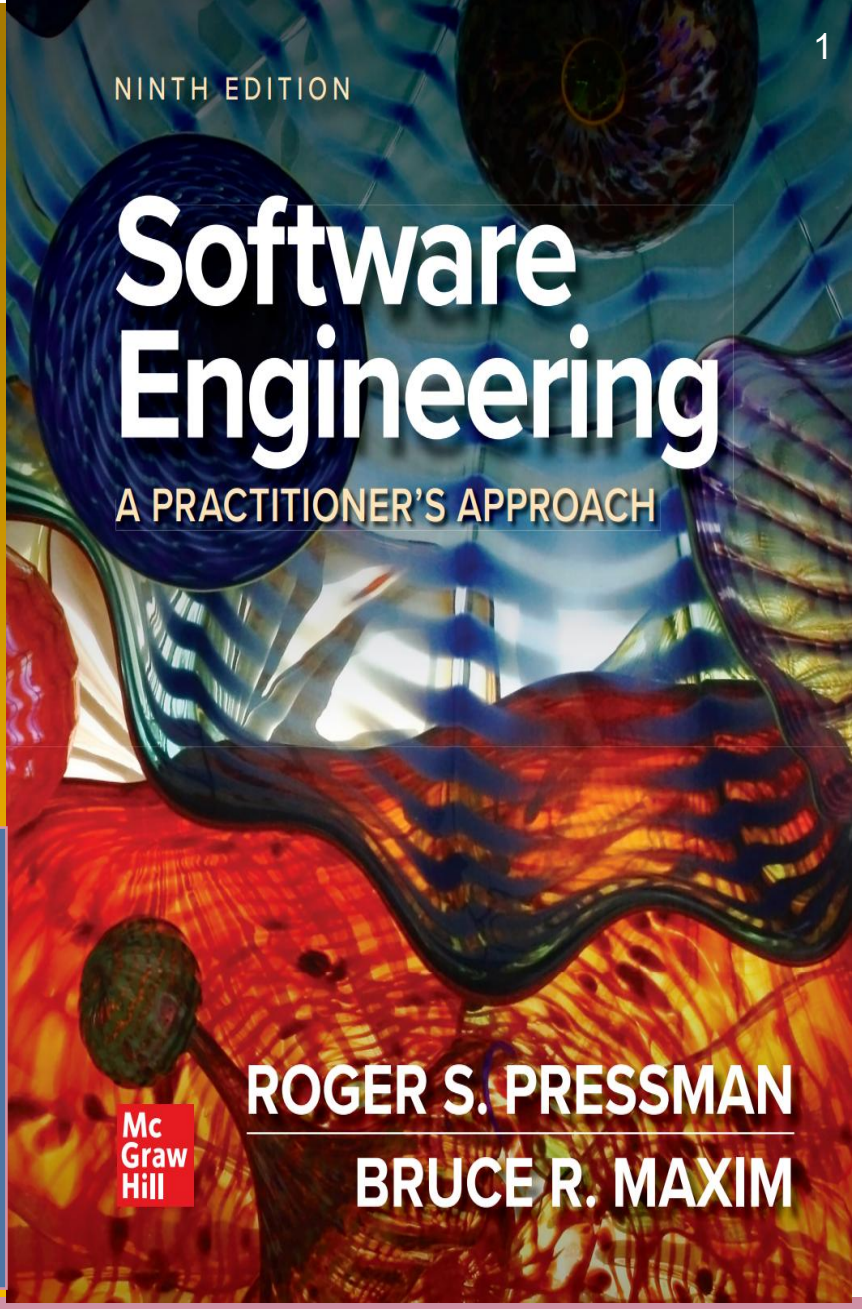
توسعه نرم افزار چابک



Website: ataghinezhad.github.io, Email:a0taghinezhad@gmail.com

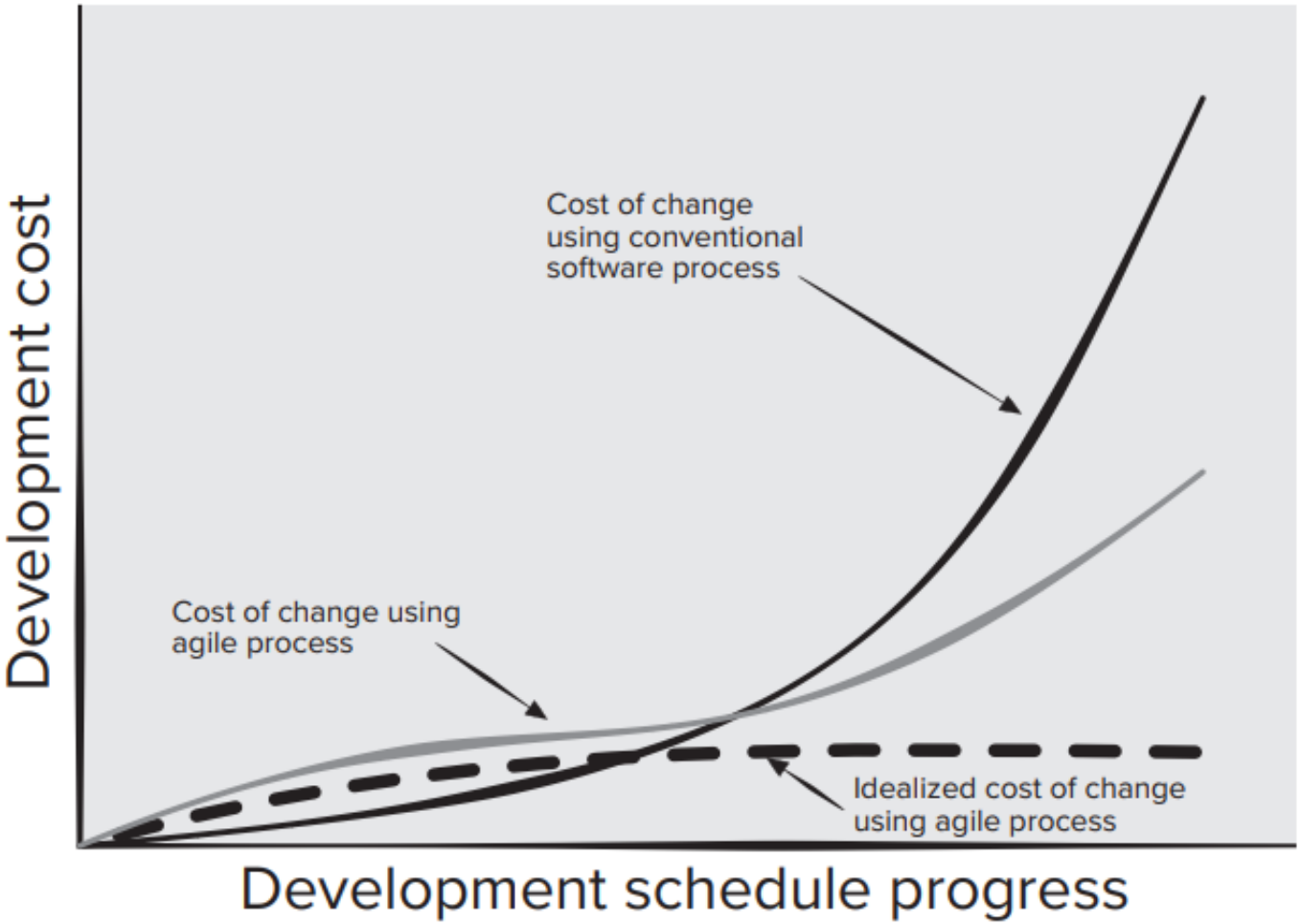
موضوعات

- متدهای چابک
- توسعه چابک و مبتنی بر برنامه
- Extreme programming برنامه‌سازی مفرط
- مدیریت پروژه چابک
- مقیاس پذیری روش‌های چابک

Dr. A. Taghinezhad

هزینه تغییر

Change costs as a function of time in development



فرایند چابک

- هر فرآیند نرم افزار چابکی به گونه ای مشخص می شود که به چند **فرض** کلیدی در مورد اکثر پروژه های نرم افزار می پردازد:
- ۱. پیش بینی اینکه کدام **الزامات نرم افزار** باقی می مانند و کدام **تغییر خواهند کرد**، از پیش دشوار است. **پیش بینی اینکه اولویت های مشتری** با **پیشرفت پروژه چگونه تغییر خواهد** کرد نیز به همان اندازه دشوار است.
- ۲. برای بسیاری از انواع نرم افزار، طراحی و ساخت به هم مرتبط هستند. پیش بینی اینکه **قبل از ساخت برای اثبات طراحی، چقدر طراحی لازم است**، دشوار است.
- ۳. **تحلیل، طراحی، ساخت** و تست از نظر برنامه ریزی به اندازه ای که ممکن است دوست داشته باشیم **قابل پیش بینی نیستند**.

Dr. A. Taghinezhad

فرایند چابک

- با توجه به این مفروضات، یک سوال مهم مطرح می شود:
- چگونه فرآیندی ایجاد کنیم که بتواند عدم قطعیت را مدیریت کند؟
- پاسخ در انعطاف پذیری فرآیند (برای سازگاری سریع با تغییرات پروژه و شرایط فنی) نهفته است.
  - بنابراین، یک فرآیند چابک باید انعطاف پذیر باشد.

Dr. A. Taghinezhad

بیانیه چابک

- به ارزش های زیر رسیده ایم:
  - ارزش مندی افراد به نسبت تعاملات بر فرآیندها و ابزارها.
  - ارزشمندی نرم افزار در حال کار بر روی مستندات جامع.
  - ارزشمندی همکاری با مشتری بر مذاکره قرارداد.
  - ارزشمندی پاسخگویی به تغییر به جای پیروی از یک برنامه.

Dr. A. Taghinezhad

اصول	توضیح
تعامل مشتری	مشتریان باید در طول فرآیند توسعه به طور نزدیک درگیر شوند. **نقش** آنها **ارائه** و **اولویت بندی الزامات** جدید سیستم و **ارزیابی تکرارهای** سیستم است.
تحویل افزایشی	نرم افزار به صورت **افزایشی** توسعه می یابد و **مشتری الزاماتی را که باید در هر افزایش گنجانده شود**، مشخص می کند.
افراد نه فرآیندها	باید **مهارت های تیم توسعه را شناسایی** و از آنها بهره برداری کرد. باید به اعضای تیم اجازه داد تا **بدون فرآیندهای دستوری، روش های کاری** خود را توسعه دهند.
درآغوش گرفتن تغییر	انتظار داشته باشید که **الزامات سیستم تغییر کند**، بنابراین سیستمی را طراحی کنید که بتواند این تغییرات را در خود جای دهد.
نگهداری سادگی	روی سادگی هم در نرم افزار در حال توسعه و هم در فرآیند توسعه تمرکز کنید. در هر کجا که ممکن است، **به طور فعال برای حذف پیچیدگی از سیستم کار کنید.**

کاربرد روش چابک

- **محصولات مناسب برای توسعه چابک:**
- **توسعه محصول**: جایی که یک شرکت نرم افزار محصولی با اندازه **کوچک یا متوسط** برای فروش تولید می کند.
- توسعه **سیستم سفارشی درون یک سازمان**:
  - جایی که تعهد روشنی از طرف مشتری برای مشارکت در فرآیند توسعه وجود دارد و قوانین و **مقررات خارجی زیادی بر نرم افزار تأثیر نمی گذارد**.

محدودیت های روش چابک:

به دلیل تمرکز بر **تیم های کوچک و کاملاً یکپارچه**، در مقیاس بندی روش های چابک برای **سیستم های بزرگ مشکلاتی** وجود دارد.

Dr. A. Taghinezhad

چالش های روش چابک:

- حفظ **علاقه مشتریانی** که در فرآیند مشارکت دارند، می تواند دشوار باشد. (**حوصله اولویت بندی و مشارکت در تکرارها**)
- اعضای تیم ممکن است برای مشارکت عمیقی که مشخصه روش های چابک است، مناسب نباشند. (**ارتباط افراد**)
- **اولویت بندی تغییرات** در جایی که **ذینفعان متعددی** وجود دارند، می تواند دشوار باشد.
- قراردادها مانند سایر **رویکردهای توسعه تکرارشونده** ممکن است مشکل ساز شوند.

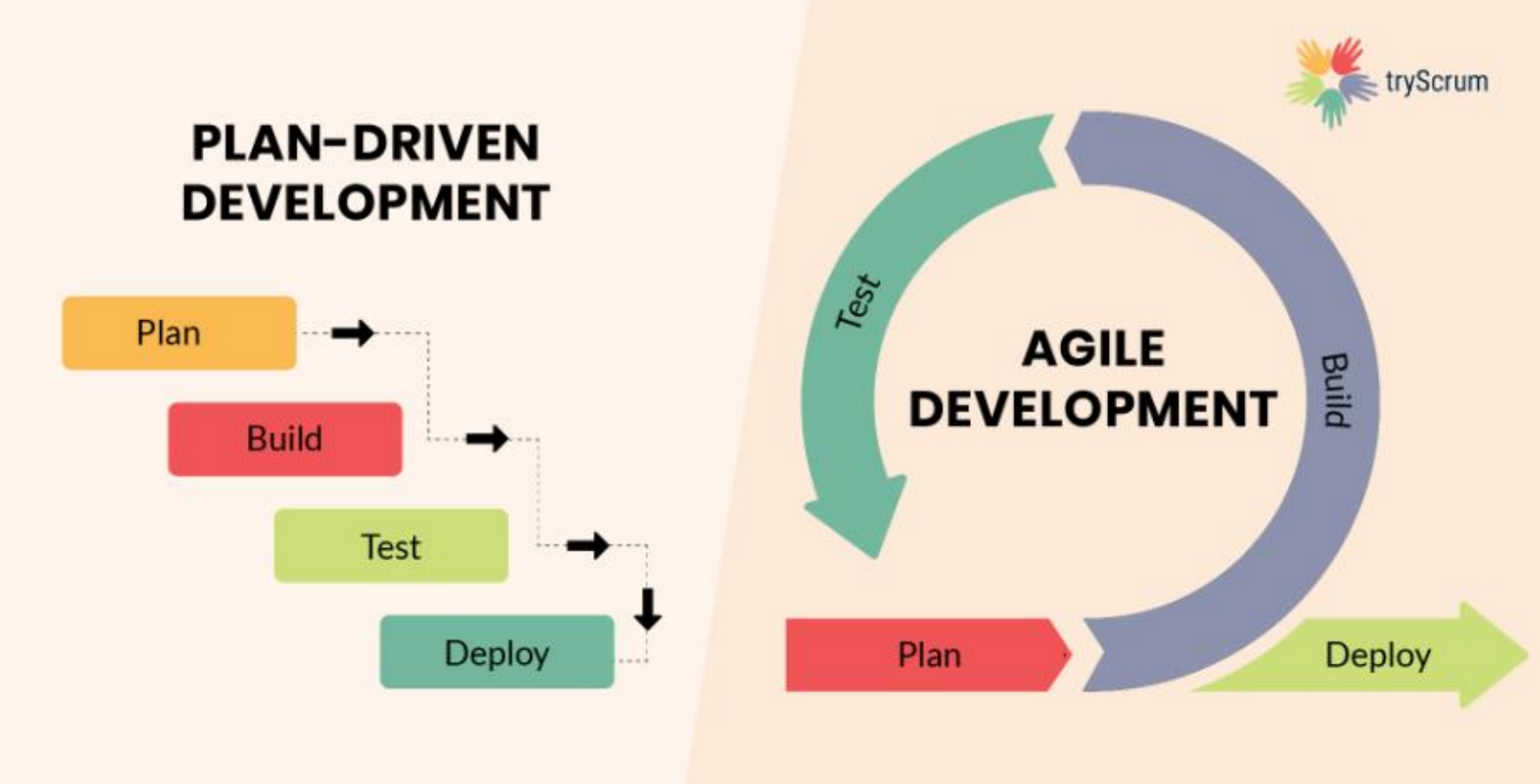
Dr. A. Taghinezhad

نگهداری نرم افزار و روش های چابک

- **مسائل کلیدی در نگهداری نرم افزار با روش چابک:**
  - با توجه به تأکید بر به حداقل رساندن مستندات رسمی در فرآیند توسعه، آیا سیستم هایی که با استفاده از رویکرد چابک توسعه یافته اند، **قابل نگهداری هستند**؟
  - آیا می توان از روش های چابک برای تکامل یک سیستم در پاسخ به درخواست های تغییر مشتری استفاده کرد؟
  - در صورتی که تیم توسعه اولیه قابل نگهداری نباشد، ممکن است مشکلاتی ایجاد شود.

Dr. A. Taghinezhad

توسعه مبتنی بر برنامه در مقابل توسعه چابک



توسعه مبتنی بر برنامه در مقابل توسعه چابک:

- توسعه مبتنی بر برنامه:
  - **رویکرد مبتنی بر برنامه** در مهندسی نرم افزار بر اساس **مراحل توسعه جداگانه با خروجی هایی** است که باید در هر **یک از این مراحل از پیش برنامه‌ریزی** شود.
  - لزوماً مدل آبشاری نیست - توسعه افزایشی مبتنی بر برنامه امکان پذیر است.
  - **تکرار در درون فعالیت ها رخ می دهد.**
  - **ویژگی‌ها**:۱)هر مرحله قبل از شروع مرحله بعد باید تکمیل شود،۲) مستندسازی گسترده،۳) مشارکت محدود مشتری،۴) فرآیند مستقل مدیریت تغییرات
- توسعه چابک:
  - مشخصات، **طراحی، پیاده سازی و تست در هم آمیخته می شوند** و **خروجی های فرآیند توسعه از طریق مذاکره** در طول فرآیند توسعه نرم افزار تصمیم گیری می شوند.

Dr. A. Taghinezhad

مثال‌های توسعه مبتنی بر برنامه‌ریزی

مثال مدل آبشاری:

- **پروژه** :توسعه یک سیستم حقوق و دستمزد برای یک سازمان بزرگ. **نتیجه** :محصول نهایی پس از تکمیل همه مراحل تحویل داده می‌شود و امکان تغییرات در طول مسیر محدود است.
- **فرآیند: مرحله ۱ جمع‌آوری نیازمندی‌ها- مرحله ۲: طراحی سیستم -مرحله ۳ پیاده‌سازی- مرحله ۴: آزمایش**

مثال توسعه افزایشی مبتنی بر برنامه‌ریزی:

- **پروژه** :توسعه **سیستم درخواست وام آنلاین** برای یک بانک.
- **فرآیند:** تیم توسعه پروژه را به **سه مرحله افزایشی** تقسیم می‌کند:
  1. **ماژول ثبت‌نام کاربر**
  2. **ارائه درخواست وام**
  3. **فرآیند تأیید وام**
- **نتیجه** :مرحله اول (ثبت‌نام کاربر) تکمیل و آزمایش می‌شود قبل از شروع مرحله بعد. هرچند توسعه به‌صورت افزایشی انجام می‌شود، اما هر مرحله به‌طور دقیق برنامه‌ریزی شده است.

Dr. A. Taghinezhad

مقایسه توسعه مبتنی بر برنامه‌ریزی و توسعه چابک

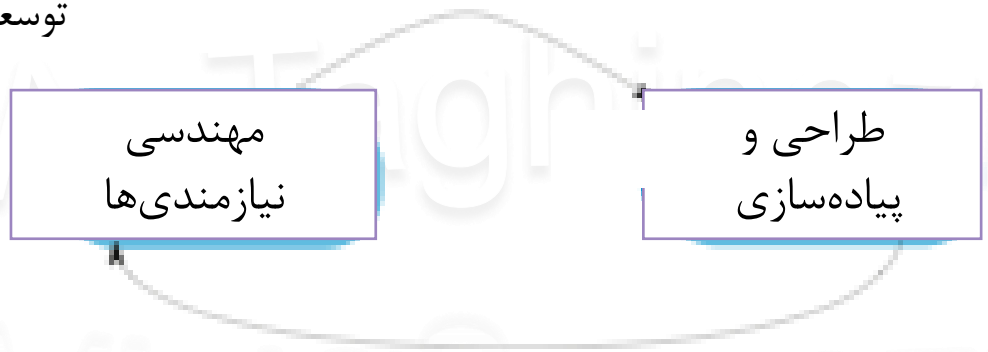
جنبه	توسعه چابک	توسعه مبتنی بر برنامه‌ریزی
جریان فرآیند	تکراری و افزایشی	متوالی یا مرحله‌ای (مثل آبشاری یا افزایشی)
مستندسازی	سبک و حداقلی	گسترده و دقیق در ابتدا
مشارکت مشتری	همکاری مداوم با مشتری	محدود به مرحله نیازمندی‌ها
انعطاف‌پذیری در تغییرات	بالا – تغییرات در طول فرآیند پذیرفته می‌شوند	پایین – تغییرات از طریق درخواست‌های رسمی مدیریت می‌شوند
روش تحویل	تحویل مداوم بخش‌های کوچک (نرم‌افزار قابل استفاده در هر اسپرینت)	تحویل کل محصول یا افزونه‌های بزرگ

Plan-driven and agile specification

توسعه مبتنی بر برنامه

درخواست تغییر نیازمندی‌ها

توسعه چابک



مسائل سازمانی انسانی و فنی

- **انتخاب رویکرد مناسب:** فرآیندهای مبتنی بر برنامه و چابک
- آیا **داشتن مشخصات و طراحی** بسیار دقیق قبل از رفتن به پیاده سازی مهم است؟ در این صورت، احتمالاً باید **از رویکرد مبتنی بر برنامه** استفاده کنید.
  - آیا **استراتژی تحویل افزایشی** که نرم افزار را به مشتریان تحویل می دهید و **بازخورد سریع** از آنها دریافت می کنید، واقع بینانه است؟ در این صورت، استفاده از **روش های چابک** را در نظر بگیرید.
  - **اندازه سیستمی** که در حال توسعه است چقدر است؟ **روش‌های چابک** زمانی مؤثرتر هستند که سیستم بتواند با یک **تیم کوچک** هم مکان که می‌توانند به صورت غیررسمی ارتباط برقرار کنند، توسعه یابد.
    - سیستم بزرگ -> ریکرد مبتنی بر برنامه

مسائل سازمانی انسانی و فنی

- **نوع سیستمی که در حال توسعه است:** رویکردهای مبتنی بر برنامه ممکن است برای سیستم هایی که قبل از پیاده سازی نیاز به تجزیه و تحلیل زیادی دارند (به عنوان مثال، سیستم بلادرنگ با الزامات زمان بندی پیچیده) مورد نیاز باشد.
- **عمر مفید مورد انتظار سیستم: سیستم های با عمر طولانی ممکن است به مستندات طراحی بیشتری** نیاز داشته باشند تا بتوانند اهداف اصلی توسعه دهندگان سیستم را به تیم پشتیبانی منتقل کنند.

Dr. A. Taghinezhad

مسائل سازمانی انسانی و فنی

- **فناوری های موجود برای پشتیبانی از توسعه سیستم:** روش های چابک برای پیگیری یک طراحی در حال تکامل به ابزارهای خوبی نیاز دارند.
- **نحوه سازماندهی تیم توسعه:** اگر تیم توسعه توزیع شده باشد یا بخشی از توسعه برون سپاری شده باشد، ممکن است نیاز به تهیه **اسناد طراحی** برای برقراری ارتباط بین تیم های توسعه داشته باشید.

Dr. A. Taghinezhad

مسائل سازمانی انسانی و فنی

- **ملاحظات فرهنگی و سازمانی:** سازمان های مهندسی سنتی فرهنگ توسعه مبتنی بر برنامه را دارند، زیرا این امر در مهندسی رایج است.
- **مهارت طراحان و برنامه نویسان در تیم توسعه چقدر است؟** گاهی اوقات استدلال می شود که **روش های چابک به سطح مهارت بالاتری نسبت به رویکردهای مبتنی بر برنامه نیاز** دارند که در آن برنامه نویسان به سادگی یک طراحی دقیق را به کد ترجمه می کنند.
- **آیا سیستم تابع مقررات خارجی است؟** اگر سیستمی باید توسط یک نهاد تنظیم کننده خارجی (به عنوان مثال، سازمان جهانی هواپیمایی نرم افزاری را تأیید می کند که برای عملکرد یک هواپیما حیاتی است) تأیید شود، احتمالاً باید به عنوان بخشی از پرونده ایمنی سیستم، مستندات دقیق تهیه کنید.

مثال‌ها و جزئیات درباره تعادل بین رویکردهای برنامه‌محور و چابک

- در بسیاری از پروژه‌های نرم‌افزاری، استفاده از عناصر هر دو رویکرد **برنامه‌محور** و **چابک** ضروری است. انتخاب تعادل مناسب بین این دو روش به **عوامل مختلفی** بستگی دارد. در ادامه با ارائه مثال‌ها و توضیحات به این عوامل پرداخته شده است:
- **۱. نیاز به مشخصات دقیق و طراحی پیش از پیاده‌سازی**
- **مثال**: سیستم کنترل یک **ایستگاه تولید برق** که نیاز به هماهنگی دقیق میان اجزای سخت‌افزاری و نرم‌افزاری دارد.
  - **رویکرد**:
    - برنامه‌محور: مستندسازی جامع و دقیق برای اطمینان از عملکرد ایمن و پایدار سیستم.

توضیح: در سیستم‌هایی با **پیچیدگی فنی بالا** و وابسته به صحت عملکرد، نیاز به طراحی دقیق قبل از اجرا وجود دارد.

Dr. A. Taghinezhad

مثال‌ها و جزئیات درباره تعادل بین رویکردهای برنامه‌محور و چابک

- **۲. استراتژی تحویل تدریجی و دریافت بازخورد سریع از مشتریان**
- **مثال: اپلیکیشن سفارش غذای آنلاین** که به بازخورد سریع مشتریان برای بهبود تجربه کاربری نیاز دارد.
- **رویکرد:**
  - چابک: توسعه اپلیکیشن به صورت **اسپرینت‌های کوتاه** و انتشار نسخه‌های مکرر برای دریافت بازخورد.
- **توضیح:** روش چابک به **بازخورد سریع و بهبود مستمر** در محصول کمک می‌کند.

Dr. A. Taghinezhad

مثال‌ها و جزئیات درباره تعادل بین رویکردهای برنامه‌محور و چابک

- **۳. اندازه سیستم و تعداد تیم‌های توسعه**
- **مثال**: **سیستم اطلاعاتی یک دانشگاه** که شامل ماژول‌های مختلف برای مدیریت دانشجویان، اساتید و دوره‌ها است.
- **رویکرد**:
  - برنامه‌محور: برای هماهنگی بین **چندین تیم بزرگ** و جلوگیری از تناقض در بخش‌های مختلف سیستم.
  - چابک: استفاده از تیم‌های کوچک برای توسعه ماژول‌های مجزا.
- **توضیح**: در سیستم‌های بزرگ که نیاز به **تیم‌های بزرگ یا توزیع‌شده** است، برنامه‌محوری مفیدتر است.
- **۴. نوع سیستم در حال توسعه**
- **مثال**: سیستم کنترلی برای **یک خط تولید صنعتی با نیاز به هماهنگی زمانی دقیق**.
- **رویکرد**: برنامه‌محور برای تحلیل دقیق نیازمندی‌های **زمانی و فنی** پیش از پیاده‌سازی.
- **توضیح**: سیستم‌های بلادرنگ نیاز به **برنامه‌ریزی دقیق** قبل از اجرا دارند.

مثال‌ها و جزئیات درباره تعادل بین رویکردهای برنامه‌محور و چابک

- **۵. طول عمر پیش‌بینی‌شده سیستم**
- **مثال: سیستم ثبت مالیات دولتی** که قرار است بیش از ۲۰ سال استفاده شود.
- **رویکرد:** برنامه‌محور برای **مستندسازی جامع** و تسهیل نگهداری توسط تیم‌های آینده.
- **توضیح:** سیستم‌های با **عمر طولانی** نیاز به مستنداتی دارند که مفهوم و اهداف توسعه اولیه را حفظ کنند.
- **۶. ابزارهای پشتیبانی از توسعه سیستم**
- **مثال: پلتفرم مدیریت پروژه آنلاین** که از ابزارهایی مانند **Jira** و **GitHub** استفاده می‌کند.
- **رویکرد:** چابک برای **مدیریت طراحی‌های در حال تغییر** و پیگیری کدهای منتشر شده.
- **توضیح:** روش‌های چابک به **ابزارهای مناسب** برای مدیریت تغییرات وابسته‌اند.

Dr. A. Taghinezhad

مثال‌ها و جزئیات درباره تعادل بین رویکردهای برنامه‌محور و چابک

- **۷. سازماندهی تیم‌های توسعه**
- **مثال**: توسعه یک **پلتفرم تجارت الکترونیک جهانی** با تیم‌های مستقر در کشورهای مختلف.
- **رویکرد**: برنامه‌محور برای ارائه **اسناد طراحی رسمی** و هماهنگی بین تیم‌ها.
- **توضیح**: در تیم‌های توزیع‌شده یا **برون‌سپاری شده**، اسناد رسمی برای ارتباط ضروری هستند.
- **۸. مسائل فرهنگی یا سازمانی**
- **مثال**: یک **شرکت مهندسی عمران** که سیستم مدیریت پروژه‌ای برای خود ایجاد می‌کند.
- **رویکرد**: برنامه‌محور، زیرا این شرکت به **فرهنگ سنتی مهندسی** و مستندسازی پایبند است.
- **توضیح**: در سازمان‌های مهندسی، استفاده از **روش‌های برنامه‌محور** مرسوم است.

Dr. A. Taghinezhad

مثال‌ها و جزئیات درباره تعادل بین رویکردهای برنامه‌محور و چابک

- **۹. مهارت تیم‌های توسعه**
- **مثال**: یک **پروژه تحقیقاتی** با توسعه‌دهندگانی که روی الگوریتم‌های جدید هوش مصنوعی کار می‌کنند.
- **رویکرد**: چابک، زیرا به **مهارت بالا** در تیم برای تغییرات مداوم نیاز دارد.
- **توضیح**: روش‌های چابک به **توسعه‌دهندگان ماهر** برای مدیریت تغییرات بدون دستورالعمل دقیق وابسته هستند.
- **۱۰. سیستم‌های تحت نظارت قوانین خارجی**
- **مثال**: **نرم‌افزار کنترل هواپیما** که نیاز به تاییدیه FAA دارد.
- **رویکرد**: برنامه‌محور برای ارائه **مستندات دقیق** جهت دریافت تاییدیه‌های قانونی.
- **توضیح**: سیستم‌هایی که نیاز به **تاییدیه‌های نظارتی** دارند، به مستندسازی دقیق وابسته هستند.

Extreme programming

- **مثالی از روش چابک: برنامه‌سازی مفرط XP**
  - ممکن است نسخه های جدید چندین بار در روز ساخته شوند.
  - هر ۲ هفته یکبار نسخه ای به مشتریان تحویل داده می شود.
  - تمام تست ها باید برای هر ساخت انجام شوند و ساخت فقط در صورتی پذیرفته می شود که تست ها با موفقیت اجرا شوند.

Dr. A. Taghinezhad

اصول کلیدی XP و چابک

- توسعه افزایشی از طریق **انتشارات کوچک و مکرر** سیستم پشتیبانی می شود.
- مشارکت مشتری به معنای مشارکت **تمام وقت مشتری با تیم** است.
- **افراد نه فرآیند** از طریق **برنامه نویسی زوجی، مالکیت جمعی** و فرآیندی که از ساعات کاری طولانی جلوگیری می کند.
- **پشتیبانی** از **تغییر** از طریق **انتشارات منظم** سیستم.
- حفظ سادگی از طریق **بازسازی مداوم کد refactoring** .

Refactoring

- تیم برنامه نویسی به دنبال بهبودهای احتمالی نرم افزار هستند و این پیشرفت ها را حتی در جایی که نیاز فوری به آنها نباشد، انجام می دهند.
- این کار باعث بهبود **قابلیت درک نرم افزار** می شود و در نتیجه نیاز به مستندات را کاهش می دهد.
- انجام **تغییرات** به دلیل **ساختار خوب و** واضح بودن کد **آسان** تر است.
- با این حال، برخی **تغییرات نیازمند بازنگری معماری** هستند و این بسیار پرهزینه تر است.

Dr. A. Taghinezhad

What is Code Refactoring?

- بازآرایی (Refactoring) به معنای سازماندهی کد شما بدون تغییر عملکرد اصلی آن است. بازآرایی فرآیندی برای ایجاد تغییرات یا تنظیمات کوچک در کد شما بدون تأثیر یا تغییر در نحوه عملکرد کد در هنگام استفاده است.
- **افزایش خوانایی کد (Readability):** کد مرتب و سازماندهی شده، برای خودتان و سایر برنامه‌نویسان خواناتر و قابل فهم‌تر است. این موضوع باعث صرف زمان کمتر برای درک منطق کد و صرف زمان بیشتر برای توسعه و بهبود آن می‌شود.
- **کاهش پیچیدگی کد (Reduced Complexity):** با بازآرایی کد، بخش‌های تکراری حذف شده و منطق کد به شکل ساده‌تری پیاده‌سازی می‌شود. این کار باعث کاهش پیچیدگی کد و رفع باگ‌های احتمالی در آینده می‌شود.
- **تسهیل نگهداری کد (Easier Maintenance):** کدی که به خوبی سازماندهی شده باشد، در آینده به راحتی قابل نگهداری و توسعه است. اضافه کردن ویژگی‌های جدید یا رفع باگ‌ها در چنین کدی، به مراتب ساده‌تر و سریع‌تر انجام می‌شود.
-

نمونه از بازآرایی کد

- **سازماندهی مجدد سلسله مراتب کلاس** برای حذف کد تکراری
- **مرتب کردن و تغییر نام ویژگی ها** و **متدها** برای سهولت درک آنها
- **جایگزینی کد درون خطی** با **فراخوانی متدهایی** که در کتابخانه برنامه گنجانده شده اند.

Dr. A. Taghinezhad

مثلا کد زیر را چطور می‌شود بازآرایی کرد؟

- فرض کنید تابع زیر برای محاسبه مساحت یک شکل بر اساس نوع آن نوشته شده است:

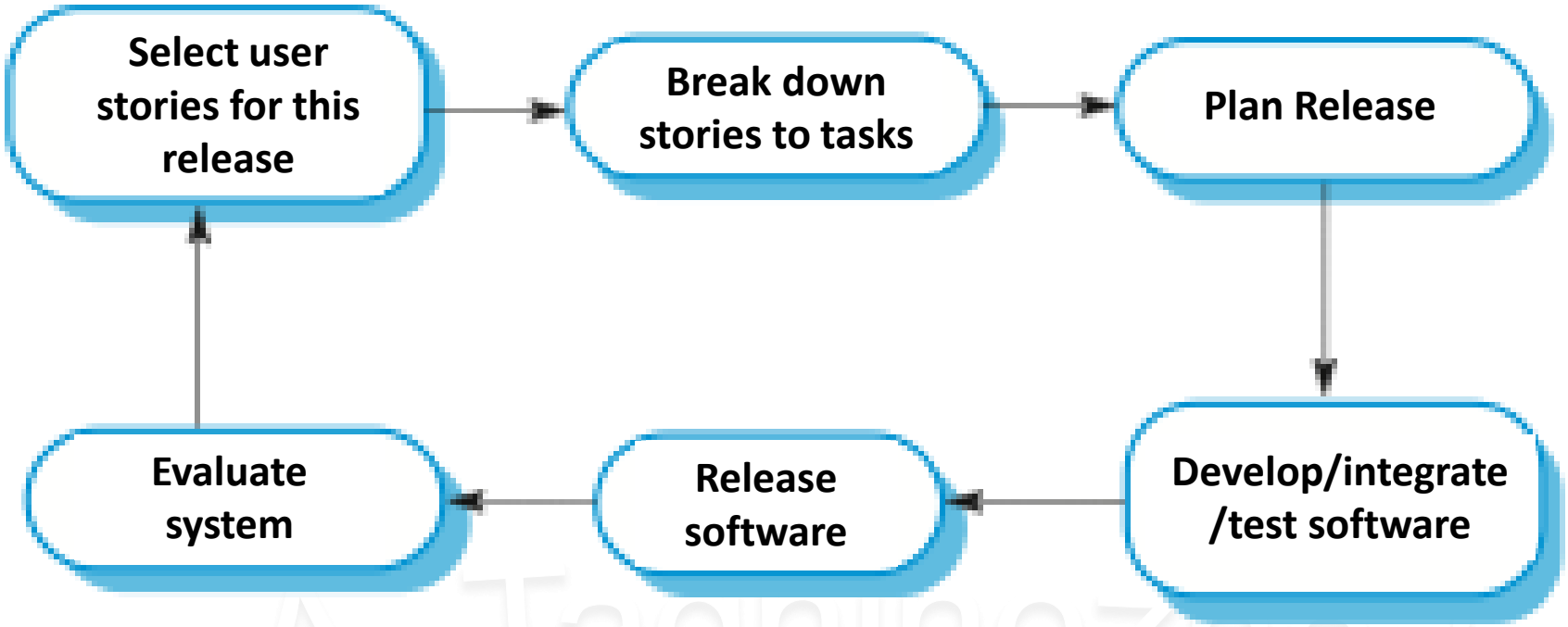
```
public class Shape {

  public double getArea(String type, double... dimensions
    if (type.equals("rectangle")) {
      if (dimensions.length != 2) {
        throw new IllegalArgumentException("Rectangle req
      }
      return dimensions[0] * dimensions[1];
    } else if (type.equals("circle")) {
      if (dimensions.length != 1) {
        throw new IllegalArgumentException("Circle requir
      }
      return Math.PI * Math.pow(dimensions[0], 2);
    } else {
      throw new IllegalArgumentException("Unsupported sha
    }
  }
}
```

(ادامه) بازآرایی کد

```
public class Shape {

  public double getArea(String type, double... dimensions
    if (type.equals("rectangle")) {
      return calculateRectangleArea(dimensions);
    } else if (type.equals("circle")) {
      return calculateCircleArea(dimensions);
    } else {
      throw new IllegalArgumentException("Unsupported sha
    }
  }

  private double calculateRectangleArea(double[] dimensio
    if (dimensions.length != 2) {
      throw new IllegalArgumentException("Rectangle requi
    }
    return dimensions[0] * dimensions[1];
  }

  private double calculateCircleArea(double[] dimensions)
    if (dimensions.length != 1) {
      throw new IllegalArgumentException("Circle requires
    }
    return Math.PI * Math.pow(dimensions[0], 2);
  }
}
```

- استخراج متد (Extract Method)
- می‌توان منطق محاسبه مساحت برای هر شکل را به متدهای جداگانه استخراج کرد. این کار باعث خوانایی و نگهداری بهتر کد می‌شود.

The extreme programming release cycle

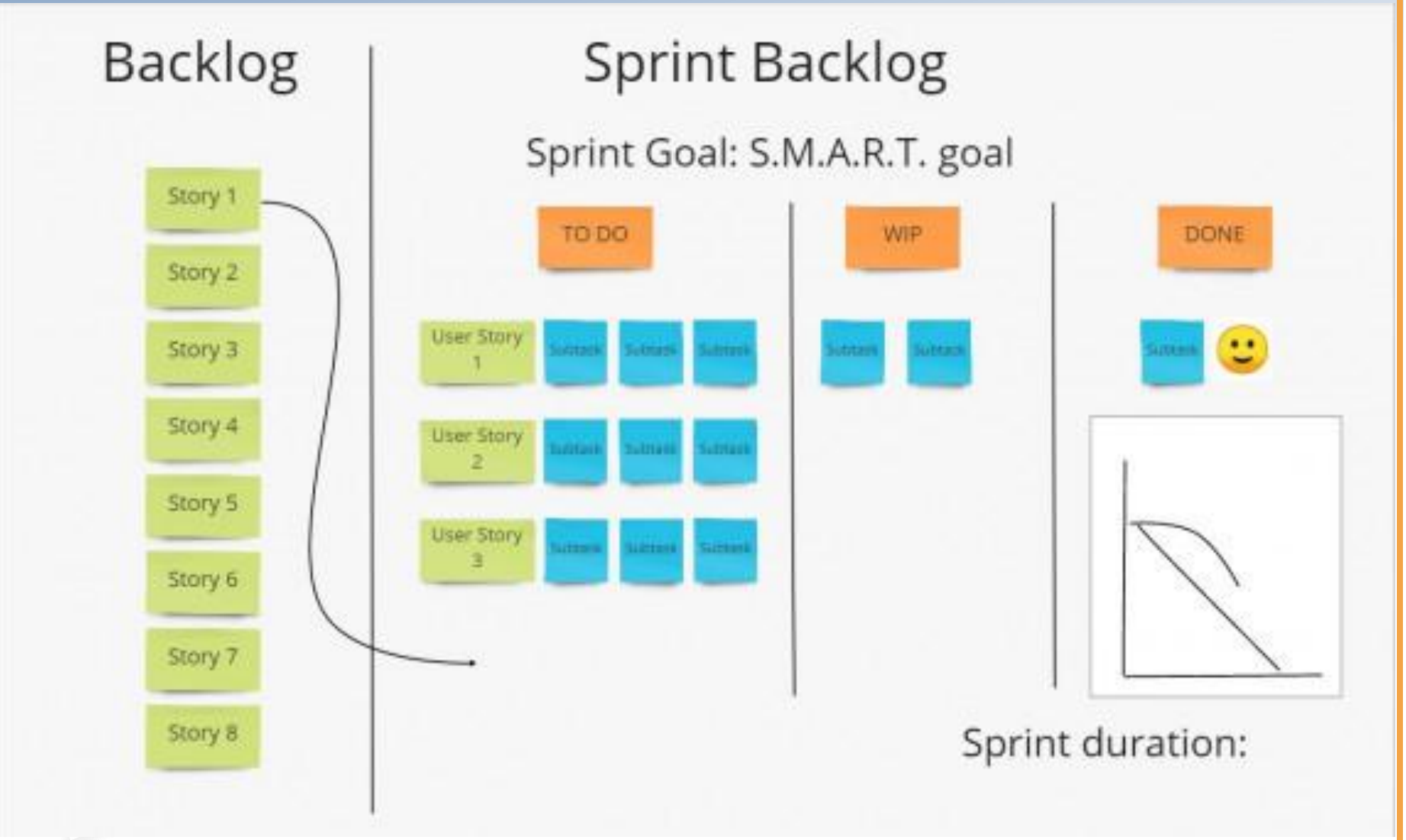


Dr. A. Taghinezhad

داستان کاربر User Story

- هر **داستان کاربری توسط مشتری** نوشته شده و روی یک کارت فهرست قرار می گیرد.
- **مشتری** بر اساس ارزش کلی کسب و کار ویژگی یا عملکرد، ارزشی (یعنی **اولویت**) را به **داستان** اختصاص می دهد.
- سپس اعضای تیم هر **داستان** را **ارزیابی** کرده و **هزینه** ای را بر حسب **هفته** توسعه به آن اختصاص می دهند.
- مهم است که توجه داشته باشید **که داستان های جدید** را می توان در هر **زمان** نوشت.
- مشتریان و توسعه دهندگان با هم کار می کنند تا تصمیم بگیرند که چگونه داستان ها را در نسخه بعدی (افزایش نرم افزار بعدی) که توسط تیم XP توسعه می شود، گروه بندی کنند.

در توسعه چابک(Agile development) ، یک حماسه (Epic) مجموعه‌ای بزرگ از کارها است که می‌توان آن را به قطعات کوچک‌تر و قابل مدیریت‌تر به نام داستان‌های کاربری (User Stories) تقسیم کرد.


Backlog
Sprint Backlog
Sprint Goal: S.M.A.R.T. goal
Story 1
Story 2
Story 3
Story 4
Story 5
Story 6
Story 7
Story 8
TO DO
WIP
DONE
User Story 1
User Story 2
User Story 3
Subtask
Sprint duration:

داستان کاربر User Story

- پس از انجام تعهد اولیه (**توافق** در مورد **داستان‌های گنجانده** شده، **تاریخ تحویل** و سایر موارد پروژه) برای یک نسخه، تیم داستان‌هایی را که در یکی از سه روش توسعه داده می‌شوند، اولویت‌بندی می‌کند:
  - **همه داستان ها بلافاصله** (طی چند هفته) اجرا شوند.
  - **داستان های با بالاترین ارزش** در برنامه بالا رفته و ابتدا اجرا شوند.
  - **ریسک پذیرترین داستان ها** در برنامه بالا رفته و **ابتدا** اجرا شوند.

Dr. A. Taghinezhad

اصول	توضیح
برنامه ریزی افزایشی	• الزامات سیستم روی کارت های داستان ثبت می شوند و داستان هایی که باید در یک نسخه گنجانده شوند بر اساس زمان موجود و اولویت نسبی آنها تعیین می شوند. • توسعه دهندگان این داستان ها را به "وظایف" توسعه تقسیم می کنند
نسخه های منتشر شده کوچک	حداقل مجموعه مفیدی از قابلیت ها که ارزش تجاری را ارائه می دهد، توسعه می یابد. انتشارات سیستم مکررا انجام می شود و به صورت افزایشی قابلیت هایی را به نسخه اولیه اضافه می کند.
طراحی ساده	فقط به اندازه ای که نیازهای فعلی را برآورده کند، طراحی انجام می شود و نه بیشتر.
توسعه آزمون اول Test-First Development (TDD)	قبل از پیاده سازی یک قابلیت جدید، از یک چارچوب تست واحد خودکار برای نوشتن تست های آن قابلیت استفاده می شود.
بازسازی مجدد	از همه توسعه دهندگان انتظار می رود که به محض کشف بهبودهای احتمالی کد، به طور مداوم کد را بازنگری کنند.

برنامه نویسی جفتی	**توسعه دهندگان به صورت زوجی** کار می کنند، کار یکدیگر را **بررسی** می کنند و از هم **حمایت** می کنند تا همیشه کار خوبی ارائه دهند.
مالکیت جمعی	**زوج های توسعه دهنده روی تمام بخش های** سیستم کار می کنند، به طوری که هیچ جزیره ای از تخصص ایجاد نشود و همه توسعه دهندگان مسئولیت کل کد را بر عهده بگیرند. هر کسی می تواند هر چیزی را تغییر دهد.
Continuous integration **ادغام مداوم**	به **محض تکمیل کار روی یک وظیفه، آن را در کل سیستم ادغام می کنند**. پس از هر ادغامی از این دست، همه تست های واحد در سیستم باید با موفقیت اجرا شوند.
سرعت پایدار	کار زیاد در **ساعات غیرکاری قابل قبول نیست**، زیرا اثر خالص آن اغلب کاهش کیفیت کد و بهره وری در میان مدت است.
مشتری در محل	نماینده ای از کاربر نهایی سیستم (**مشتری**) باید به **صورت تمام وقت در اختیار تیم برنامه ریزی** مفرط باشد. در یک فرآیند برنامه ریزی مفرط، مشتری عضوی از تیم توسعه است و مسئول ارائه الزامات سیستم به تیم برای پیاده سازی می باشد.

سناریوهای الزامات

- در XP، **مشتری** یا کاربر بخشی از تیم XPاست و **مسئول** تصمیم گیری در مورد **الزامات** می باشد.
- **الزامات** کاربر **به صورت سناریوها یا داستان های کاربر** بیان می شود.
- **اینها روی کارت نوشته** شده اند و تیم توسعه آنها را به وظایف پیاده سازی تقسیم می کنند. این وظایف **مبنای برآورد زمان و هزینه** هستند.
- مشتری **داستان** ها را برای قرار **گرفتن در نسخه بعدی** بر اساس اولویت ها و برآوردهای زمان بندی انتخاب می کند.

Dr. A. Taghinezhad

داستان تجویز دارو

- رکورد اطلاعات بیمار باید برای ورود اطلاعات باز باشد. روی فیلد دارو کلیک کنید و یکی از موارد «داروی فعلی»، «داروی جدید» یا «لیست داروهای مجاز» را انتخاب کنید.
- در صورت انتخاب «داروی فعلی»، از شما خواسته می شود که دوز را بررسی کنید. اگر می خواهید دوز را تغییر دهید، دوز جدید را وارد کرده و سپس نسخه را تأیید کنید.
- در صورت انتخاب «داروی جدید»X، سیستم فرض می کند که دارویی را که می خواهید تجویز کنید، می دانید. چند حرف اول نام دارو را تایپ کنید. سپس لیستی از داروهای احتمالی را که با این حروف شروع می شوند، مشاهده خواهید کرد. داروی مورد نظر خود را انتخاب کنید. سپس از شما خواسته می شود که تأیید کنید دارویی که انتخاب کرده اید صحیح است. دوز را وارد کرده و سپس نسخه را تأیید کنید.
- ....

داستان تجویز دارو

- در صورت انتخاب «لیست داروهای مجاز»، کادر جستجویی برای لیست داروهای تأیید شده نمایش داده می شود. به دنبال داروی مورد نظر خود بگردید و سپس آن را انتخاب کنید. سپس از شما خواسته می شود که تأیید کنید دارویی که انتخاب کرده اید صحیح است. دوز را وارد کرده و سپس نسخه را تأیید کنید.
- در همه موارد، سیستم بررسی می کند که دوز تجویز شده در محدوده مجاز قرار دارد و در صورتی که خارج از محدوده دوزهای توصیه شده باشد، از شما می خواهد آن را تغییر دهید. پس از تأیید نسخه، برای بررسی نمایش داده می شود. «تأیید» یا «تغییر» را کلیک کنید. اگر «تأیید» را کلیک کنید، نسخه شما در پایگاه داده ثبت می شود. اگر «تغییر» را کلیک کنید، دوباره وارد فرآیند «تجویز دارو» می شوید.

نمونه هایی از کارت های وظیفه برای تجویز دارو

وظیفه اول: تغیر دوز داروی تجویز شده

وظیفه دوم انتخاب فرمول

وظیفه سوم بررسی دوز

- **بررسی دوز یک اقدام احتیاطی** ایمنی برای بررسی اینکه آیا پزشک دوز خطرناک یا کوچک یا بزرگ را تجویز نکرده است.
- با استفاده از شناسه **فرمولاسیون** برای نام **ژنریک دارو**، فهرست **داروهای مجاز را جستجو کرده** و حداکثر و حداقل دوز توصیه شده را بازیابی کنید. دوز تجویز شده را با حداقل و حداکثر مجاز مقایسه کنید.
- در صورت خارج بودن از محدوده، پیام خطایی مبنی بر "دوز بیش از حد مجاز" یا "دوز کمتر از حد مجاز" صادر کنید.
- در صورت قرار گرفتن در محدوده، دکمه "تأیید" را فعال کنید.

Testing in XP

- تست در XPاز اهمیت ویژه‌ای برخوردار است و XPرویکردی را توسعه داده است که در آن برنامه پس از هر تغییری که ایجاد می‌شود، آزمایش می‌شود.
- **ویژگی های تست :XP**
- توسعه مبتنی بر تست Test-First Development
- توسعه تست افزایشی بر اساس سناریوها
- مشارکت کاربر در توسعه و اعتبارسنجی تست
- از چارچوب‌های تست خودکار برای اجرای تمام تست‌های اجزا هر بار که یک نسخه جدید ساخته می‌شود، استفاده می‌شود.

توسعه آزمون اول—Test-first development

- **نوشتن تست قبل از کد، الزاماتی را که باید اجرا شوند را روشن می کند.**
- تست ها به عنوان برنامه نوشته می شوند نه داده، به این ترتیب که می توانند به صورت خودکار اجرا شوند. تست شامل بررسی صحت اجرای آن است.
- معمولاً به یک چارچوب تست مانند Junit متکی است.
- **با اضافه شدن قابلیت جدید، تمام تست های قبلی و جدید به صورت خودکار اجرا می شوند** و بدین ترتیب بررسی می شود که قابلیت جدید باعث ایجاد خطا نشده باشد.

نحوه کار توسعه مبتنی بر تست:

1.نوشتن تست:

1. توسعه‌دهنده با نوشتن یک **تست ناموفق** که رفتار مورد نظر کد را توصیف می‌کند، شروع می‌کند. این تست در ابتدا ناموفق خواهد بود زیرا عملکرد مربوط به آن هنوز پیاده‌سازی نشده است.

2.اجرای تست:

1. توسعه‌دهنده مجموعه تست‌ها را اجرا می‌کند تا **تأیید کند که تست جدید ناموفق است** (این مرحله اطمینان می‌دهد که تست معتبر است و به درستی کمبود عملکرد را شناسایی می‌کند).

3.پیاده‌سازی کد:

1. توسعه‌دهنده به اندازه کافی کد می‌نویسد تا تست را موفق کند. تمرکز بر روی کد حداقلی و کاربردی است که نیازهای تست را برآورده می‌کند.

4.اجرای مجدد مجموعه تست‌ها:

5.بازسازی کد:

1. پس از موفقیت تست، توسعه‌دهنده کد را **بازسازی** می‌کند تا ساختار آن را بدون تغییر رفتار آن بهبود بخشد و اطمینان حاصل کند که کد تمیز و قابل نگهداری است.

6.تکرار:

Dr. A. Taghinezhad

نحوه کار توسعه مبتنی بر تست:

- مثال عملی از توسعه مبتنی بر تست (در پایتون):
- **وظیفه:** ایجاد یک تابع که بررسی کند آیا یک عدد زوج است یا خیر.

1. مرحله ۱: ابتدا تست را بنویسید.

```
import unittest
from my_module import is_even  # Assume this function doesn't
exist yet

class TestIsEven(unittest.TestCase):
    def test_even_number(self):
        self.assertTrue(is_even(4))  # 4 should be even

    def test_odd_number(self):
        self.assertFalse(is_even(5))  # 5 should not be even

if __name__ == '__main__':
    unittest.main()
```

Dr. A. Taghinezhad

نحوه کار توسعه مبتنی بر تست:

- مرحله ۲: تست را اجرا کنید (ناموفق خواهد بود).
- خطا `ModuleNotFoundError: No module named 'my_module':`
(این تأیید می‌کند که تست، کمبود عملکرد را شناسایی می‌کند).
- مرحله ۳: کد را برای موفقیت تست بنویسید.
-

```
# my_module.py
def is_even(n):
    return n % 2 == 0
```

مرحله ۴: تست‌ها را دوباره اجرا کنید.

```
----------------------------------------------------------------------
Ran 2 tests in 0.001s

OK
```

مرحله ۵: بازسازی (در صورت لزوم).

- در این مورد، تابع ساده است، بنابراین ممکن است نیاز به بازسازی نباشد.

Dr. A. Taghinezhad

Customer involvement

- **نقش مشتری در فرآیند تست، کمک به توسعه تست های پذیرش برای داستان هایی است که قرار است در نسخه بعدی سیستم اجرا شوند.**
- مشتری که عضوی از تیم است، همزمان با **پیشرفت توسعه، تست ها را می نویسد**. بنابراین، تمام کدهای جدید برای اطمینان از مطابقت با نیازهای مشتری اعتبارسنجی می شوند.
- با این حال، **افرادی که نقش مشتری را بر عهده می گیرند، زمان محدودی در اختیار دارند** و بنابراین نمی توانند به صورت تمام وقت با تیم توسعه همکاری کنند. آنها ممکن است احساس کنند که ارائه الزامات به عنوان یک مشارکت کافی بوده است و بنابراین تمایلی به مشارکت در فرآیند تست نداشته باشند.

شرح سناریوی تست برای بررسی دوز دارو

ورودی (Input):

1.عددی بر حسب میلی‌گرم (mg) که نشان‌دهنده یک دوز واحد دارو است.

2.عددی که تعداد دوزهای واحد در روز را نشان می‌دهد.

خروجی (Output):

- "تأیید" (OK): در صورتی که دوز کل (دوز واحد در روز × تعداد دوز در روز) در محدوده مجاز قرار گیرد.
- پیام خطا: در صورتی که دوز کل خارج از محدوده مجاز باشد. پیام خطا باید مشخص کند که دوز تجویز شده بیش از حد مجاز است یا کمتر از حد مجاز.

تست‌ها (Tests):

1.تست برای ورودی‌هایی که دوز واحد صحیح است اما تعداد دفعات مصرف در روز بیش از حد مجاز است.

2.تست برای ورودی‌هایی که دوز واحد خیلی زیاد یا خیلی کم است.

3.تست برای ورودی‌هایی که حاصلضرب دوز واحد در تعداد دفعات مصرف در روز، بیش از حد مجاز یا کمتر از حد مجاز است.

4.تست برای ورودی‌هایی که حاصلضرب دوز واحد در تعداد دفعات مصرف در روز، در محدوده مجاز قرار دارد.

خروجی مورد انتظار: "تأیید" (OK)

خودکار سازی تست

- **اتوماسیون تست به این معنی است که تست ها قبل از اجرای کار به عنوان اجزای قابل اجرا نوشته می شوند.**
- این **اجزای تست باید مستقل باشند**، باید شبیه ساز ارسال ورودی برای تست باشند و باید بررسی کنند که **خروجی مطابق با مشخصات خروجی** است.
- یک چارچوب تست خودکار به عنوان مثال Junit سیستمی است که نوشتن تست های قابل اجرا و ارسال مجموعه ای از تست ها برای اجرا را آسان می کند.
- از هر زمان که هر گونه کارایی به سیستم اضافه شود، تست ها قابل اجرا هستند و مشکلاتی که کد جدید ایجاد کرده است را می توان به طور مستقیم شناسایی کرد.

XP و چالش‌های تست

- **برنامه نویسان کد نویسی را به تست ترجیح می دهند و گاهی اوقات هنگام نوشتن تست ها میانبرهایی را انتخاب می کنند.**
- به عنوان مثال، آنها ممکن است تست های ناقصی بنویسند که تمام **استثنائات احتمالی ممکن را بررسی نکنند.**
- نوشتن **برخی از تست ها به صورت افزایشی می تواند بسیار دشوار باشد.** به عنوان مثال، در یک رابط کاربری پیچیده، نوشتن تست های واحد برای کدی که منطق نمایش و گردش کار بین صفحات را اجرا می کند، اغلب دشوار است.
- **قضاوت در مورد کامل بودن مجموعه ای از تست ها دشوار است.** اگرچه ممکن است تست های سیستم زیادی داشته باشید، اما مجموعه تست شما ممکن است پوشش کاملی نداشته باشد.

برنامه‌نویسی زوجی

- **برنامه نویسی زوجی در XP**
- در XP، برنامه نویسان به **صورت زوجی کار می کنند و برای توسعه کد در کنار هم می نشینند.**
- این کار به توسعه **مالکیت مشترک کد و گسترش دانش** در کل تیم کمک می کند.
- این به عنوان یک فرآیند **بررسی غیررسمی** عمل می کند زیرا هر خط کد توسط بیش از یک نفر بررسی می شود. این امر باعث بازنگری کد می شود
- اندازه گیری ها نشان می دهد که بهره وری توسعه با برنامه نویسی زوجی مشابه بهره وری دو فردی است که به طور مستقل کار می کنند.

برنامه‌نویسی زوجی

- **در برنامه نویسی زوجی، برنامه نویسان برای توسعه نرم افزار در کنار هم در یک ایستگاه کاری می نشینند.**
- **زوج ها به صورت پویا** ایجاد می شوند تا همه اعضای تیم در طول فرآیند توسعه با هم کار کنند.
- **اشتراک دانش در حین برنامه نویسی زوجی** بسیار مهم است زیرا ریسک کلی پروژه را در زمان **خروج اعضای تیم** کاهش می دهد.
- برنامه نویسی جفتی یک تکنیک توسعه نرم افزار است که در آن دو برنامه نویس در یک ایستگاه کاری با هم کار می کنند. یک نفر، به نام راننده، به طور فعال کد را می نویسد، در حالی که دیگری، به نام ناوبر، مشاهده می کند، کد را بررسی می کند، و بهبودهایی را پیشنهاد می کند.

مزایای برنامه‌نویسی زوجی

- این روش از **ایده مالکیت و مسئولیت جمعی** برای سیستم پشتیبانی می کند.
- افراد به صورت جداگانه برای **مشکلات کد مسئول شناخته** نمی شوند. در عوض، **تیم مسئولیت جمعی** برای حل این مشکلات بر عهده دارد.
- از آنجایی که هر خط کد توسط حداقل دو نفر بررسی می شود، این روش به عنوان یک **فرآیند بررسی غیررسمی** عمل می کند.
- این روش از بازنگری کد Refactoringپشتیبانی می کند که فرآیندی برای بهبود نرم افزار است.
- در جایی که از برنامه نویسی زوجی و مالکیت مشترک استفاده می شود، دیگران بلافاصله از بازنگری بهره مند می شوند، بنابراین به احتمال زیاد از این فرآیند حمایت می کنند.

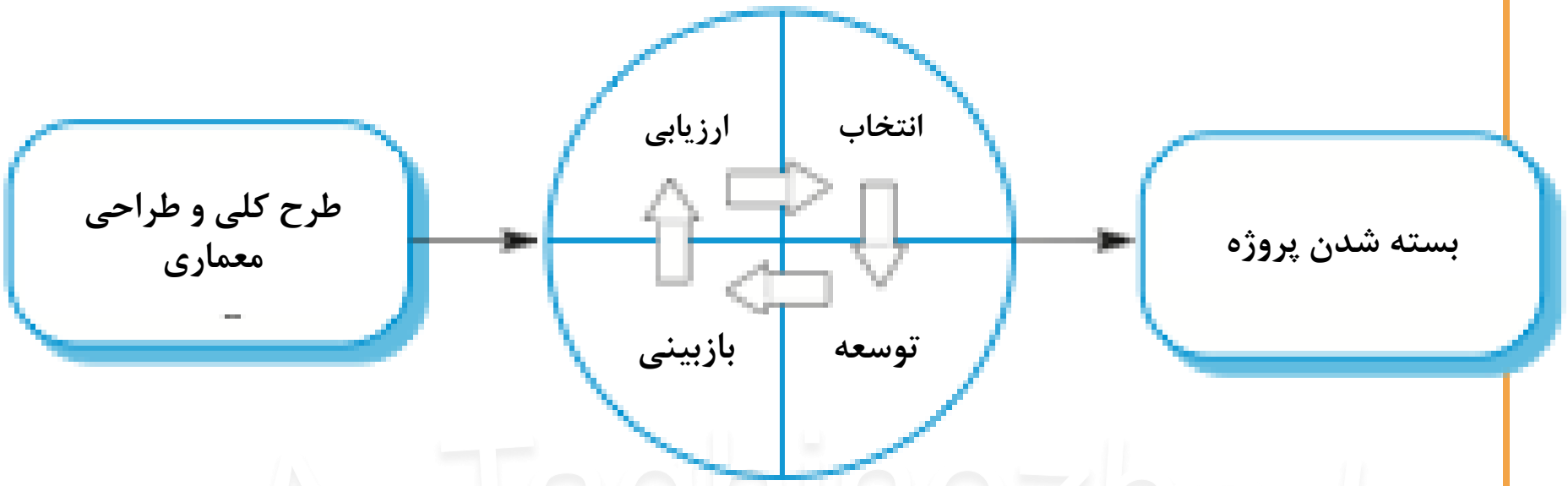
مدیریت پروژه چابک

- مسئولیت اصلی مدیران پروژه نرم افزار،
  - مدیریت پروژه به گونه ای است **که نرم افزار طبق زمانبندی و بودجه برنامه ریزی شده** برای پروژه تحویل داده شود.
- رویکرد **استاندارد برای مدیریت پروژه**، رویکرد مبتنی بر برنامه است.
- مدیران برای پروژه برنامه ای تهیه می کنند که **نشان می دهد چه چیزی باید تحویل داده شود، چه زمانی باید تحویل داده شود و چه کسی روی توسعه تحویل های پروژه کار** خواهد کرد.

اسکرام

- رویکرد اسکرام یک روش چابک کلی است اما تمرکز آن بر مدیریت توسعه تکرارشونده Iterative Development به جای شیوه های چابک خاص است.
- **اسکرام** دارای **سه فاز** است.
- فاز اولیه، **فاز برنامه ریزی کلی است که در آن اهداف کلی پروژه** را تعیین کرده و **معماری نرم افزار** را طراحی می کنید.
- این مرحله به دنبال مجموعه‌ای از چرخه‌های اسپرینت Sprint است که در هر چرخه، **افزایشی از سیستم** توسعه می‌یابد.
- فاز **اختتامیه** پروژه، پروژه را جمع بندی می کند، مستندات مورد نیاز مانند چارچوب های راهنمای سیستم و دستورالعمل‌های کاربر را تکمیل می کند و به ارزیابی درس های آموخته شده از پروژه می پردازد.

فرآیند اسکرام



چرخه اسپرینت

Dr. A. Taghinezhad

چرخه اسکرام

- اسپرینت ها دارای طول ثابتی هستند، **به طور معمول ۲ تا ۴ هفته**. آنها با توسعه یک نسخه از سیستم در XP مطابقت دارند.
- نقطه شروع برای برنامه ریزی، بک‌لاگ محصول Product Backlog است که فهرستی از کارهایی است که باید روی پروژه انجام شود.
- فاز انتخاب شامل **کل تیم پروژه** است که با **مشتری برای انتخاب ویژگی ها** و قابلیت هایی که در طول اسپرینت توسعه داده می شود، کار می کنند.

Dr. A. Taghinezhad

چرخه اسکرام

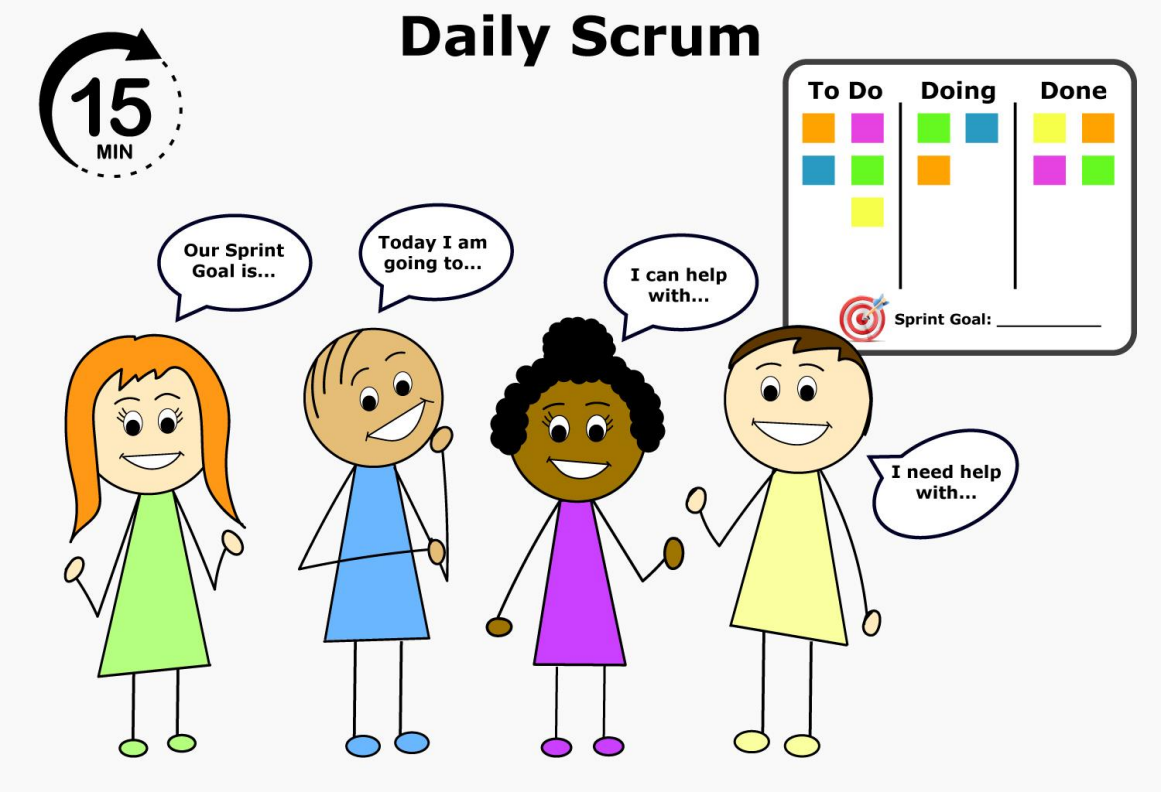
- پس از توافق بر روی این موارد، تیم برای **توسعه نرم افزار** خود را **سازماندهی** می کند. در این مرحله، **تیم از مشتری و سازمان جدا شده و تمامی ارتباطات از طریق فردی به نام «اسکرام مستر»** هدایت می شود.
- نقش **اسکرام مستر محافظت از تیم توسعه** در برابر عوامل **حواس پرتی** خارجی است.
- در پایان اسپرینت، کار **انجام شده بررسی و به ذینفعان** ارائه می شود. سپس چرخه اسپرینت بعدی آغاز می شود.

کار تیمی در اسکرام

- **اسکرام مستر تسهیل** کننده ای است که
  - جلسات روزانه را ترتیب می دهد،
  - لیست کارهای معوقه Backlog را برای انجام کار پیگیری می کند،
  - **تصمیمات** را ثبت می کند،
  - **پیشرفت** را در مقابل بک‌لاگ اندازه گیری می کند
  - با **مشتریان و مدیریت خارج از تیم ارتباط** برقرار می کند.
- **کل تیم در جلسات روزانه کوتاهی** شرکت می کنند که در آن همه **اعضای** تیم **اطلاعات** را به اشتراک می گذارند، **پیشرفت** خود را از زمان **آخرین جلسه**، **مشکلات پیش** آمده و **برنامه ریزی برای روز بعد** را شرح می دهند.
  - این بدان معناست که همه افراد تیم از جریان کار مطلع هستند و در صورت بروز مشکل می توانند برنامه ریزی کوتاه مدت را برای مقابله با آنها دوباره انجام دهند.

اسکرام روزانه چیست؟

- متن: اسکرام روزانه جلسه‌ای کوتاه (حدود ۱۵ دقیقه) است که در ابتدای هر روز کاری در متدولوژی اسکرام برگزار می‌شود.
- تصویر: تصویری از یک تیم که در حال برگزاری جلسه هستند.



اهداف اسکرام روزانه

- لیست اهداف:
  - بررسی پیشرفت انجام‌شده‌ی هر عضو تیم در روز گذشته
  - برنامه‌ریزی برای وظایف روز جاری
  - شناسایی موانع احتمالی و یافتن راه‌حل برای رفع آن‌ها
  - بهبود هماهنگی بین اعضای تیم

Dr. A. Taghinezhad

نمونه‌ای از یک اسکرام روزانه

- جدول:

برنامه‌ی امروز	گزارش روز گذشته	نقش

- ردیف ۱: سرپرست تیم (مریم)
  - گزارش: خوش‌آمدگویی و درخواست گزارش از سایر اعضا
  - برنامه: هدایت جلسه و تسهیل تبادل اطلاعات

Dr. A. Taghinezhad

اعضای تیم

- مالک محصول: حسینی
- اسکرام مستر: رضا
- تیم توسعه: آلیس، باب، چارلی

- مدت زمان اسپرینت: ۲ هفته

Dr. A. Taghinezhad

روز اول: برنامه‌ریزی اسپرینت

- آیتم‌های بک‌لاگ محصول برای اسپرینت:
  - طراحی رابط کاربری برای ویژگی جدید
  - پیاده‌سازی منطق بک‌اند
  - ادغام فرانت‌اند و بک‌اند
  - تست ویژگی جدید
  - مستندسازی ویژگی جدید

Dr. A. Taghinezhad

اسکرام روزانه: روز ۲ تا روز ۱۳

- **دیروز چه کاری انجام دادی؟**
  - آلیس: روی طراحی رابط کاربری کار کردم .
  - باب: شروع به راه‌اندازی منطق بک‌اند کردم .
  - چارلی: به هر دو آلیس و باب کمک کردم .
- **امروز چه کاری انجام خواهی داد؟**
  - آلیس: به کار روی طراحی رابط کاربری ادامه می‌دهم .
  - باب: به کار روی منطق بک‌اند ادامه می‌دهم .
  - چارلی: به محض آمادگی آلیس و باب، روی ادغام شروع به کار می‌کنم .
- **آیا مانعی بر سر راهت وجود دارد؟**
  - آلیس: خیر .
  - باب: با اتصال به پایگاه داده مشکلاتی دارم .
  - چارلی: به باب در حل مشکلش کمک می‌کنم.

Dr. A. Taghinezhad

روز ۱۴: بازنگری اسپرینت و گذشته‌نگری اسپرینت

- **آیتم‌های تکمیل‌شده :**
- طراحی رابط کاربری برای ویژگی جدید
- پیاده‌سازی منطق بک‌اند
- ادغام فرانت‌اند و بک‌اند
- تست ویژگی جدید
- **آیتم‌های تکمیل‌نشده :**
- مستندسازی ویژگی جدید (به اسپرینت بعدی منتقل شد)
- **در طول اسپرینت چه چیزی به خوبی پیش رفت؟**
- **ارتباط موثر** بین اعضای تیم.
- **تکمیل به موقع** وظایف.
- **چه چیزی را می‌توان در اسپرینت بعدی بهبود بخشید؟**
- شروع **مستندسازی** از مراحل اولیه‌ی اسپرینت.

Dr. A. Taghinezhad

مزایای اسکرام

- محصول به **مجموعه** ای از **قطعات قابل مدیریت** و **قابل درک** تقسیم می شود.
- کل **تیم** بر همه چیز **دید دارند** و در نتیجه **ارتباط تیمی** بهبود می یابد.
- مشتریان شاهد **تحویل به موقع افزایش ها** هستند و بازخورد در مورد نحوه عملکرد محصول را دریافت می کنند.
- اعتماد بین مشتریان و توسعه دهندگان برقرار

Dr. A. Taghinezhad

روش مقیاس پذیری در اسکرام

- روش‌های چابک برای پروژه‌های کوچک و متوسطی که توسط یک تیم کوچک هم مکان توسعه داده می‌شوند، موفق بوده‌اند.
- گاهی اوقات گفته می‌شود که موفقیت این روش‌ها به دلیل بهبود ارتباطاتی است که با همکاری همه افراد حاصل می‌شود.
- مقیاس گذاری روش‌های چابک نیازمند **تغییر آن‌ها برای تطبیق با پروژه‌های بزرگ‌تر و طولانی‌تر** با تیم‌های توسعه متعدد است که شاید در مکان‌های مختلف کار کنند.

روش مقیاس پذیری در اسکرام

- قیاس‌پذیری اسکرام در یک پروژه‌ی بزرگ بانکی
- زمینه مسئله
- یک شرکت بزرگ مالی تصمیم دارد یک «پلتفرم بانکداری دیجیتال» شامل ماژول‌هایی مانند مدیریت حساب، پرداخت، تحلیل تراکنش، احراز هویت و تقلب‌سنجی توسعه دهد. این سامانه بسیار بزرگ است و تنها با یک تیم کوچک قابل توسعه نیست. شرکت قبلاً با روش آبشاری کار می‌کرده اما اکنون قصد دارد از اسکرام در مقیاس سازمانی استفاده کند.

چالش‌های سیستم‌های بزرگ و توسعه چابک

عوامل چالش‌زا

تیم‌های توزیع‌شده
توسعه در مکان‌ها و ساعت‌های مختلف

پیچیدگی یکپارچه‌سازی
ادغام با سیستم‌های موجود

الزامات رگولاتوری
محدودیت‌های خارجی و قانونی

چرخه‌های طولانی
زمان طولانی تأمین و توسعه

جابجایی نیرو
دشواری حفظ دانش سیستم

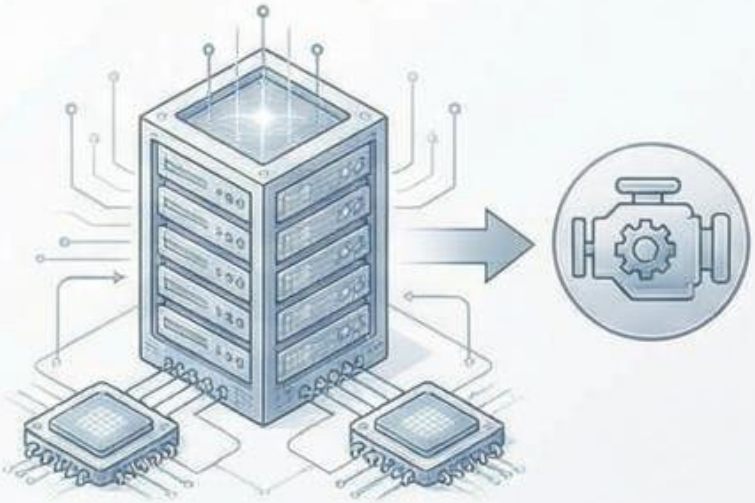
ذینفعان متنوع
چالش در مشارکت همگانی

پیامدها و اثرات

- کاهش انعطاف‌پذیری افزایشی
- افزایش هزینه پیکربندی
- دشواری حفظ دانش سیستمی
- عدم مشارکت کامل ذینفعان

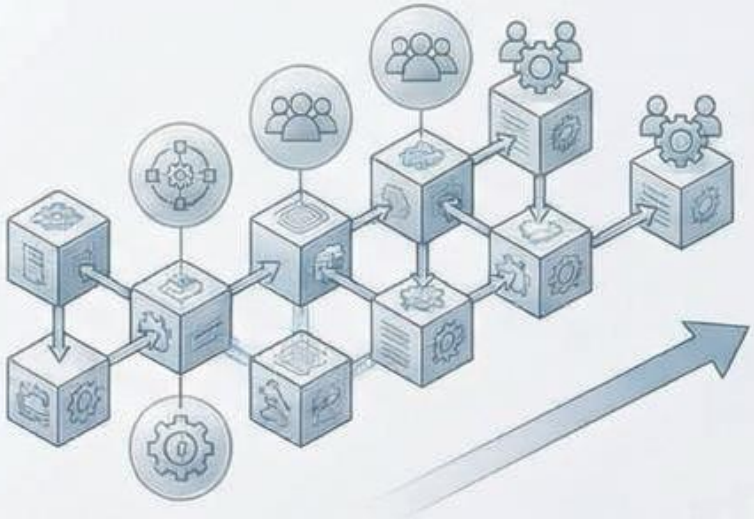
تفاوت‌های کلیدی: Scaling Up و Scaling Out در اسکرام

Scaling Up (ارتقا پذیری سامانه)



به ارتقای معماری سامانه برای پردازش بار بیشتر اشاره دارد؛ تمرکز بر بهبود منابع سخت‌افزاری و نرم‌افزاری یک سیستم واحد است.

Scaling Out (توسعه‌پذیری سازمان)



به گسترش تعداد تیم‌ها و نهادهای سازمانی برای پوشش سیستم‌های نرم‌افزاری بزرگ اشاره دارد؛ در اسکرام اهمیت بیشتری دارد، زیرا چند تیم باید همزمان روی اجزای مختلف مختلف کار کنند بدون از دست رفتن اصول چابک.

اسکرام چابک در مقیاس بزرگ

روش‌های مقیاس‌پذیری اسکرام
Scaling Up:ارتقاپذیری سامانه

- تمرکز روی انطباق اسکرام با *پروژه‌ی بزرگ*:
- **تقسیم سامانه به ماژول‌های مستقل**
  هر تیم اسکرام مسئول یک سرویس یا ماژول مانند «پرداخت» یا «هویت‌سنجی» است.
- **هماهنگی میان تیم‌ها از طریق Scrum of Scrums**
  نمایندگان تیم‌ها روزانه یا هفتگی برای مدیریت وابستگی‌ها جلسه برگزار می‌کنند.
- **معماری تکاملی Evolutionary Architecture**
  به‌جای طراحی سنگین اولیه، ساختار سامانه با هر انتشار به‌صورت تدریجی غنی می‌شود.
- **بازنشرهای زمان‌بندی‌شده و یکپارچه‌سازی مداوم**
  یک محیط CI/CDمرکزی تضمین می‌کند که ماژول‌های تیم‌های مختلف همواره با هم سازگار باشند.
- **حل وابستگی‌های متقابل از طریق Product Owner‌های هماهنگ**
  هر تیم یک POدارد، اما یک **Chief PO**نقشه‌ی محصول و وابستگی‌ها را مدیریت می‌کند.

Dr. A. Taghinezhad

روش‌های مقیاس‌پذیری اسکرام
Scaling Outتوسعه‌پذیری سازمانی

- تمرکز روی چگونگی *تغییر فرهنگ و ساختار سازمان*:
- **تغییر فرایندهای قدیمی سازمان**
  واحدهایی که به روش آبشاری عادت دارند (مانند مدیریت تغییر یا امنیت) باید با اسپرینت‌های کوتاه سازگار شوند.
- **ایجاد Chapterها و Guildها**
  متخصصان مشابه (امنیت، تست، پایگاه داده) از تیم‌های مختلف در انجمن‌هایی مشترک، استانداردهای سازمانی را هماهنگ می‌کنند.
- **حمایت مدیریت ارشد**
  برای موفقیت اسکرام در مقیاس، مدیریت باید اختیار بدهد، موانع اداری را کاهش دهد و جریان کار را ساده کند.
- **آموزش گسترده سازمان**
  همهٔ نقش‌ها—از مدیران تا تحلیلگران—آموزش اسکرام چندتیمی می‌بینند تا زبان مشترک ایجاد شود.
- **تدوین Definition of Doneسازمانی**
  تا همه تیم‌ها معیارهای یکسانی برای کیفیت، امنیت، تست و مستندسازی داشته باشند.

Dr. A. Taghinezhad

توسعه سیستم‌های بزرگ

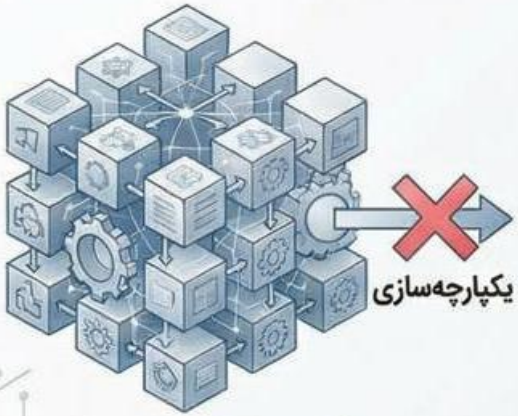
- سیستم‌های بزرگ معمولاً مجموعه‌ای از سیستم‌های مستقل و مرتبط هستند که توسط **تیم‌های جداگانه** توسعه می‌یابند.
- این تیم‌ها اغلب **در مکان‌های مختلف و گاهی با اختلاف ساعت کار می‌کنند**. به این معنی که با سیستم‌های موجود ادغام و با آن‌ها تعامل دارند.
- بسیاری از **الزامات سیستم بر این تعامل تمرکز** دارند و به همین دلیل **انعطاف‌پذیری آن‌ها برای توسعه افزایشی کمتر** است.
- بخش قابل توجهی از توسعه برای سیستم‌های یکپارچه بزرگ، مربوط به پیکربندی سیستم به جای توسعه کد جدید است.

Dr. A. Taghinezhad

چالش‌ها و راهکارهای یکپارچه‌سازی در سیستم‌های بزرگ

استراتژی‌هایی برای مدیریت پیچیدگی و حفظ چابکی

چالش: یکپارچه‌سازی پیوسته دشوار

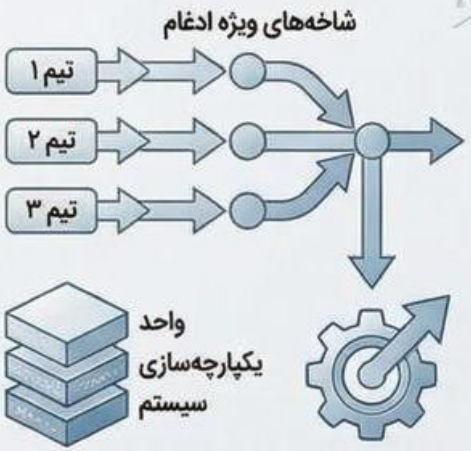


یکپارچه‌سازی کامل در هر کامیت ممکن نیست

راهبرد فرکانس: ساخت و انتشار منظم

انتشار منظم هر چند هفته یکبار ضروری است

ابزارهای فرآیند: مدیریت ریسک



تست‌های لایه‌ای

استقرار خودکار

کاهش ریسک

بازخورد سریع

امکان تصحیح مسیر در بازه‌های کوتاه

توسعه سیستم‌های بزرگ (ادامه)

- **قوانین و مقررات خارجی** اغلب **فرآیندهای توسعه** را برای سیستم‌های بزرگ محدود می‌کنند.
- این پروژه‌ها **چرخه‌های طولانی تأمین و توسعه** دارند و **حفظ تیم‌های منسجمی** که دانش سیستم را دارند، **دشوار** است زیرا اعضای تیم به ناچار به پروژه‌های دیگر منتقل می‌شوند.
- مجموعه **متنوعی از ذینفعان** در سیستم‌های بزرگ باعث می‌شود **که مشارکت همه آن‌ها در فرآیند توسعه** تقریباً **غیرممکن** باشد.

Scaling out(**توسعه پذیری سازمان**) **و** scaling up **(ارتقا پذیری سامانه)**

- "**مقیاس گذاری**" به استفاده از روش‌های چابک برای سیستم‌های نرم‌افزاری بزرگ اشاره دارد که به **بیش از یک تیم** نیاز دارند. "مقیاس‌گذاری برون‌سازمانی" بر چگونگی معرفی روش‌های چابک در یک **سازمان بزرگ** با تجربه گسترده توسعه نرم‌افزار تمرکز دارد.
- در اینجا چند نکته کلیدی برای به خاطر سپردن هنگام مقیاس گذاری روش‌های چابک آورده شده است:
- حفظ اصول چابک: **برنامه‌ریزی انعطاف‌پذیر، انتشارهای مکرر، یکپارچه‌سازی مداوم، توسعه مبتنی بر تست** و ارتباطات خوب تیمی.

Scaling out(توسعه پذیری سازمان) و scaling up (ارتقا پذیری سامانه)

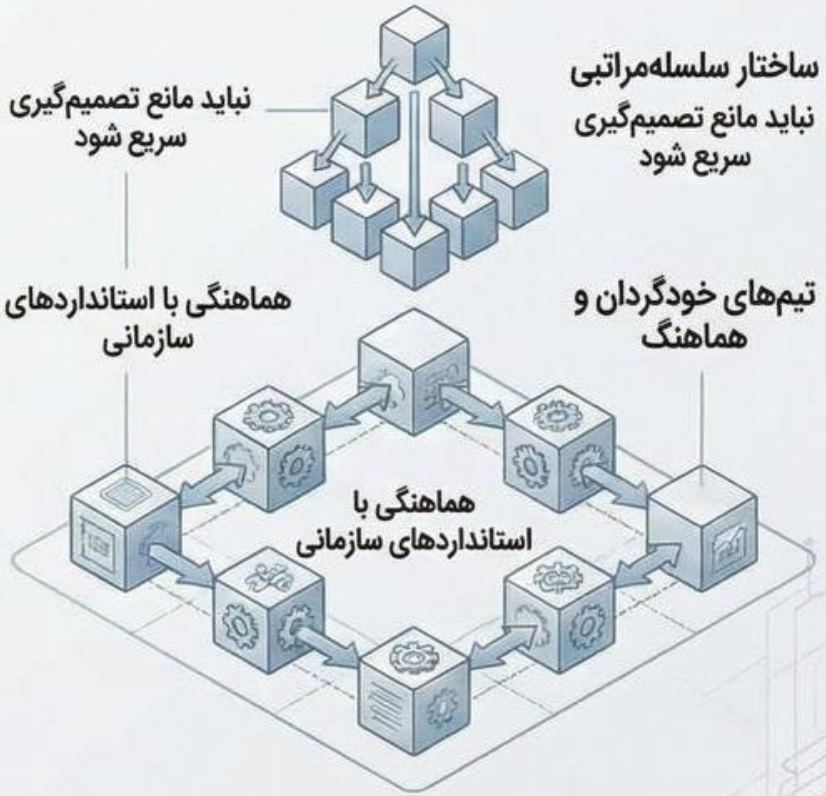
- **مقیاس‌گذاری در حوزه توسعه نرم‌افزار**
- به معنای به‌کارگیری روش‌های چابک برای پروژه‌ها و سیستم‌های نرم‌افزاری بزرگ است که به تعداد زیادی از تیم‌ها نیاز دارند.
  - برای مثال، اگر یک شرکت نرم‌افزاری بخواهد سیستم پیچیده‌ای مانند پلتفرم بانکی یا شبکه اجتماعی را توسعه دهد، تنها یک تیم کوچک کافی نیست؛ بلکه به چندین تیم نیاز دارد که همزمان بر روی بخش‌های مختلف این سیستم کار کنند.
  - مقیاس‌گذاری چابک به این تیم‌ها کمک می‌کند تا همچنان با اصول چابک، یعنی سرعت بالا، انعطاف‌پذیری و هم‌آهنگی بالا کار کنند.
- **مقیاس‌گذاری برون‌سازمانی:**
- این نوع مقیاس‌گذاری به نحوه پیاده‌سازی اصول و فرآیندهای چابک در سازمان‌های بزرگ اشاره دارد؛ به ویژه در شرکت‌هایی که پیشینه طولانی در توسعه نرم‌افزار دارند و ممکن است روش‌های سنتی را به کار گرفته باشند.
  - برای مثال، یک شرکت با تجربه که پیش‌تر از روش‌های آبشاری استفاده می‌کرده، ممکن است بخواهد به روش‌های چابک تغییر کند، اما نیاز به راهبردهای مشخصی برای تطبیق این روش‌ها با سازمان خود دارد.

عوامل موفقیت در توسعه‌پذیری سازمان (Scaling Out)

حفظ اصول چابک و تعادل بین خودگردانی و هماهنگی

اصول کلیدی چابک برای حفظ

ساختار سازمانی و تصمیم‌گیری



ارتقا پذیری برای سامانه‌های بزرگ

- افزایش طراحی اولیه و مستندات سیستم: **سیستم‌های بزرگ** به **برنامه‌ریزی اولیه بیشتری نسبت به پروژه‌های کوچک** نیاز دارند.
- **مکانیزم‌های ارتباط بین تیم‌ها: برگزاری منظم کنفرانس‌های تلفنی** و **ویدیویی** به همراه جلسات **الکترونیکی کوتاه و مکرر** برای به‌روزرسانی پیشرفت تیم‌ها ضروری است.
- **تمرکز بر ساخت و انتشار مکرر سیستم**: در حالی که یکپارچه‌سازی مداوم (ساخت کل سیستم هر بار که تغییری ایجاد می‌شود) ممکن است امکان‌پذیر نباشد، **ساخت‌های مکرر و انتشارهای منظم** سیستم ضروری هستند.

توسعه پذیری برای سازمان‌های بزرگ

- **مقاومت مدیر پروژه: مدیرانی** که با روش‌های چابک آشنایی ندارند، ممکن است نسبت به **پذیرش ریسک** رویکرد جدید **تردید** داشته باشند.
- رویه‌ها و استانداردهای کیفیت ناسازگار: سازمان‌های بزرگ اغلب رویه‌ها و **استانداردهای کیفیتی تثبیت‌شده‌ای** دارند که ممکن است با ماهیت **انعطاف‌پذیر روش‌های چابک در تضاد** باشد.
- **سطوح مهارت متنوع**: اعضای تیم در سازمان‌های بزرگ ممکن است مجموعه مهارت‌های گسترده‌ای داشته باشند که می‌تواند بر اثربخشی اتخاذ شیوه‌های چابک تأثیر بگذارد.
- **مقاومت فرهنگی** در برابر روش‌های چابک: **سازمان‌هایی که سابقه طولانی** در استفاده از فرآیندهای مهندسی سیستم‌های متعارف دارند، ممکن است **مقاومت فرهنگی در برابر روش‌های چابک** داشته باشند.

راهکارهای غلبه بر چالش‌های فرهنگی و مهارتی در سازمان‌های بزرگ

چالش‌ها

سطوح مهارت متفاوت
تنوع گسترده در تجربه و تخصص اعضای تیم

فرهنگ مهندسی سنتی
مقاومت در برابر تغییر و فرآیندهای تثبیت‌شده

راهکارها و اقدامات

جشن موفقیت‌های کوچک
افزایش انگیزه و ترویج فرهنگ یادگیری

تعریف مسیر رشد چابک
توسعه مهارت‌ها و پیشرفت شغلی

جفت‌برنامه‌نویسی بین تیم‌ها
یادگیری متقابل و بهبود کیفیت کد

ایجاد جوامع عملی تجربیات
اشتراک دانش و بین متخصصان

هدف نهایی: ترویج یادگیری مستمر و کاهش مقاومت فرهنگی

پایان

Dr. A. Taghinezhad