

# آنچه باید دربارهٔ فرآیند توسعهٔ نرم‌افزار بدانید!

محمدحسین احمدی

smhahmadi@chmail.ir

# معرفی اینجانب

- محمدحسین احمدی
  - فارغ‌التحصیل کارشناسی مهندسی IT و ارشد نرم‌افزار شریف
  - سابقه برنامه‌نویسی C++, Bash و PHP در کار
  - چهار سال مسئول اصلاح فرایند توسعه نرم‌افزار (و مدیر گروه بهره‌وری تولید) در شرکت امن‌افزار‌گستر شریف
  - ۵ سال مدیر گروه و مدیر محصول فایروال در شرکت امن‌افزار‌گستر شریف
  - مشاوره بهبود فرایند توسعه نرم‌افزار به شرکت‌های مختلف

# فهرست

- «فرایند توسعه نرم افزار» یعنی چه؟
- چرا مهم است؟
- آشنایی مختصر با متدولوژی های مشهور
- آشنایی با توسعه چابک نرم افزار
- آشنایی با پرکتیس های فنی چابک و DevOps

# منظور از «فایند توسعه نرم افزار» چیست؟

- هر کاری که انجام می‌دهیم تا یک نرم افزار تولید شود، به مشتری تحويل داده شود و پس از تحويل نگهداری شود.
- از کارهای «بزرگ» مثل برنامه‌ریزی، جمع‌آوری نیازمندی‌ها از مشتری، طراحی، برنامه‌نویسی، آزمون و تحويل گرفته تا موارد ظاهراً جزئی مثل نحوه نوشتن گُد توسط برنامه‌نویسان، نحوه مرور گُد توسط سایر برنامه‌نویسان، محل نشستن اعضای تیم، چگونگی نوشتن آزمون‌های خودکار و ... ذیل موضوع «فایند توسعه نرم افزار» می‌گنجند.
- اگر فایند توسعه نرم افزار مدون باشد، «متدولوزی» نامیده می‌شود.

# مثال‌هایی از فرایندهای توسعهٔ 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 چاپک: 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 گُدد 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 گد استفاده می‌کنند.
- استفاده از UML برای مدل‌سازی نیازمندی‌ها و طراحی در بسیاری از پروژه‌های نرم‌افزاری رایج است.
- بسیاری از مدیران پروژه کم تجربه 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 «تحویل پیوسته» (Continuous Delivery) تمام آزمون‌ها به طور خودکار انجام می‌شوند و گُد برنامه‌نویسان در چند دقیقه در صورت پاس کردن تمام تست‌ها، به طور خودکار در سیستم اصلی نصب می‌شود و زیر بار می‌رود.
- 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 توسعهٔ نرم‌افزار؟

- بعد از مسائل تجاری و مهارت و تجربهٔ افراد درگیر، موفقیت یک محصول یا پروژهٔ نرم‌افزاری را فرایند توسعهٔ آن نرم‌افزار مشخص می‌کند.
- یک فرایند توسعهٔ نرم‌افزار خوب می‌تواند بازدهی یک تیم را از نظر سرعت و کیفیت نرم‌افزار خروجی چندین برابر افزایش دهد.
- فرایند توسعهٔ نرم‌افزار روی رضایت مشتری و ذی‌نفعان، کیفیت خروجی، رضایت و انگیزهٔ توسعه‌دهندگان، سرعت کار و قابل پیش‌بینی شدن کار توسعهٔ نرم‌افزار تأثیر می‌گذارد.

## 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 بزرگ‌تر و پیچیده‌تر می شود
- (در صورتی که هیچ یک از موارد فوق صدق نمی کند) وقتی تیم مهارت و تجربه کمتری دارد

# پروژهای نرم افزاری معمولاً شکست می خورند

- آمار گزارش Standish Group از CHAOS نشان می دهد
  - فقط ۱۶٪ از پروژهای نرم افزاری موفق هستند
  - ۵۳٪ با مشکلات (در محدوده، کیفیت، زمان یا هزینه) به پایان می رسد
  - ۳۱٪ کلاً قبل از اتمام لغو می شوند
  - هر پروژه به طور متوسط دو برابر زمان تخمینی اولیه طول می کشد
- آمار موفقیت پروژهای بزرگ آئی تی در یک دهه: ۴/۶٪  
(ComputerWorld, ۲۰۱۳)

# 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 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 اقلًا 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

# مثال: تأثیر فرایند توسعه نرم افزار روی کیفیت

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

# بازی!

- اثر نرخ بازخورد روی کیفیت کار

# تاریخچه‌ای مختصر از روش‌های توسعهٔ نرم‌افزار

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

# تاریخچه‌ای مختصر از روش‌های توسعهٔ نرم‌افزار

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

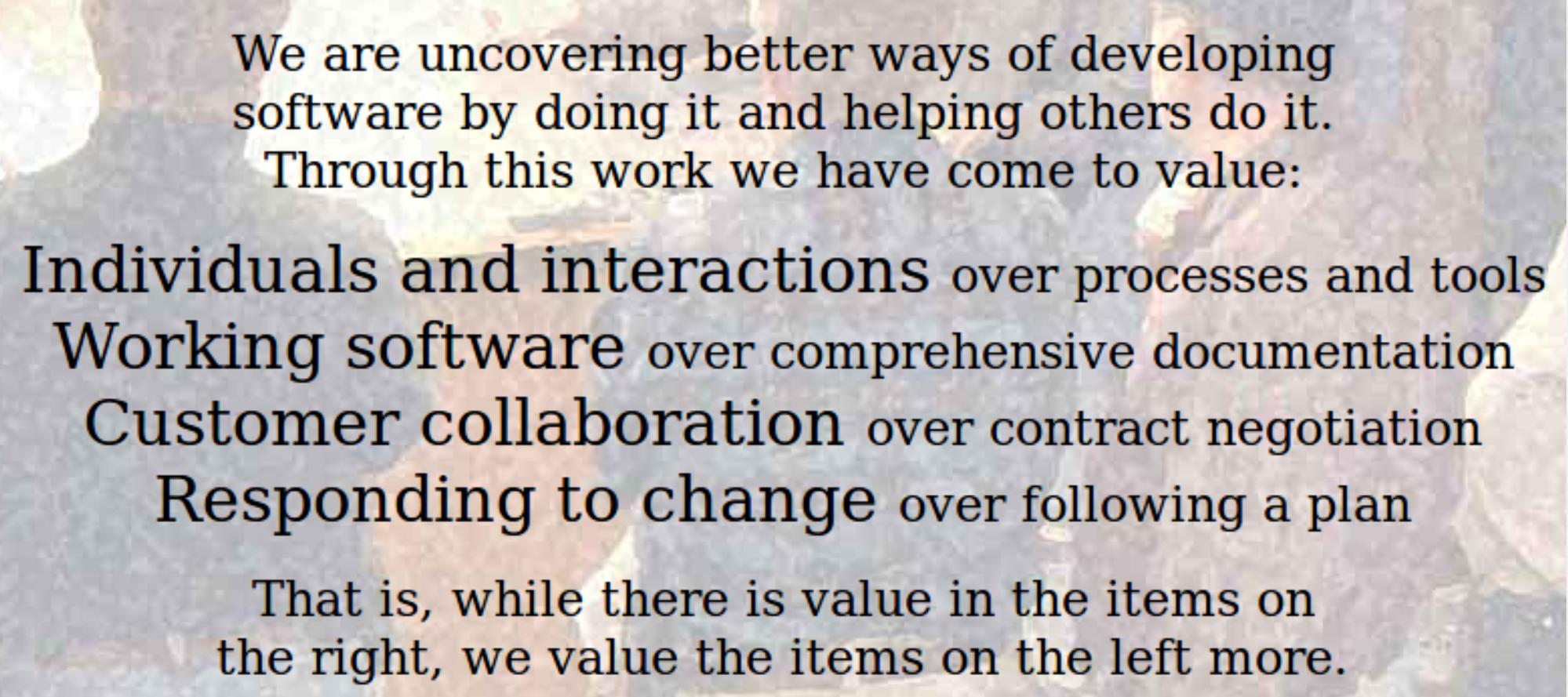
# بازی!

- شبیه‌سازی ارتباط مشتری با تیم توسعه

# فرایندهای چابک توسعه نرم افزار

- فرایندهای چابک توسعه نرم افزار تا حد زیادی در پاسخ به متداولوژی‌های سنگین‌وزن نظیر RUP پدید آمدند
- تأکید بر بازخورد سریع و انعطاف در مقابل تغییرات، توجه به نیاز واقعی مشتری، کار تیمی و مشارکتی، «توسعه‌دهنده‌پسند» یا «عضو تیم‌پسند» کردن فرایند و کم کردن تشریفات
- تغییر فضای توسعه نرم افزار از «شبیه سایر مهندسی‌ها» به «مهارت-هنر-تجربه حرفه‌ای»

# مانیفست توسعہ چاپک



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

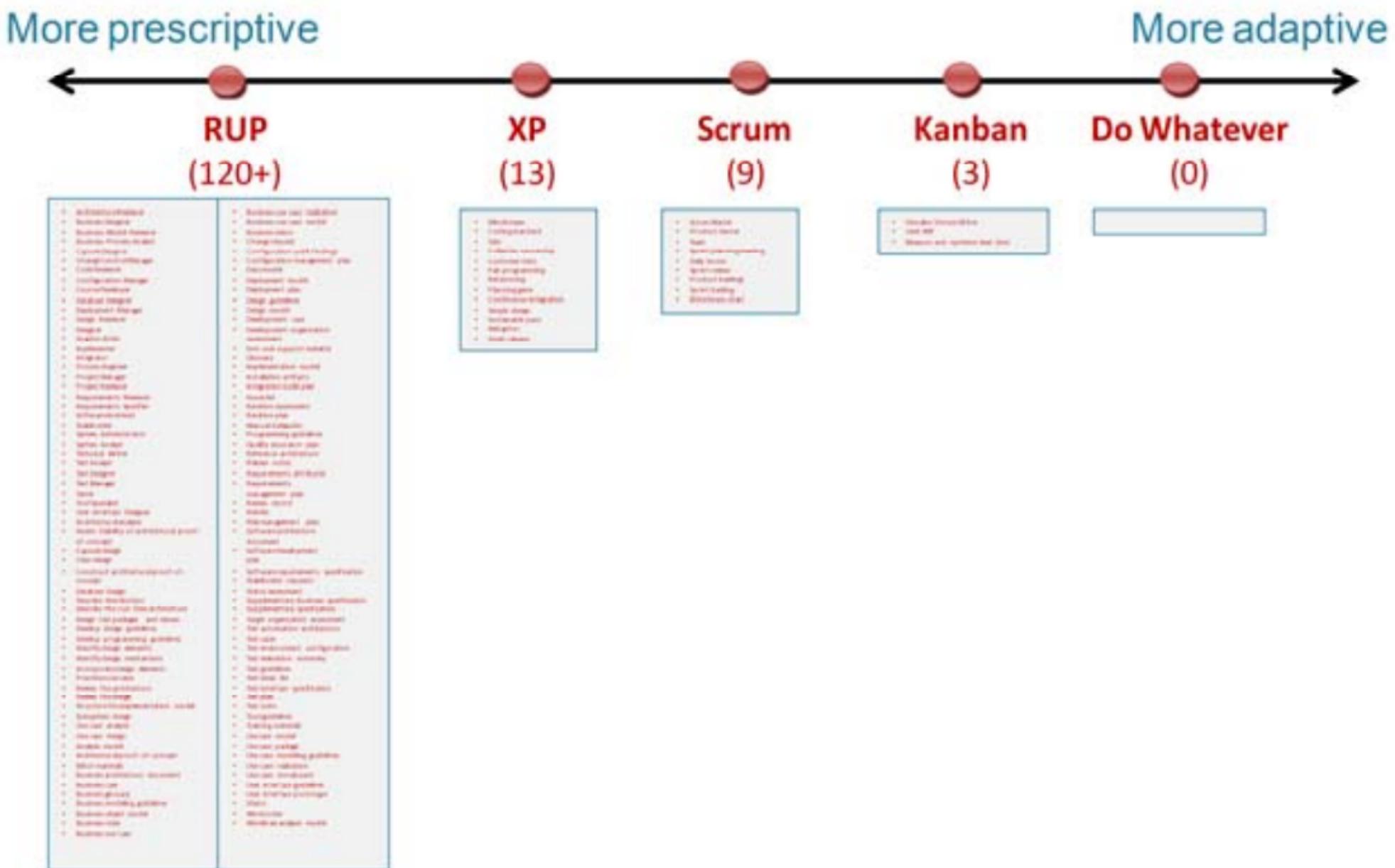
**Responding to change** over following a plan

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 میکند.
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 باشد 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 دهنده ها 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 حداقل رساندن مقدار کار انجام نشده -- ضروری است.
۱۱. بهترین معماری ها ، نیاز مندی ها و طراحی ها از تیم های خود سازمانده پدید آور می شود.
۱۲. در فواصل منظم ، تیم برچگونگی موثرتر شدن تأمل و تفکر می نماید و سپس تیم رفتار خود را بر اساس بازتاب این تفکر تنظیم و هم سو می نماید.

# مقایسه متداولوژی‌های چاپک و سنتی (۱)



# مقایسهٔ روش‌های چابک و سنتی (۲)

- در روش‌های چابک:

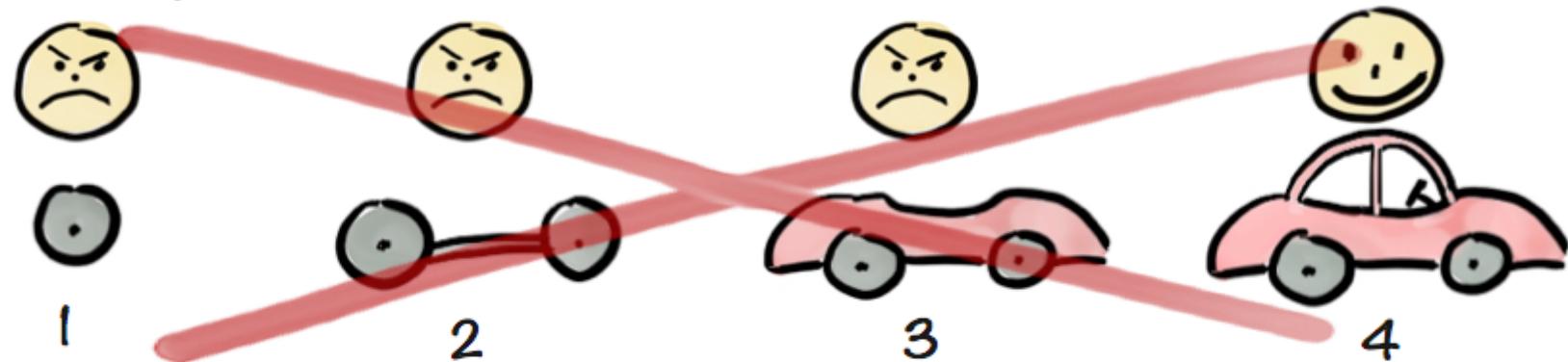
- مشتری در بازه‌های زمانی کوتاه نرمافزار را مرتبأً تحويل می‌گیرد و بازخورد می‌دهد
- تحلیل، طراحی، پیاده‌سازی، یکپارچه‌سازی و آزمون، دائمی و در طول پروژه انجام می‌شود
- تمام فعالیت‌های مربوط به هر نیازمندی مشتری به طور تیمی و با همکاری تخصص‌های مختلف انجام می‌شود (**Cross-functional Team**)
- ترجیح بر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

- در روش‌های سنتی:

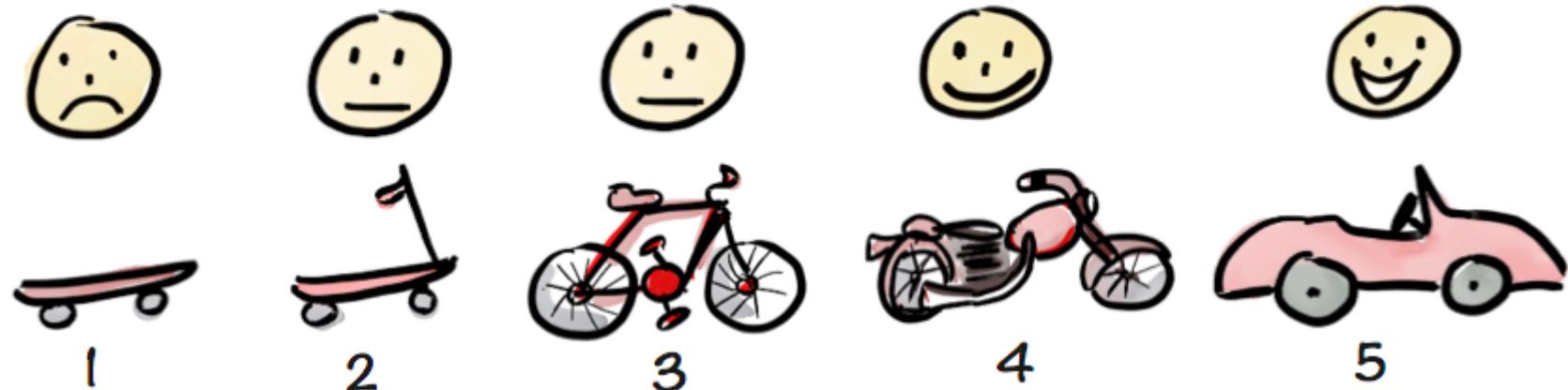
- مشتری تا آخر پروژه نرمافزار قابل استفاده نمی‌بیند
- تحلیل و طراحی در ابتدا و یکپارچه‌سازی و آزمون در آخر پروژه انجام می‌شود
- هر وظیفه را یک فرد یا تیم جداگانه که تخصص متفاوتی دارد انجام می‌دهد
- اطلاعات به صورت کتبی منتقل می‌شود
- وظایف هر فرد را مدیر پروژه اختصاص داده و پیگیری می‌کند
- سلسله‌مراتب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

# مفهوم توسعهٔ تکراری-نمای

Not like this....

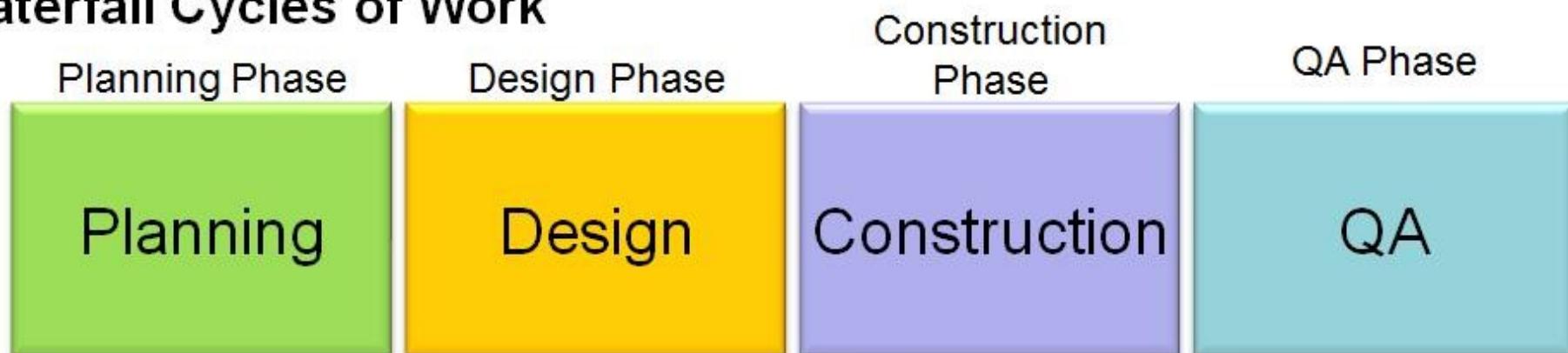


Like this!

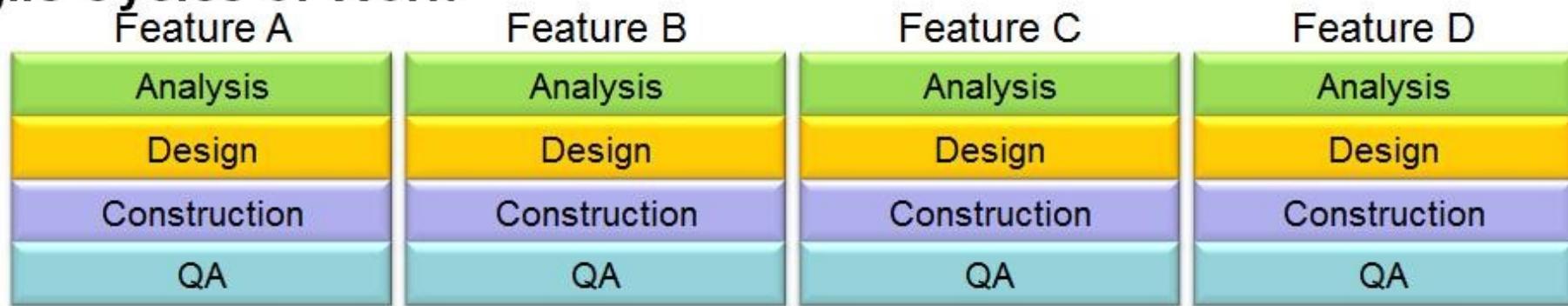


# تفاوت در نحوه شکست کار چابک با روش سنتی

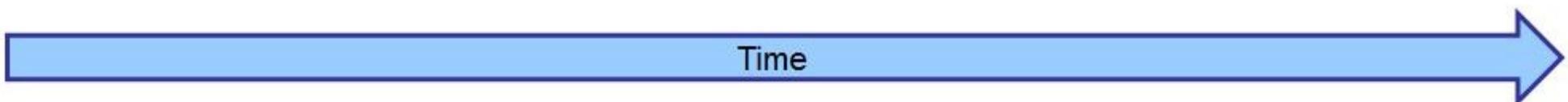
## Waterfall Cycles of Work



## 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 لاغین کاربر، وارد کردن مشخصات کاربر، پرداخت آنلاین هزینه، چاپ بلیت کنفرانس
  - روش ۴: پرداخت آنلاین فقط با یک ایمیل، ایمیل کردن گذشته کننده، ثبت نام و لاغین 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 لاغین کاربر، وارد کردن مشخصات کاربر، پرداخت آنلاین هزینه، چاپ بلیت کنفرانس
  - روش ۴: پرداخت آنلاین فقط با یک ایمیل، ایمیل کردن گذشته کننده، ثبت نام و لاغین کاربر (و تغییر فرایند پرداخت)، وارد کردن نام کاربر و چاپ بلیت کنفرانس
- تفاوت User Story چاپک و Task سنتی!
- بکلاگ محصل از User Story‌ها تشکیل می‌شود نه از Task‌های سنتی

# سوءتفاهم‌های رایج در بارهٔ چابک

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

# رویدهای چاک مرسوم

- اسکرام (Scrum)
- پرکتیس‌های فنی اکسپی (XP: eXtreme Programming)
- پرکتیس‌های Continuous Delivery و DevOps
- کانبان (Kanban)

# تکنیک‌های فنی چابک

- بدون برتری فنی (Technical Excellence) چابکی فقط یک ادعای توخالی است
- نرمافزار باید از ابتدا با کیفیت 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)
  - توسعهٔ مبتنی بر آزمون (TDD: Test-Driven Development)
  - برنامه‌نویسی دونفره (Pair Programming) و مرور گُد (Code Review)
  - مالکیت جمعی گُد (Collective Code Ownership)

# جنبش CI/CD و DevOps

DevOps = Development + Operations •

- چابکی بیشتر با Cross-functional بودن بیشتر
- بازخورد سریع تر

CI/CD: Continuous Integration / Continuous •  
Delivery

- امکان تحویل مکرر خروجی به مشتری (از هفتاهای چند بار تا صدها بار در روز)
- حذف و خودکارسازی «فاز»های یکپارچه‌سازی، تست و تحویل

# تحویل پیوسته نیاز به ابزار دارد

- ابزارهای ایجاد زیرساخت و استقرار نرمافزار

(Infrastructure as Code)

- کانتینر (مثل Docker)

- مدیریت پیکربندی ماشین‌ها (مثل Ansible)

- ایجاد و مدیریت خودکار زیرساخت (مثل Terraform و Kubernetes)

• ابزارهای مانیتورینگ زیرساخت (مثل Zabbix و ELK)

- ابزارهای CI

- کنترل نسخه (مثل Git)

- بیلد خودکار (مثل Jenkins و Gitlab)

• آزمون واحد خودکار (مثل PyUnit و / unittest)

• مرور کد خودکار (مثل Sonar و Mypy، Pylint، Pyflakes)

• آزمون پذیرش خودکار (مثل Selenium و Cucumber)

# سایر مباحث

- جهت مطالعه خارج از کلاس

# اسکرام

- چارچوب مدیریت پروژه چابک
  - به مسائل فنی خاص حوزه نرمافزار نظیر آزمون و یکپارچه‌سازی نمی‌پردازد
- مشهورترین و رایج‌ترین فرایند مورد استفاده در توسعه نرمافزار در حال حاضر
- فهم آن ظاهراً ساده است ولی پیاده‌سازی واقعی آن در عمل بسیار دشوار
- هدف: تحويل مكرر نرمافزار ارزشمند به مشتری

# چرخه اسکرام



# تیم اسکرام

- تیم متمرکز بر **هدف محصول**، متشکل از تمام تخصص‌های مورد نیاز (cross-functional)، خوداداره‌کننده، شامل:
  - توسعه‌دهندگان (Developers)
  - حداقل ۸ نفر، شامل غیربرنامه‌نویسان هم می‌شود
- مالک محصول (Product Owner)
  - مسئول نهایی موققیت محصول
  - مسئول جهت‌دهی به محصول در راستای نیازهای مشتری از طریق **هدف محصول و بکلاگ محصول**
- اسکرام‌مستر (ScrumMaster)
  - مسئول نهایی کارایی تیم اسکرام
  - مربی تیم و مسئول اطمینان از اجرای صحیح و مفید اسکرام

# تخمین چاپک

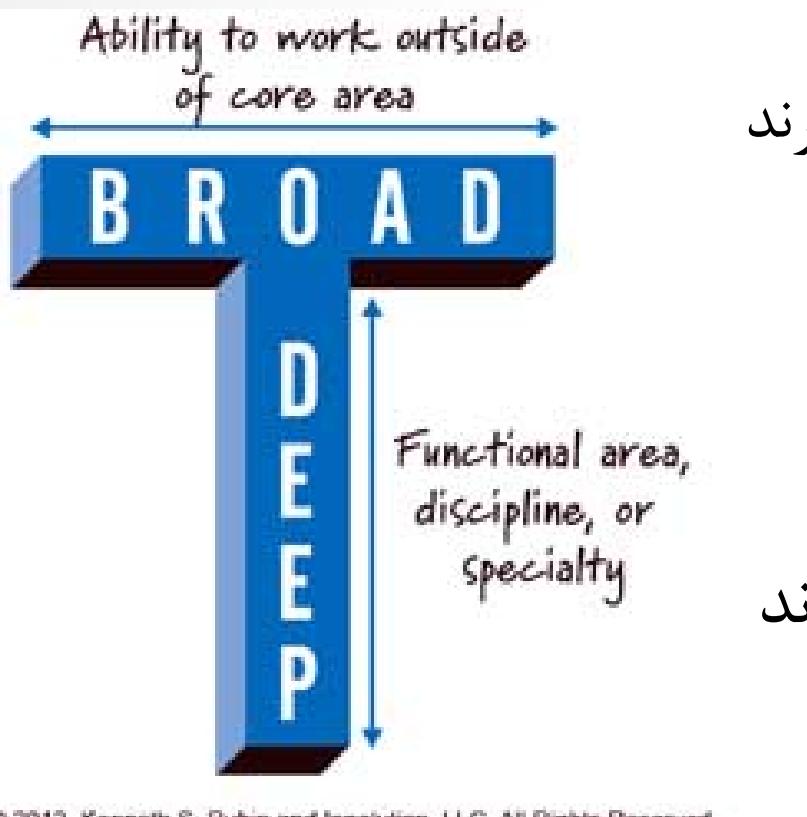
- در اسکرام، روش رایج و توصیه شده برای تخمین، روش «تیمی» و «نسبی» است
  - تخمین توسط تیم توسعه زده می‌شود نه افراد خارج از تیم!
  - تمام اعضای تیم در تخمین مشارکت می‌کنند
  - User Story‌ها نسبت به یکدیگر تخمین زده می‌شوند
  - واحد تخمین دلخواه است (Story Point)
  - پس از پایان اسپرینت، مشخص می‌شود که تیم چقدر ظرفیت (Velocity) دارد
  - برای اسپرینت بعدی، تیم بر اساس ظرفیت اسپرینت‌های قبلی خود کار برمی‌دارد
- هدف از تخمین  فقط کمک به پیش‌بینی آینده است

# کار تیمی را دست کم نگیرید!

- کار تیمی را دست کم نگیرید!
  - قطعاً یکی از سه عامل مهم موفقیت است
  - «تولید نرمافزار 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 می‌گم / چون من دوست دارم» نداریم

# کار تیمی در اسکرام

- تیم به کمک اسکرام مستر، خودسازماندهی را یاد می‌گیرد
  - اعتماد، بحث سازنده، تعهد به تصمیمات تیم، به چالش کشیدن یکدیگر و تمرکز روی هدف تیم



- تیم‌های چابک واقعی، متخصصین عمومیت‌گرا دارند (Generalizing Specialist)