکن است خود باعث باگ جدید شود
 - برنامه رفع مشکلات کیفی: «تا وقتی باگ داریم، به آزمون و رفع باگ ادامه می‌دهیم. مدت: نامعلوم»

مشکلات رایج در پروژه‌های نرم‌افزاری (۲)

- نیازهای کارفرما مبهم است یا مرتب تغییر می‌کند
 - «کارفرما تازه یادش افتاده که ویژگی ایکس را هم می‌خواهد»
 - «کارفرما تکلیفش با خودش معلوم نیست»
- در اواخر پروژه فشار کار بسیار زیاد می‌شود
- وابستگی به نیرو باعث ایجاد بحران می‌شود
 - «فقط فلانی قسمت ایکس را بلد است، ولی پذیرشش آمده و دارد می‌رود خارج»

مشکلات رایج در پروژه‌های نرم‌افزاری (۳)

- در کل پیش‌بینی و برنامه‌ریزی بسیار سخت است

- نه کارفرما می‌داند پروژه چقدر طول می‌کشد و چقدر نیرو می‌برد، نه پیمانکار

- باعث بی‌اعتمادی بین کارفرما و پیمانکار می‌شود

- «این قسمت اقلأ شش ماه کار می‌برد، کارفرما اصرار دارد که در سه ماه تمامش کنیم»

- «دو ماه است که می‌گویند ۹۰ درصد از کار تمام شده»

- پیچیدگی ذاتی و فشار ناشی از زمان‌بندی غلط و خوش‌بینانه، باعث باگ زیاد و افت روحیه نیروها می‌شود

- پیچیدگی ذاتی باعث تصمیمات غلط (به‌خصوص توسط مدیر پروژه) و در نتیجه رشد بی‌اعتمادی در تیم می‌شود

- ارتباط ناکافی درون‌تیمی و بین تیم با ذی‌نفعان، باعث اشتباه در برداشت و نبود درک مشترک می‌شود

مثال: تأثیر فرایند توسعه نرم افزار روی کیفیت

- فرایند توسعه نرم افزار مهم ترین عامل در تضمین کیفیت نرم افزار است.
 - خیلی فرق می کند که کاربر نهایی فقط در آغاز و پایان پروژه درگیر آن شود یا در طول پروژه درگیر باشد!
 - خیلی فرق می کند که برای تضمین کیفیت، متکی به آزمون دستی در پایان پروژه باشید یا آزمون خودکار در طول پروژه!
 - خیلی فرق می کند که کیفیت مسئولیت آزمونگرها باشد یا مسئولیت کل تیم!
 - خیلی فرق می کند که برای تضمین کیفیت فقط از آزمون استفاده کنید یا از مرور (Review) یا سایر تکنیک ها هم استفاده کنید!

بازی!

- اثر نرخ بازخورد روی کیفیت کار

تاریخچه‌ای مختصر از متدولوژی‌های توسعه نرم‌افزار

- متدولوژی‌های پیش از شیء‌گرایی (اواخر دهه ۱۹۷۰): طراحی ساخت‌یافته، SSADM
- متدولوژی‌های شیء‌گرایی اولیه (اواخر دهه ۱۹۸۰): Shlaer-Mellor, Coad-Yourdon, RDD, Booch, OMT, OOSE
- متدولوژی‌های شیء‌گرایی یکپارچه (اواسط دهه ۱۹۹۰): OPM, Catalysis, OPEN, RUP
- متدولوژی‌های چابک (اواخر دهه ۱۹۹۰): DSDM, Crystal, Scrum, XP
- متدولوژی‌های ناب و چابک مدرن (دهه ۲۰۰۰): Kanban, DevOps, Continuous Delivery
- چارچوب‌های مقیاس‌پذیری چابک (اواخر دهه ۲۰۰۰): LeSS, SAFe, DAD, Nexus

تاریخچه‌ای مختصر از متدولوژی‌های توسعه نرم‌افزار

- متدولوژی‌های پیش از شیء‌گرایی (اواخر دهه ۱۹۷۰): طراحی ساخت‌یافته، SSADM
- متدولوژی‌های شیء‌گرایی اولیه (اواخر دهه ۱۹۸۰): Shlaer-Mellor, Coad-Yourdon, RDD, Booch, OMT, OOSE
- متدولوژی‌های شیء‌گرایی یکپارچه (اواسط دهه ۱۹۹۰): OPM, Catalysis, OPEN, RUP
- متدولوژی‌های چابک (اواخر دهه ۱۹۹۰): DSDM, Crystal, Scrum, XP
- متدولوژی‌های ناب و چابک مدرن (دهه ۲۰۰۰): Kanban, DevOps, Continuous Delivery
- چارچوب‌های مقیاس‌پذیری چابک (اواخر دهه ۲۰۰۰): LeSS, SAFe, DAD, Nexus

بازی!

- شبیه‌سازی ارتباط مشتری با تیم توسعه

فرایندهای چابک توسعه نرم افزار

- فرایندهای چابک توسعه نرم افزار تا حد زیادی در پاسخ به متدولوژی های سنگین وزن نظیر RUP پدید آمدند
- تأکید بر بازخورد سریع و انعطاف در مقابل تغییرات، توجه به نیاز واقعی مشتری، کار تیمی و مشارکتی، «توسعه دهنده پسند» یا «عضو تیم-پسند» کردن فرایند و کم کردن تشریفات
- تغییر فضای توسعه نرم افزار از «شبه سایر مهندسی ها» به «مهارت-هنر-تجربه حرفه ای»

مانیفست توسعه چابک

We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:

Individuals and interactions over processes and tools

Working software over comprehensive documentation

Customer collaboration over contract negotiation

Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

اصول چابکی

۱. بالاترین اولویت ما جلب رضایت مشتری با تحویل زود و مداوم نرم افزاری ارزشمند می باشد.

۲. استقبال از تغییر نیازمندی ها، حتی در اواخر فرآیند توسعه. فرآیند های چابک، تغییر را در جهت مزیت رقابتی مشتری مهار میکنند.

۳. تحویل زود به زود نرم افزار قابل استفاده دو، سه هفته یک بار تا دو ، سه ماه یک بار با ترجیح بر فاصله های زمانی کوتاه تر.

۴. ذی نفعان کسب و کار و توسعه دهنده ها می بایست به صورت روزانه در طول پروژه با هم کار کنند.

۵. پروژه ها را بر دوش افراد با انگیزه بنا کنید. فضای لازم را به آنها بدهید و از نیازهای آن ها پشتیبانی کنید و به آنها اعتماد کنید تا کارها را انجام دهند.

۶. کارآمدترین و موثرترین روش انتقال اطلاعات به تیم توسعه و تبادل آن در میان اعضای تیم ، گفتگوی چهره به چهره است.

۷. نرم افزار قابل استفاده اصلی ترین معیار سنجش پیشرفت است.

۸. فرآیند های چابک توسعه پایدار را ترویج می دهند. حامیان مالی ، توسعه دهندگان و کاربران باید بتوانند سرعت پیشرفت ثابتی را برای مدت نامحدودی حفظ کنند.

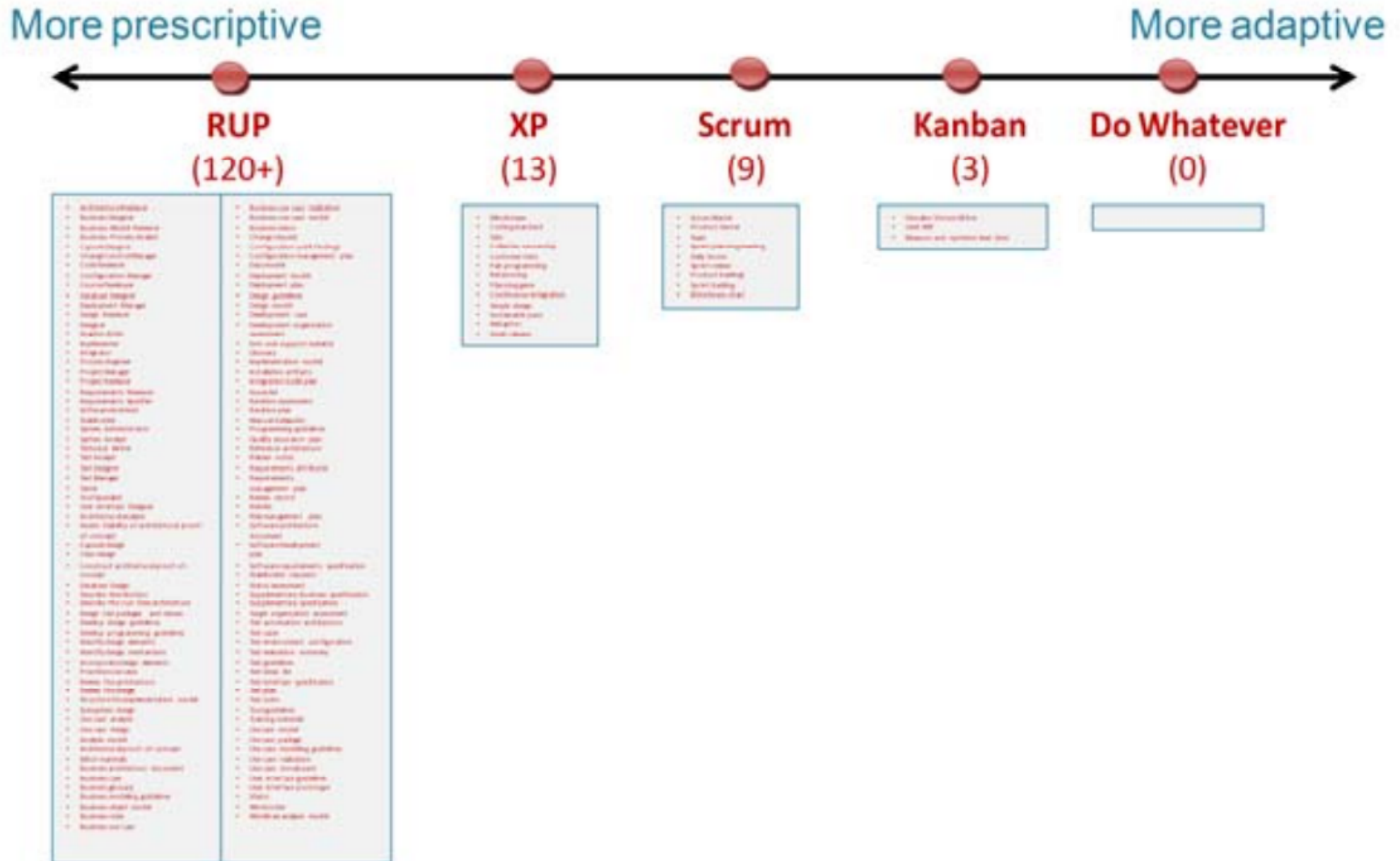
۹. توجه مداوم به برتری فنی و طراحی خوب باعث افزایش چابکی می شود.

۱۰. سادگی -- هنر به حداکثر رساندن مقدار کار انجام نشده -- ضروری است.

۱۱. بهترین معماری ها ، نیازمندی ها و طراحی ها از تیم های خود سازمانده پدید آور می شود.

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

مقایسه متدولوژی‌های چابک و سنتی (۱)



مقایسه روش‌های چابک و سنتی (۲)

• در روش‌های سنتی:

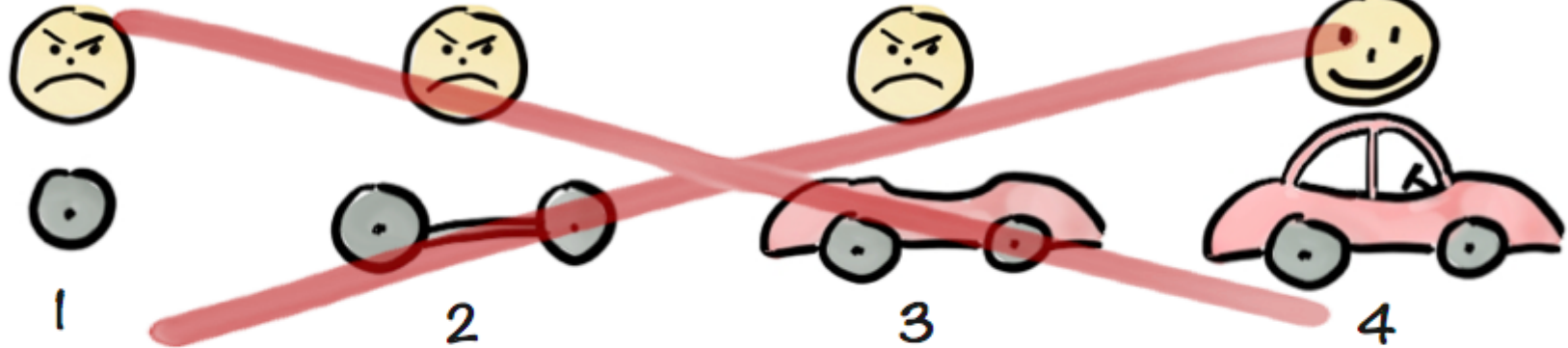
- مشتری تا آخر پروژه نرم‌افزار قابل استفاده نمی‌بیند
- تحلیل و طراحی در ابتدا و یکپارچه‌سازی و آزمون در آخر پروژه انجام می‌شود
- هر وظیفه را یک فرد یا تیم جداگانه که تخصص متفاوتی دارد انجام می‌دهد
- اطلاعات به صورت کتبی منتقل می‌شود
- وظایف هر فرد را مدیر پروژه اختصاص داده و پیگیری می‌کند
- سلسله‌مراتب و مناسبات رسمی بر روابط و تصمیم‌گیری‌ها حاکم است

• در روش‌های چابک:

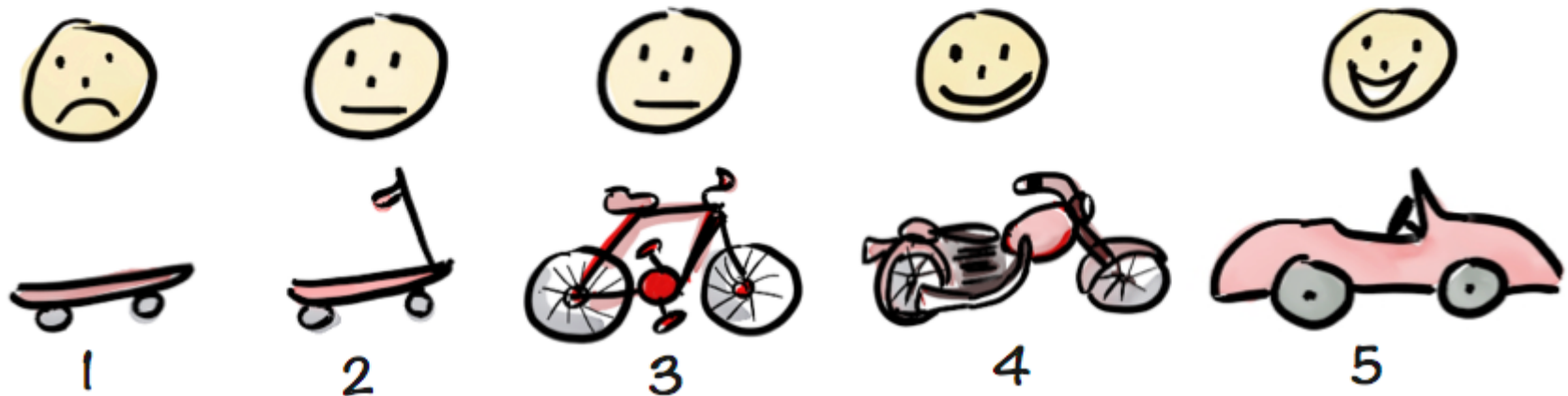
- مشتری در بازه‌های زمانی کوتاه نرم‌افزار را مرتباً تحویل می‌گیرد و بازخورد می‌دهد
- تحلیل، طراحی، پیاده‌سازی، یکپارچه‌سازی و آزمون، دائمی و در طول پروژه انجام می‌شود
- تمام فعالیت‌های مربوط به هر نیازمندی مشتری به طور تیمی و با همکاری تخصص‌های مختلف انجام می‌شود
- ترجیح بر مکالمه رودررو است
- تیم‌ها خودسازمان‌ده هستند
- تصمیمات به طور جمعی و مشارکتی گرفته می‌شوند

مفهوم توسعه تکراری - نمو

Not like this....

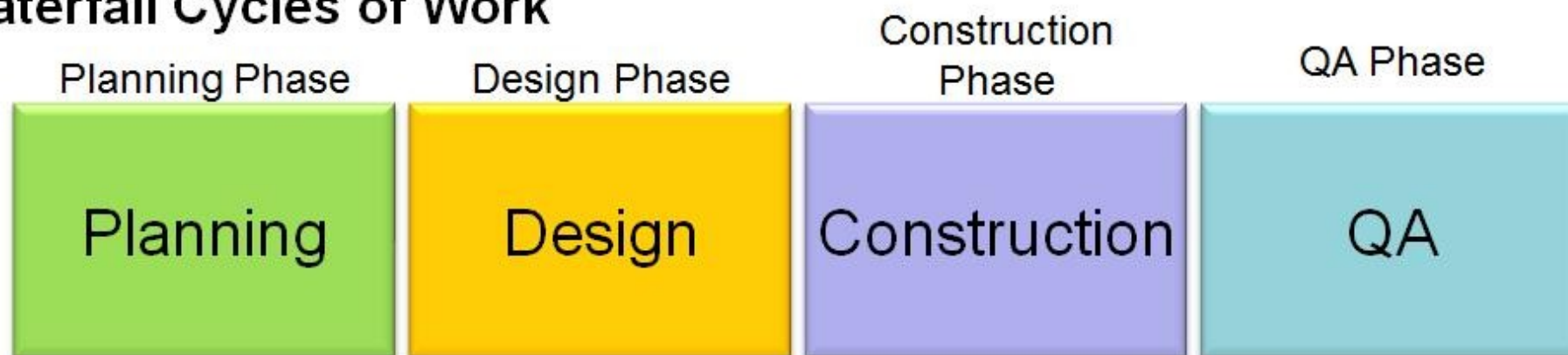


Like this!

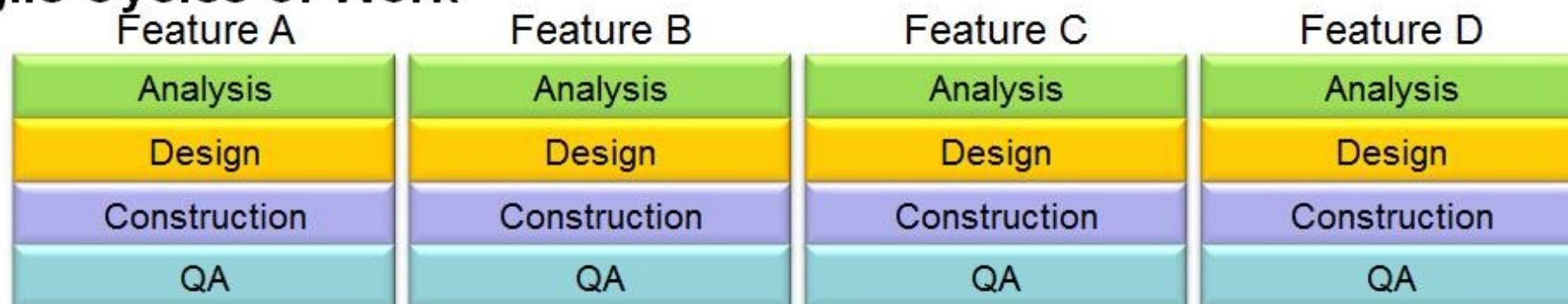


تفاوت در نحوه شکست کار چابک با روش سنتی

Waterfall Cycles of Work



Agile Cycles of Work



Time

مثال: چند نوع شکست کار

• محصول: سیستم آنلاین ثبت نام در کنفرانس

- روش ۱: طراحی و پیاده‌سازی پایگاه داده، منطق برنامه، واسط کاربری
- روش ۲: تحلیل نیازمندی‌ها، طراحی، پیاده‌سازی، آزمون
- روش ۳: ثبت نام و لاگین کاربر، وارد کردن مشخصات کاربر، پرداخت آنلاین هزینه، چاپ بلیت کنفرانس
- روش ۴: پرداخت آنلاین فقط با یک ایمیل، ایمیل کردن گد شرکت‌کننده، ثبت نام و لاگین کاربر (و تغییر فرایند پرداخت)، وارد کردن نام کاربر و چاپ بلیت کنفرانس

مثال: چند نوع شکست کار

- محصول: سیستم آنلاین ثبت نام در کنفرانس

- روش ۱: طراحی و پیاده‌سازی پایگاه داده، منطق برنامه، واسط کاربری

- روش ۲: تحلیل نیازمندی‌ها، طراحی، پیاده‌سازی، آزمون

- روش ۳: ثبت نام و لاگین کاربر، وارد کردن مشخصات کاربر، پرداخت آنلاین هزینه،

- چاپ بلیت کنفرانس

- روش ۴: پرداخت آنلاین فقط با یک ایمیل، ایمیل کردن گد شرکت‌کننده، ثبت نام و

- لاگین کاربر (و تغییر فرایند پرداخت)، وارد کردن نام کاربر و چاپ بلیت کنفرانس

- تفاوت User Story چابک و Task سنتی!

- بکلاگ محصول از User Story ها تشکیل می‌شود نه از Task های سنتی

تمرین!

- نیازمندی‌های آخرین پروژه‌ای را که انجام دادید به صورت چابک بشکنید!

سوء تفاهم‌های رایج دربارهٔ چابک

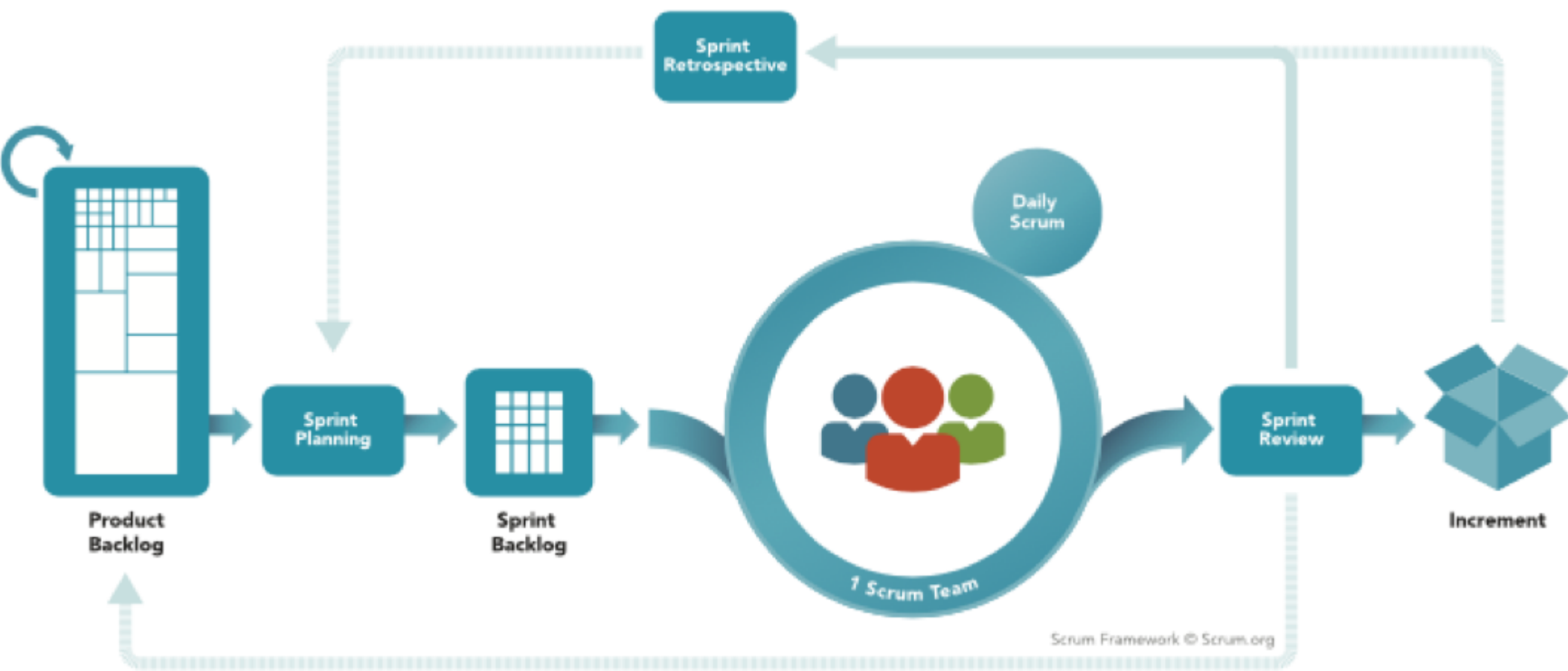
- روش‌های چابک یعنی این که فقط کُد بزنی! روش‌های چابک تحلیل و طراحی ندارند
- روش‌های چابک با مستندسازی مخالفند
- روش‌های چابک یعنی عدم انضباط و به‌هم‌ریختگی
- روش‌های چابک به درد پروژه‌های بزرگ و حساس نمی‌خورند

متدولوژی‌های چابک مرسوم

- اسکرام (Scrum)
- تکنیک‌های اکس‌پی (XP: eXtreme Programming)
- کانبان (Kanban)

- چارچوب مدیریت پروژه چابک
 - به مسائل فنی خاص حوزه نرم افزار نظیر آزمون و یکپارچه سازی نمی پردازد
- مشهورترین و رایج ترین فرایند مورد استفاده در توسعه نرم افزار در حال حاضر
- فهم آن ظاهراً ساده است ولی پیاده سازی واقعی آن در عمل بسیار دشوار
- هدف: تحویل مکرر نرم افزار ارزشمند به مشتری

چرخه اسکرام



نقش‌های اسکرام

- مالک محصول (Product Owner)

- مسئول نهایی موفقیت محصول
- مسئول انتقال اولویتهای مشتری به تیم از طریق بکلاگ محصول و جهت‌دهی به محصول در راستای نیازهای مشتری

- اسکرام‌مستر (ScrumMaster)

- مسئول نهایی موفقیت فرایند
- مربی تیم و مسئول اطمینان از اجرای صحیح و مفید اسکرام

- تیم توسعه (Development Team)

- تیم ۳ تا ۹ نفره، خودسازمان‌ده، متشکل از تمام تخصص‌های مورد نیاز (cross-functional)

تخمین چابک

- در اسکرام، روش رایج و توصیه شده برای تخمین، روش «تیمی» و «نسبی» است
 - تخمین توسط تیم توسعه زده می شود نه افراد خارج از تیم!
 - تمام اعضای تیم در تخمین مشارکت می کنند
 - User Story ها نسبت به یکدیگر تخمین زده می شوند
 - واحد تخمین دلخواه است (Story Point)
 - پس از پایان اسپرینت، مشخص می شود که تیم چقدر ظرفیت (Velocity) دارد
 - برای اسپرینت بعدی، تیم بر اساس ظرفیت اسپرینت های قبلی خود کار برمی دارد
- هدف از تخمین فقط کمک به پیش بینی آینده است

کار تیمی را دست کم نگیرید!

- کار تیمی را دست کم نگیرید!
 - قطعاً یکی از سه عامل مهم موفقیت است
 - «تولید نرم افزار یک ورزش تیمی است.»
- کار تیمی قابل یادگیری است
- در یک تیم واقعی
 - «به من چه» و «به تو چه» نداریم
 - «این کار رو انجام بدیم چون من می گم / چون من دوست دارم» نداریم

کار تیمی در اسکرام

- تیم به کمک اسکرام مستر، خود سازمان دهی را یاد می گیرد

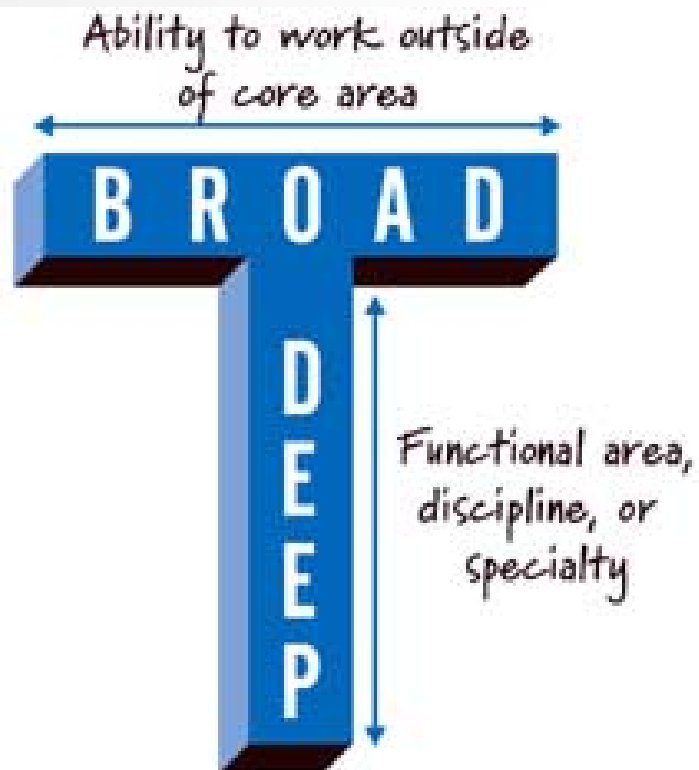
– اعتماد، بحث سازنده، تعهد به تصمیمات تیم، به چالش کشیدن یکدیگر و تمرکز روی هدف تیم

– مهارت های ارتباطی و نرم نقش کلیدی دارند

- تیم های چابک واقعی، متخصصین

عمومیت گرا

(Generalizing Specialist) دارند



تکنیک‌های فنی چابک

- بدون برتری فنی (Technical Excellence) چابکی فقط یک ادعای توخالی است
- نرم‌افزار باید از ابتدا باکیفیت تولید شود نه اینکه در آخر به فکر کیفیت بیفتیم
- تکنیک‌های فنی چابک
 - یکپارچه‌سازی مداوم (Continuous Integration)
 - ریفکتورینگ (Refactoring)
 - طراحی تدریجی (Continuous Design)
 - آزمون پذیرش خودکار (Automated Acceptance Testing)
 - توسعه مبتنی بر آزمون (TDD: Test-Driven Development)
 - برنامه‌نویسی دونفره (Pair Programming) و مرور کد (Code Review)
 - مالکیت جمعی کد (Collective Code Ownership)