مهندسی نرم‌افزار

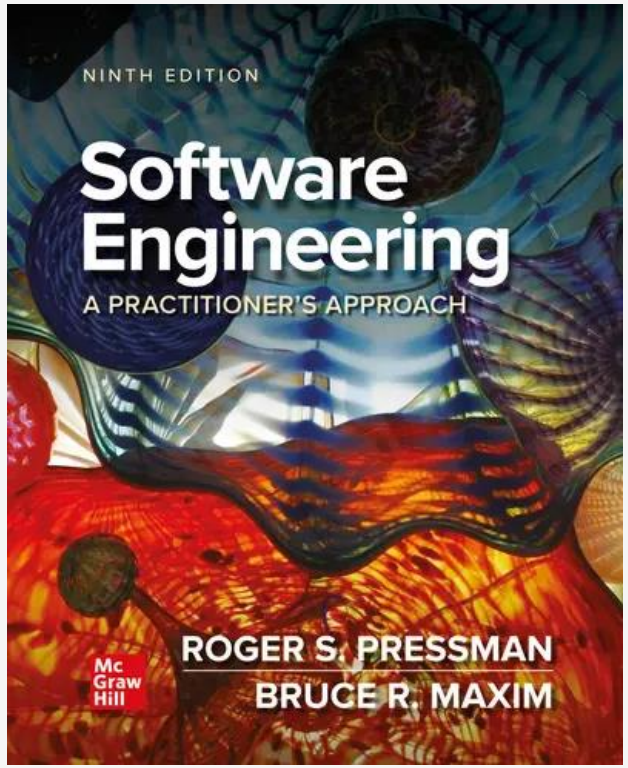
Mahmoud Matinfar

www.matinfar.ir

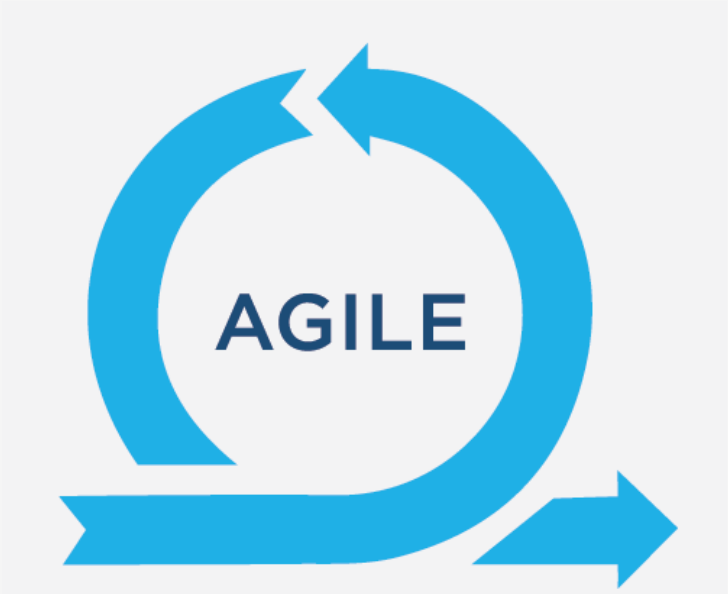
matinfar.ir@gmail.com

@matinfar_ir

منابع درسی



کتاب مهندسی نرم‌افزار
اثر راجر اس پرسمن و بروس آر ماکسیم
ویرایش نهم



بیانیه چابک (Agile Manifesto)
راهنمای اسکرام (Scrum Guide)
و....

مهندسی نرم افزار

Software Engineering

۱ نرم‌افزار و مهندسی نرم‌افزار

Mahmoud Matinfar
matinfar.ir
@matinfar_ir
matinfar.ir@gmail.com

1. ماهیت نرم افزار

تعریف نرم افزار

نرم‌افزار شامل برنامه‌های کامپیوتری، ساختارهای داده و اطلاعات توصیفی است که برای اجرا در کامپیوترها طراحی شده‌اند.

نرم‌افزار شامل سه جزء اصلی است:

(1) دستورات (برنامه‌های کامپیوتری) که هنگام اجرا، ویژگی‌ها، عملکرد و کارایی مورد نظر را ارائه می‌دهند؛

(2) ساختارهای داده‌ای که به برنامه‌ها امکان دستکاری مناسب اطلاعات را می‌دهند؛

(3) اطلاعات توصیفی در قالب‌های چاپی و مجازی که نحوه عملکرد و استفاده از برنامه‌ها را شرح می‌دهند.

1. ماهیت نرم افزار

انواع نرم افزار

1. نرم افزارهای عمومی (Generic): این سیستم ها مستقل ((stand-aloneهستند که توسط یک سازمان توسعه دهنده تولید شده‌اند و در بازار آزاد به هر مشتری که قادر به خرید آن باشد، فروخته می‌شوند. مثال برای این نوع محصولات میتوان: محصولاتی شامل اپ های موبایل، نرم‌افزار برای Pcها مانند پایگاه‌داده ها، ویرایشگر های متن، پکیج های طراحی و ابزار های مدیریت پروژه اشاره کرد.

2. نرم افزارهای سفارشی (Customized): اینها سیستم‌هایی هستند که توسط مشتری خاصی و برای همان مشتری توسعه داده شده اند. پیمانکار نرم افزاری را برای مشتری و مختص آن طراحی و پیاده سازی می کند. مثال بر این توع نرم‌افزارها: سیستم های کنترلی برای دستگاه‌های الکترونیکی، سیستم هایی که برای پشتیبانی فرآیند تجاری مشخصی نوشته می‌شوند و سیستم های کنترل ترافیک هوایی.

تفاوت:

تمایز مهم بین این سیستم ها این هست که در محصولات عمومی، سازمانی که محصول را توسعه داده است، مشخصات نرم‌افزار را نیز کنترل می کند. به این معنی که اگر به مشکلات توسعه برخورد کنند، آنها می توانند مجدد فکر کنند که چه چیزی درحال توسعه است. در محصولات سفارشی، مشخصات توسط سازمانی که نرم‌افزار را خریداری کرده است، کنترل و توسعه داده می شوند. توسعه دهندگان نرم‌افزار باید با همان مشخصات کار کنند.

1.2. تعریف رشته

تعریف مهندسی نرم افزار

موسسه مهندسی برق و الکترونیک IEEE تعریف زیر را برای مهندسی نرم‌افزار ارائه داده است:

مهندسی نرم‌افزار: کاربرد روشی نظام‌مند، منظم و قابل کمیّت‌سازی در توسعه، راه‌اندازی و نگهداری نرم‌افزار؛ به عبارت دیگر، کاربرد مهندسی در نرم‌افزار.

این رشته بر پایه‌ای از کیفیت استوار است و شامل فرآیندها، روش‌ها و ابزارهایی است که به توسعه‌دهندگان کمک می‌کنند تا نرم‌افزارهای با کیفیت بالا را تولید کنند.

چرا مهندسی نرم‌افزار مهم است؟

نرم‌افزار در تمام جنبه‌های زندگی ما نفوذ کرده و نیاز به ساخت سیستم‌های پیچیده با کیفیت بالا را افزایش داده است. مهندسی نرم‌افزار به توسعه‌دهندگان کمک می‌کند تا رویکرد خود را به گونه‌ای تطبیق دهند که بهترین نتیجه را به دست آورند.

۱.۲. تعریف رشته

لایه‌های مهندسی نرم‌افزار

مهندسی نرم‌افزار یک فناوری لایه‌ای است که شامل چهار لایه اصلی است:

۱. **تمرکز بر کیفیت (Quality Focus):** این لایه پایه و اساس مهندسی نرم‌افزار است. تمرکز بر کیفیت به این معناست که تمام فعالیت‌ها، روش‌ها و ابزارهای مورد استفاده در فرآیند توسعه نرم‌افزار باید با هدف دستیابی به کیفیت بالا طراحی شوند.

۲. **فرآیند (Process):** لایه فرآیند، چارچوبی را برای توسعه نرم‌افزار فراهم می‌کند. این لایه شامل مجموعه‌ای از فعالیت‌ها، اقدامات و وظایف است که برای ایجاد یک محصول نرم‌افزاری انجام می‌شوند.

۳. **روش‌ها (Methods):** لایه روش‌ها شامل تکنیک‌های فنی است که برای ساخت نرم‌افزار استفاده می‌شوند. این روش‌ها به توسعه‌دهندگان کمک می‌کنند تا نیازمندی‌ها را تحلیل کنند، طراحی کنند، کد بنویسند و نرم‌افزار را تست کنند.

۴. **ابزارها (Tools):** لایه ابزارها شامل پشتیبانی خودکار یا نیمه‌خودکار برای فرآیند و روش‌ها است. این ابزارها به توسعه‌دهندگان کمک می‌کنند تا کارهای خود را به صورت کارآمدتر و با خطای کمتر انجام دهند.

1.3. فرآیند نرم افزار

فرآیند نرم افزار

فرآیند نرم‌افزار مجموعه‌ای از فعالیت‌ها، اقدامات و وظایف است که برای ایجاد یک محصول نرم‌افزاری انجام می‌شود. این فرآیند باید انعطاف‌پذیر و قابل تطبیق با نیازهای پروژه باشد.

چارچوب فرآیند

چارچوب فرآیند، با شناسایی تعداد کمی از فعالیت‌های چارچوب که برای همه پروژه‌های نرم‌افزاری، صرف نظر از اندازه یا پیچیدگی آن‌ها، قابل اجرا هستند، بنیاد یک فرآیند مهندسی نرم‌افزار کامل را ایجاد می‌کند. علاوه بر این، چارچوب فرآیند شامل مجموعه‌ای از فعالیت‌های فراگیر است که در کل فرآیند نرم‌افزار قابل اجرا هستند.

یک چارچوب فرآیند عمومی برای مهندسی نرم‌افزار شامل پنج فعالیت است:

۱. **ارتباطات** (Communication): درک نیازهای ذینفعان و جمع‌آوری الزامات.

۲. **برنامه‌ریزی** (Planning): ایجاد نقشه‌ای برای پروژه، شامل وظایف فنی، منابع و زمان‌بندی.

۳. **مدل‌سازی** (Modeling): ایجاد مدل‌هایی برای درک بهتر نیازها و طراحی راه‌حل.

۴. **ساخت** (Construction): تولید کد و تست آن برای کشف خطاها.

۵. **استقرار** (Deployment): تحویل نرم‌افزار به مشتری و دریافت بازخورد.

1.3. فرآیند نرم افزار

فعالیت‌های چتری (Umbrella Activities)

چارچوب فرآیند مهندسی نرم‌افزار توسط تعدادی فعالیت چترگونه تکمیل می‌شود. به‌طور کلی، فعالیت‌های چترگونه در طول یک پروژه نرم‌افزاری اعمال می‌شوند و به تیم نرم‌افزار کمک می‌کنند تا پیشرفت، کیفیت، تغییر و ریسک را مدیریت و کنترل کنند.

این فعالیت‌ها در طول پروژه نرم افزاری اعمال می‌شوند و شامل موارد زیر است:

- مدیریت پروژه: پیگیری پیشرفت و کنترل پروژه.
- مدیریت ریسک: شناسایی و مدیریت ریسک‌های پروژه.
- تضمین کیفیت: اطمینان از کیفیت نرم‌افزار.
- بررسی‌های فنی: ارزیابی محصولات کاری برای کشف خطاها.
- مدیریت پیکربندی: مدیریت تغییرات در طول فرآیند.
- مدیریت قابلیت استفاده مجدد: برنامه‌ریزی برای استفاده مجدد از کد و طراحی.

1.3. فرآیند نرم افزار

انطباق فرآیند

فرآیند مهندسی نرم‌افزار، نسخه‌ای خشک و غیرقابل تغییر نیست که تیم نرم‌افزاری الزاماً باید به طور دقیق از آن پیروی کند. بلکه باید چابک و قابل انطباق باشد (با مسئله، با پروژه، با تیم و با فرهنگ سازمانی). بنابراین، فرآیندی که برای یک پروژه انتخاب می‌شود، ممکن است به طور قابل توجهی با فرآیندی که برای پروژه دیگری انتخاب می‌شود، متفاوت باشد.

از جمله این تفاوت‌ها می‌توان به موارد زیر اشاره کرد:

- جریان کلی فعالیت‌ها، اقدامات و وظایف و وابستگی‌های متقابل بین آن‌ها
- میزان تعریف اقدامات و وظایف در هر فعالیت چارچوب
- میزان شناسایی و الزام محصولات کاری
- شیوه اعمال فعالیت‌های تضمین کیفیت
- شیوه اعمال فعالیت‌های پیگیری و کنترل پروژه
- میزان کلی جزئیات و دقت در توصیف فرآیند
- میزان مشارکت مشتری و سایر ذینفعان در پروژه
- سطح خودمختاری داده شده به تیم نرم‌افزار
- میزان تعیین سازماندهی تیم و نقش‌ها

1.3. شروع یک پروژه نرم افزاری

هر پروژه نرم‌افزاری با یک نیاز کسب‌وکار شروع می‌شود. این نیاز ممکن است شامل تصحیح یک نقص، تطبیق یک سیستم قدیمی یا ایجاد یک محصول جدید باشد.

مهندسی نرم‌افزار شامل چهار گام اصلی است:

۱. درک مسئله: شناسایی ذینفعان و نیازهای آن‌ها.

۲. برنامه‌ریزی راه‌حل: طراحی راه‌حل بر اساس الگوها و راه‌حل‌های موجود.

۳. اجرای برنامه: تولید کد و تست آن.

۴. بررسی نتیجه: تست نرم‌افزار برای اطمینان از صحت آن.

1.6. ویژگی های نرم افزار خوب

1. مقبولیت (Acceptability)

نرم افزار باید برای کاربرانی که برای آنها طراحی شده، قابل پذیرش باشد. به این معنی که باید قابل فهم باشد، قابل استفاده و سازگار با دیگر سیستم هایی که آنها استفاده می کنند، باشد.

2. قابلیت اطمینان و امنیت (Dependability and security)

قابلیت اطمینان نرم افزار شامل گستره ای از ویژگی ها از جمله قابلیت اعتماد، امنیت و ایمنی می باشد. نرم افزار قابل اعتماد نباید بهنگام رخداد خطای سیستمی، منجر به آسیب های فیزیکی یا اقتصادی شود. نرم افزار باید امن باشد و در نتیجه کاربران مخرب نتوانند به سیستم دسترسی داشته باشند یا به آن آسیب وارد کنند.

3. کارایی (Efficiency)

نرم افزار نباید در استفاده از منابع سیستمی همچون مموری و پردازنده، ولخرجی داشته باشد.

4. قابلیت نگهداری (Maintainability)

نرم افزار باید به روشی نوشته شود که بتواند منجر به تغییرات مورد نیاز کاربران شود. این ویژگی بحرانی می باشد، چرا که تغییر نرم افزار نیاز غیر قابل امتناع از تغییر محیط بیزینس مورد نظر می باشد و باید بتوان با محیط جدید و تغییرات وفق پیدا کند.

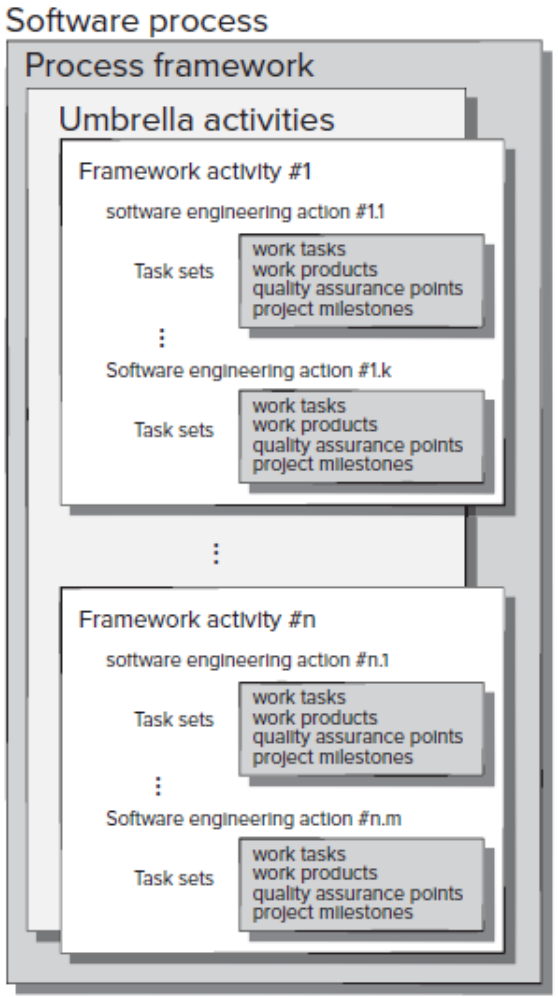
مهندسی نرم افزار

Software Engineering

۲ مدل های توسعه نرم افزار

Mahmoud Matinfar
matinfar.ir
@matinfar_ir
matinfar.ir@gmail.com

2. مدل فرآیند عمومی



فرایند نرم افزار چیست؟

فرآیند نرم‌افزار شامل مجموعه‌ای از فعالیت‌ها، اقدامات و وظایف است که برای ایجاد یک محصول نرم‌افزاری انجام می‌شود. هر یک از این فعالیت‌ها، اقدامات و وظایف در چارچوبی قرار می‌گیرند که رابطه‌شان با فرآیند و با یکدیگر را تعریف می‌کند.

فرآیند نرم‌افزار به صورت شماتیک در شکل روبرو نشان داده شده است. در این شکل، هر فعالیت چارچوبی توسط مجموعه‌ای از اقدامات مهندسی نرم‌افزار پر می‌شود. هر اقدام مهندسی نرم‌افزار نیز توسط یک مجموعه وظیفه تعریف می‌شود که شامل وظایف کاری، محصولات کاری، نقاط تضمین کیفیت و نقاط عطف پروژه است.

فرآیند چارچوب عمومی برای مهندسی نرم‌افزار پنج فعالیت چارچوبی اصلی را تعریف می‌کند: ارتباطات، برنامه‌ریزی، مدل‌سازی، ساخت و استقرار.

علاوه بر این، مجموعه‌ای از فعالیت‌های چتری مانند ردیابی و کنترل پروژه، مدیریت ریسک، تضمین کیفیت، مدیریت پیکربندی و بازبینی‌های فنی در طول فرآیند اعمال می‌شوند.

2. جریان فرآیند (Process Flow)

یکی از جنبه‌های مهم فرآیند نرم‌افزار که تاکنون مورد بحث قرار نگرفته است، جریان فرآیند است.

جریان فرآیند به چگونگی سازمان‌دهی فعالیت‌های چارچوبی و اقدامات و وظایف درون آن‌ها با توجه به توالی و زمان اشاره دارد.

همانگونه که بیان شد، 5 فعالیت اصلی در چارچوب فرآیند عمومی نرم افزار تعریف می شود که شامل: ارتباطات (Communication)، برنامه‌ریزی (Planning)، مدل‌سازی (Modeling)، ساخت (Construction)، استقرار (Deployment) می باشد.

چهار نوع جریان فرآیند وجود دارد:

جریان خطی (Linear): هر یک از پنج فعالیت چارچوبی به ترتیب اجرا می‌شوند، از ارتباطات شروع شده و با استقرار پایان می‌یابد.

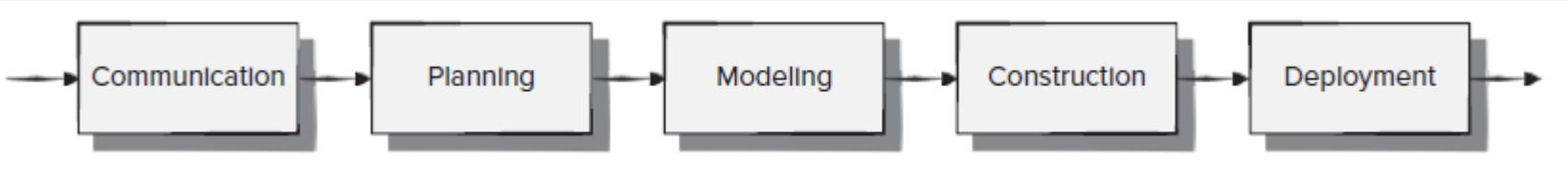
جریان تکرارشونده (Iterative): یک یا چند فعالیت قبل از رفتن به فعالیت بعدی تکرار می‌شوند.

جریان تکاملی (Evolutionary): فعالیت‌ها به صورت دایره‌ای اجرا می‌شوند و هر دور منجر به نسخه کامل‌تری از نرم‌افزار می‌شود.

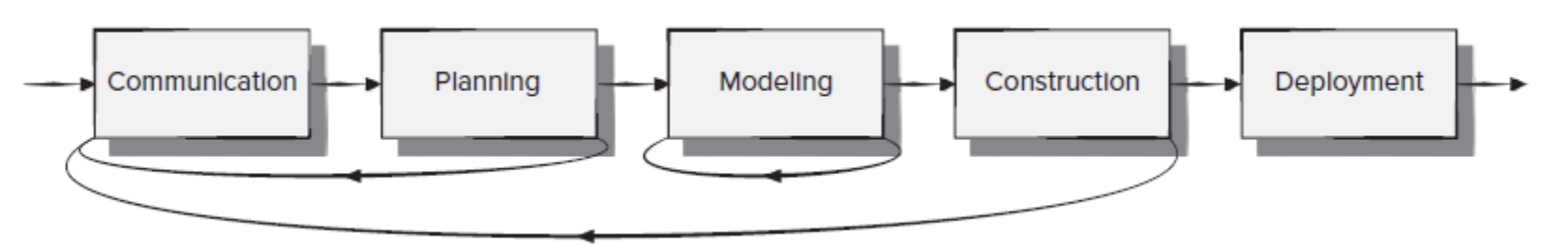
جریان موازی (Parallel): یک یا چند فعالیت به صورت موازی با سایر فعالیت‌ها اجرا می‌شوند.

2. جریان فرآیند (Process Flow)

جریان خطی (Linear): هر یک از پنج فعالیت چارچوبی به ترتیب اجرا می‌شوند، از ارتباطات شروع شده و با استقرار پایان می‌یابد.

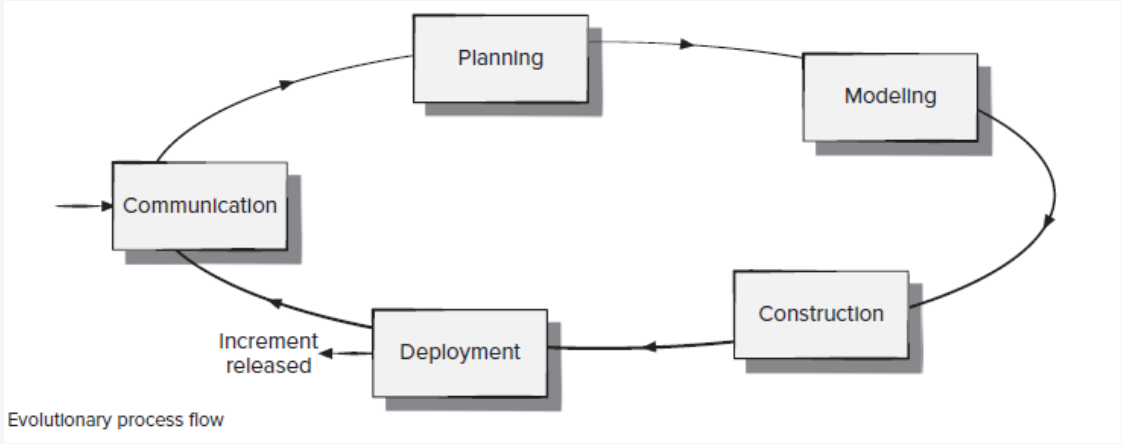


جریان تکرارشونده (Iterative): یک یا چند فعالیت قبل از رفتن به فعالیت بعدی تکرار می‌شوند.



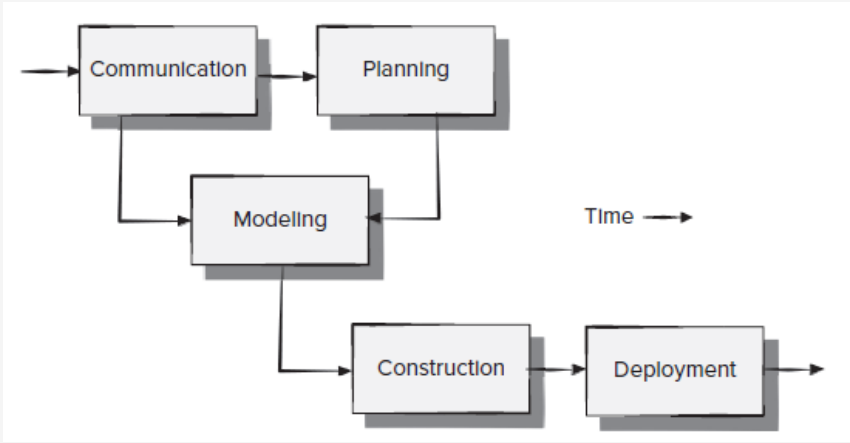
2. جریان فرآیند (Process Flow)

جریان تکاملی (Evolutionary): فعالیت‌ها به صورت دایره‌ای اجرا می‌شوند و هر دور منجر به نسخه کامل‌تری از نرم‌افزار می‌شود.



Evolutionary process flow

جریان موازی (Parallel): یک یا چند فعالیت به صورت موازی با سایر فعالیت‌ها اجرا می‌شوند.



2.2. تعریف فعالیت های چارچوب

اگرچه پنج فعالیت چارچوبی و تعریف اولیه هر یک در فصل 1 ارائه شده است، اما یک تیم نرم‌افزاری برای اجرای صحیح هر یک از این فعالیت‌ها به اطلاعات بیشتری نیاز دارد.

بنابراین، سوال کلیدی این است: چه اقداماتی برای یک فعالیت چارچوبی مناسب است؟

این اقدامات به ماهیت مسئله‌ای که باید حل شود، ویژگی‌های افراد انجام‌دهنده کار و ذینفعان پروژه بستگی دارد.

به عنوان مثال، برای یک پروژه کوچک با نیازمندی‌های ساده، فعالیت ارتباطات ممکن است تنها شامل یک تماس تلفنی یا ایمیل با ذینفع مربوطه باشد. در این حالت، تنها اقدام لازم یک مکالمه تلفنی است و مجموعه وظایف این اقدام شامل موارد زیر است: برقراری تماس با ذینفع، بحث در مورد نیازمندی‌ها و تهیه یادداشت، سازمان‌دهی یادداشت‌ها به صورت یک بیانیه کوتاه از نیازمندی‌ها و ارسال ایمیل به ذینفع برای بررسی و تأیید.

اما اگر پروژه بسیار پیچیده‌تر باشد و ذینفعان متعددی با نیازمندی‌های گاه متضاد وجود داشته باشند، فعالیت ارتباطات ممکن است شامل شش اقدام مجزا باشد: شروع، استخراج، بسط، مذاکره، مشخص‌سازی و اعتبارسنجی. هر یک از این اقدامات مهندسی نرم‌افزار ممکن است شامل وظایف کاری متعدد و در برخی موارد چندین محصول کاری مجزا باشد.

2.3. شناسایی مجموعه وظایف (Task Set)

مجموعه وظایف، کارهای واقعی را که برای دستیابی به اهداف یک اقدام مهندسی نرم‌افزار باید انجام شود، تعریف می‌کند.

به عنوان مثال، استخراج نیازمندی‌ها (که معمولاً به آن "جمع‌آوری نیازمندی‌ها" گفته می‌شود) یک اقدام مهم مهندسی نرم‌افزار است که در طول فعالیت ارتباطات انجام می‌شود. هدف از جمع‌آوری نیازمندی‌ها این است که بدانیم ذینفعان مختلف چه چیزی از نرم‌افزاری که قرار است ساخته شود، می‌خواهند.

برای یک پروژه کوچک و نسبتاً ساده، مجموعه وظایف جمع‌آوری نیازمندی‌ها ممکن است به این صورت باشد:

تهیه لیست ذینفعان پروژه؛ دعوت از همه ذینفعان به یک جلسه غیررسمی؛ درخواست از هر ذینفع برای تهیه لیستی از ویژگی‌ها و عملکردهای مورد نیاز؛ بحث در مورد نیازمندی‌ها و تهیه لیست نهایی؛ اولویت‌بندی نیازمندی‌ها/ یادداشت‌برداری از مناطق عدم قطعیت.

برای یک پروژه نرم‌افزاری بزرگ‌تر و پیچیده‌تر، مجموعه وظایف متفاوتی مورد نیاز است. این مجموعه ممکن است شامل وظایف زیر باشد:

تهیه لیست ذینفعان پروژه؛ مصاحبه جداگانه با هر ذینفع برای تعیین خواسته‌ها و نیازهای کلی؛ تهیه لیست اولیه از عملکردها و ویژگی‌ها بر اساس ورودی ذینفعان؛ برنامه‌ریزی برای یک سری جلسات مشخص‌سازی برنامه کاربردی؛ برگزاری جلسات؛ تولید سناریوهای کاربری غیررسمی به عنوان بخشی از هر جلسه؛ اصلاح سناریوهای کاربری بر اساس بازخورد ذینفعان؛ تهیه لیست اصلاح‌شده از نیازمندی‌های ذینفعان؛ استفاده از تکنیک‌های استقرار عملکرد کیفیت برای اولویت‌بندی نیازمندی‌ها؛ بسته‌بندی نیازمندی‌ها به گونه‌ای که بتوانند به صورت افزایشی تحویل داده شوند؛ یادداشت‌برداری از محدودیت‌ها و محدودیت‌هایی که بر سیستم اعمال می‌شوند؛ بحث در مورد روش‌های اعتبارسنجی سیستم.

هر دو مجموعه وظایف فوق به "جمع‌آوری نیازمندی‌ها" دست می‌یابند، اما از نظر عمق و سطح رسمیت بسیار متفاوت هستند. تیم نرم‌افزاری مجموعه وظایفی را انتخاب می‌کند که به آن‌ها امکان دستیابی به اهداف هر اقدام را می‌دهد و در عین حال کیفیت و چابکی را حفظ می‌کند.

2.3. مدل‌های فرآیندی تجویزی

مدل‌های فرآیندی تجویزی (Prescriptive Process Models) به مدل‌هایی گفته می‌شوند که مجموعه‌ای از عناصر فرآیندی و جریان کاری قابل پیش‌بینی را تعریف می‌کنند.

این مدل‌ها سعی می‌کنند با ایجاد ساختار و نظم در فرآیند توسعه نرم‌افزار، به تیم‌ها کمک کنند تا پروژه‌ها را به شیوه‌ای سیستماتیک و کنترل‌شده پیش ببرند. این مدل‌ها معمولاً فعالیت‌ها و وظایف را به صورت ترتیبی یا تکرارشونده سازمان‌دهی می‌کنند و دستورالعمل‌های مشخصی برای پیشرفت پروژه ارائه می‌دهند.

با این حال، سوالی که مطرح می‌شود این است که آیا این مدل‌ها برای دنیای نرم‌افزار که با تغییرات مداوم و نیازهای در حال تحول مواجه است، مناسب هستند؟ اگر مدل‌های سنتی (یا همان تجویزی) را کنار بگذاریم و به جای آن‌ها از مدل‌ها و روش های تطبیقی استفاده کنیم، نتیجه بهتری نخواهیم گرفت؟

2.3. مدل‌های فرآیندی تجویزی

مدل آبشاری (Waterfall)

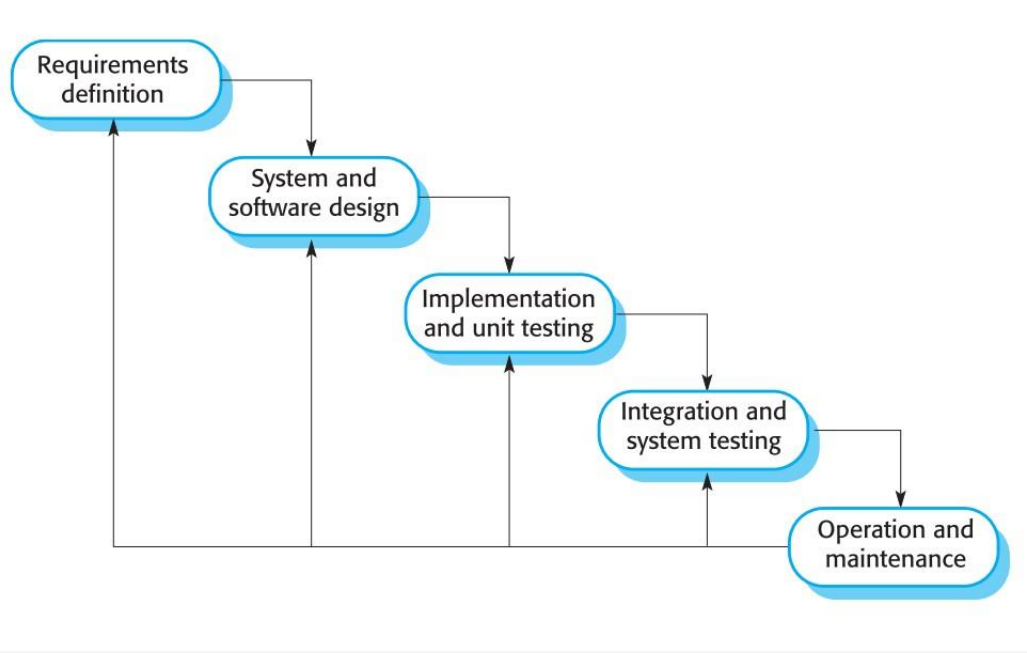
اولین مدل منتشر شده از فرآیند توسعه نرم افزار، از مدل های فرآیند مهندسی که در مهندسی سیستم های بزرگ جنگی/نظامی استفاده می شده اند، حاصل شده است. این مدل، فرایند توسعه نرم افزار را به عنوان تعدادی مرحله ارائه می دهد که به دلیل پی در پی بودن هر فاز با فاز دیگر، این مدل مدل آبشاری یا سیکل زمانی نرم افزار نامیده شده است.

مدل Waterfall که یکی از قدیمی ترین و سنتی ترین مدل های ساخت نرم افزار به حساب می آید، یک فرآیند برنامه محور است، مدلی منظم که برای همه فعالیت‌های قبل از شروع پروژه برنامه‌ریزی شده است. هر فعالیت در این مدل مشخص شده و نشان دهنده یک فاز جداگانه است. مدل آبشاری مثالی از یک فرایند برنامه ریزی شده (مبتنی بر برنامه) می باشد. در اصل، شما قبل از شروع توسعه نرم افزار باید تمامی فعالیت های فرآیند را برنامه ریزی کرده باشید.

مدل آبشاری، که گاهی اوقات به آن مدل خطی-ترتیبی نیز گفته می‌شود، یکی از قدیمی‌ترین پارادایم‌های مهندسی نرم‌افزار است. این مدل یک رویکرد سیستماتیک و ترتیبی را پیشنهاد می‌کند که با مشخص‌سازی نیازمندی‌ها شروع می‌شود و از طریق مراحل برنامه‌ریزی، مدل‌سازی، ساخت و استقرار پیش می‌رود و در نهایت به پشتیبانی از نرم‌افزار تکمیل‌شده ختم می‌شود.

2.3. مدل‌های فرآیندی تجویزی

گام های فرآیند مدل آبشاری



تعیین و تحلیل نیازمندی ها (Requirements analysis and definition)، خدمات، محدودیت ها و اهداف سیستم با مشورت کاربران سیستم معین می شود. آنها سپس به طور جزیی تعریف شده و به عنوان مشخصات سیستم به کار گرفته می شوند.

طراحی سیستم و نرم افزار (System and software design)، فرایند طراحی نرم افزار الزامات را به سخت افزار یا سیستم های نرم افزاری تخصیص می دهد و یک معماری کلی سیستم را ایجاد می کند. طراحی نرم افزار شامل شناسایی و توصیف انتزاعات اساسی سیستم نرم افزاری و روابط آنها می باشد.

پیاده سازی و تست واحد (Implementation and unit testing) در طی این مرحله، طراحی نرم افزار به عنوان مجموعه ای از برنامه ها یا واحد های برنامه تحقق می یابد. تست واحد شامل تاییدیه این است که هر واحد مشخصات خود را برآورده کند.

یکپارچگی و تست سیستم (Integration and system testing) واحد های برنامه فردی یا برنامه ها به عنوان یک سیستم کامل، یکپارچه و تست می شوند تا اطمینان حاصل شود که الزامات نرمافزار برآورده شده است. پس از تست، سیستم نرم افزاری به مشتری تحویل داده می شود.

بهره برداری و نگهداری (Operation and maintenance) به طور معمول طولانی ترین مرحله چرخه زندگی، این مرحله است. سیستم نصب شده و مورد استفاده عملی قرار می گیرد. نگهداری شامل تصحیح خطاهایی که در مراحل قبلی چرخه زندگی کشف نشده اند می پردازد و عملکرد واحد های سیستم را بهبود می بخشد و خدمات سیستم را به عنوان الزامات تازه کشف شده، ارتقا می دهد.

2.3. مدل‌های فرآیندی تجویزی

مدل نمونه سازی اولیه (Prototype)

در بسیاری از موارد، مشتری اهداف کلی را برای نرم‌افزار تعریف می‌کند، اما نیازمندی‌های دقیق را مشخص نمی‌کند. در چنین شرایطی، مدل نمونه‌سازی اولیه می‌تواند بهترین رویکرد باشد. این مدل به ذینفعان کمک می‌کند تا بهتر درک کنند چه چیزی باید ساخته شود.

در مدل نمونه سازی اولیه، یک نمونه اولیه ایجاد می شود، آزمایش می شود و سپس در صورت لزوم بازسازی می شود تا زمانی که نتیجه مناسبی برای توسعه سیستم یا محصول کامل حاصل شود. سپس بر اساس الزامات، طرح ایجاد می شود و نمونه اولیه برای یک طرح خاص مدل شده و به کاربران تحویل داده می شود. سپس بر اساس بازخورد کاربر، تغییرات مناسب اعمال می شود.

این مدل بر پایه جمع آوری الزامات کار می‌کند که در آن توسعه‌دهنده و کاربر با یکدیگر مشارکت داشته و هدف نرم‌افزار و نیازهای آن را مشخص می‌کنند. با این کار یک طراحی سریع ایجاد می‌شود و بر جنبه‌های قابل مشاهده نرم‌افزار برای کاربر تمرکز می‌کند.

این مدل هنگامی که مشتری شما خواسته ای مشروع دارد، ولی جزئیات چندانی در اختیار شما قرار نداده است یا وقتی سازنده از بازدهی یک الگوریتم، قابلیت تطابق با یک سیستم عامل خاص مطمئن نباشد، کاربرد دارد.

2.5. مدل‌های فرآیندی تجویزی

مراحل مدل نمونه سازی اولیه

ارتباطات: تعیین اهداف کلی و نیازمندی‌های شناخته‌شده.

طراحی سریع: ایجاد یک طراحی اولیه از بخش‌هایی از نرم‌افزار که برای کاربر نهایی قابل مشاهده است.

ساخت نمونه اولیه: ساخت یک نمونه اولیه بر اساس طراحی سریع.

ارزیابی نمونه اولیه: دریافت بازخورد از ذینفعان و اصلاح نیازمندی‌ها.

تکرار: تکرار مراحل تا زمانی که نمونه اولیه نیازمندی‌ها را برآورده کند.

2.3. مدل‌های فرآیندی تجویزی

مدل مارپیچی یا حلزونی (spiral)

مدل مارپیچی یک مدل فرآیندی تکاملی است که توسط بری بوهم پیشنهاد شده است. این مدل ویژگی‌های تکرارشونده نمونه‌سازی اولیه را با جنبه های کنترلی و سیستماتیک مدل آبشاری ترکیب می‌کند. مدل مارپیچی به تیم‌ها اجازه می‌دهد تا نسخه‌های تکاملی از نرم‌افزار را به سرعت توسعه دهند.

این مدل یک فرآیند ریسک محور است و به صورت تکرارشونده روند توسعه را به شکل یک حلقه ارائه می‌دهد. بر خلاف مدل‌های دیگر، مراحل آن به مشکلی که بیشترین خطر خرابی را دارد، می‌پردازند. با استفاده از مدل مارپیچی، نرم افزار به صورت یک سری نگارش های تکاملی توسعه می یابد.

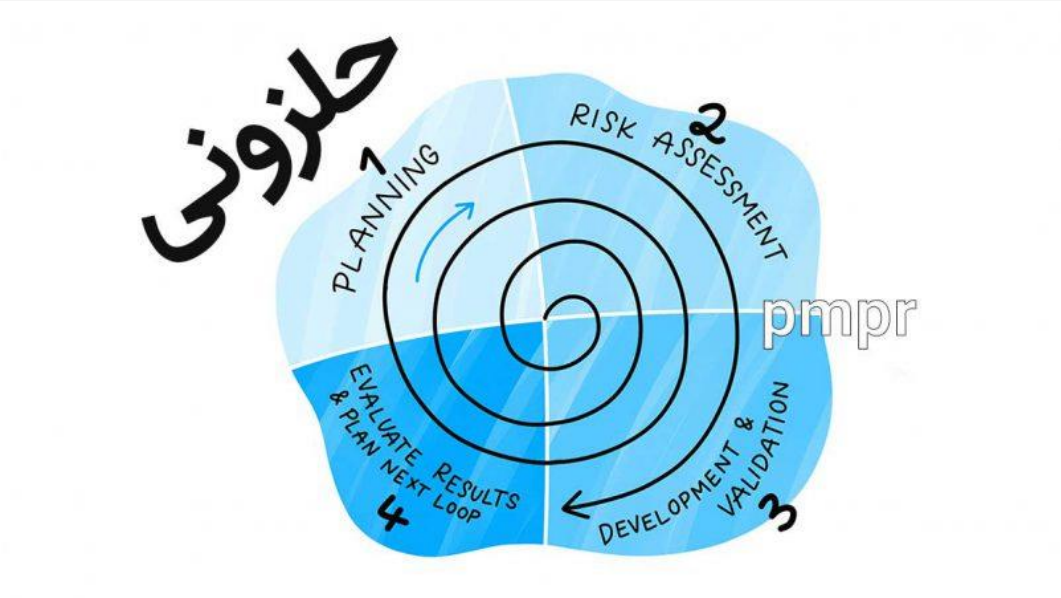
مدل حلزونی برای تیم های ریسک گریز که روی پروژه های بزرگ کار می‌کنند مناسب است.
مدل حلزونی در تئوری خارق العاده است. ولی چنین رویکر حساب شده‌ای به دلیل سربار زمان و هزینه های زیادش به ندرت در عمل استفاده میشود. لذا برای بیشتر افراد و تیم ها مناسب نیست.

2.5. مدل‌های فرآیندی تجویزی

فازهای مدل مارپیچی

مدل فرایند توسعه نرم افزار مارپیچی یا حلزونی دارای فازهای زیر است:

- **برنامه ریزی و تعیین اهداف (Planning)**: شناسایی اهداف هر دور از مارپیچ و برنامه ریزی
- **ارزیابی و تحلیل ریسک (Risk Assessment)**: ارزیابی ریسک‌های فنی و مدیریتی
- **توسعه و اعتبار سنجی (Development and Validation)**: ساخت و تست نسخه‌ای از نرم‌افزار.
- **برنامه ریزی حلقه بعدی (Plan Next Loop)**: برنامه‌ریزی برای دور بعدی مارپیچ.



2.3. مدل‌های فرآیندی تجویزی

مدل فرآیند یکپارچه (Unified Process – UP)

مدل فرآیند یکپارچه (UP) تلاش می‌کند بهترین ویژگی‌های مدل‌های سنتی را با اصول توسعه چابک ترکیب کند. این مدل بر ارتباط با مشتری، معماری نرم‌افزار و تحویل تدریجی تأکید دارد. UP از زبان مدل‌سازی یکپارچه (UML) برای توصیف نیازمندی‌ها و طراحی سیستم استفاده می‌کند.

در این فرایند مهندسی نرم‌افزار از رویکرد منظمی برای تعیین وظایف و مسئولیت‌ها در یک سازمان توسعه استفاده می‌شود. هدف آن اطمینان از تولید نرم‌فزار با کیفیت بالا و برآورده ساختن نیازهای کاربران انتهایی آن‌ها است.

فرآیند یکپارچه منطقی (Rational Unified Process) یا RUP یك فرآیند مهندسی نرم افزار خوش ساختار و خوش تعریف می باشد و به وضوح معین می کند چه کسی مسئول چه کاری می باشد و چگونه و در چه زمانی باید مسئولیت خود را انجام دهد.

RUP یک محصول فرایندی است که با نرم‌افزار ®Rational توسعه یافته و پشتیبانی می‌شود و می‌توان آن را یک راهنمای نهایی برای واگذاری مسئولیت‌ها و وظایف در یک سازمان توسعه‌یافته دانست.

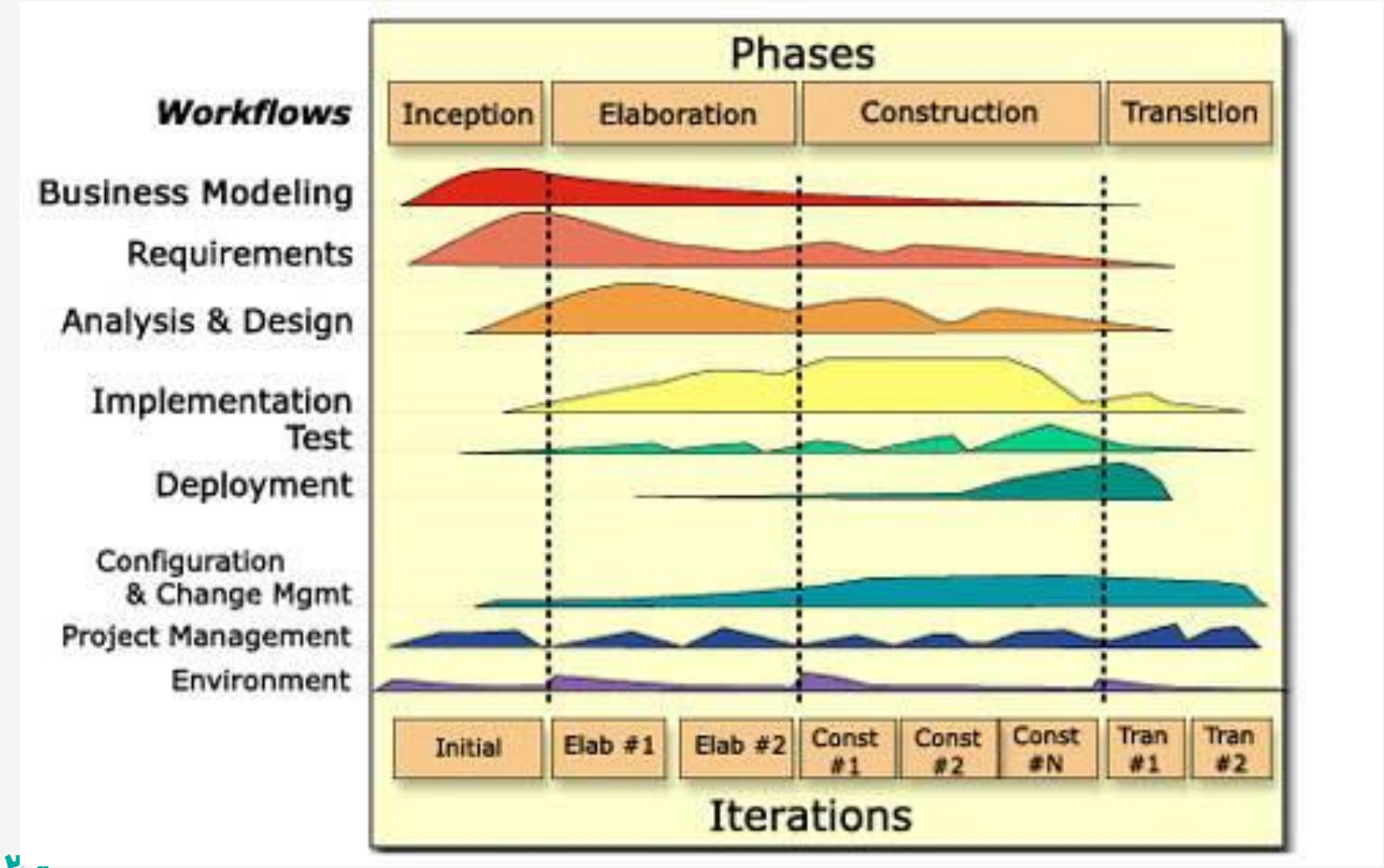
به کمك RUP، مراحل مختلف و مهم پروژه و نکات اصلی آن به سرعت شناخته می شوند.

RUP شامل فرآیندهای گوناگونی است و بر اساس آن می توان تیم های کوچك و بزرگ را توسعه داد.

2.3. مدل‌های فرآیندی تجویزی

معماری کلی RUP

معماری کلی RUP دارای 4 فاز و 9 دیسیپلین می باشد که در یک ساختار دو بعدی قرار گرفته است.



2.5. مدل‌های فرآیندی تجویزی

فازهای RUP

نوار افقی نشان دهنده فازها و مراحل مهم پروژه می باشد.

فاز شروع (Inception): شناخت حوزه و محدوده پروژه، شناسایی ذینفعان پروژه، توافق در مورد ساخت پروژه. هدف ساخت مدل تجاری

فاز شناخت (Elaboration): ساخت معماری پایه سیستم و فهم نحوه ساخت سیستم، شناخت نیازمندیها برای رسیدن به یک معماری مناسب برای توسعه نرم افزار

فاز ساخت (Construction): ساخت اولین نسخه عملیاتی، انجام حجم زیادی از تولید

فاز انتقال (Transition): ساخت نسخه نهایی و انتقال به مشتریان، یعنی انتقال محصول نرم افزاری از محیط توسعه سازمان یا شرکت به سمت مشتری

2.5. مدل‌های فرآیندی تجویزی

دیسیپلین ها یا جریان های کاری RUP

محور عمودی نشان دهنده ساختار ایستای پروژه و جریان های کاری و فعالیت های پروژه است.

1. مدل سازی تجاری (Business Modeling)
2. نیازمندی ها (Requirements)
3. تحلیل و طراحی (Analysis and Design)
4. پیاده سازی (Implementation)
5. استقرار (Deployment)
6. تست (Test)
7. مدیریت تغییرات و تنظیمات (Configuration and change management)
8. مدیریت پروژه (Project Management)
9. محیط (Environment)

مهندسی نرم افزار

Software Engineering

۳ چابکی و فرآیند

Mahmoud Matinfar
matinfar.ir
@matinfar_ir
matinfar.ir@gmail.com

مهندسی نرم افزار

Software Engineering

۴ مدل فرآیند پیشنهادی نرم‌افزار

Mahmoud Matinfar
matinfar.ir
@matinfar_ir
matinfar.ir@gmail.com

مدل فرآیند پیشنهادی نرم افزار

1. تعریف نیازمندی ها

هر پروژه نرم‌افزاری با درک مشکل و تعیین خروجی‌های مهم برای ذینفعان شروع می‌شود. این فرآیند مهندسی نیازمندی‌ها نامیده می‌شود و شامل درک نیازهای تجاری و مسائل فنی است. تیم‌هایی که زمان کافی را برای این کار صرف نمی‌کنند، ممکن است با مشکلاتی مانند بازکاری پرهزینه، افزایش هزینه‌ها، کیفیت پایین محصول، تأخیر در تحویل، نارضایتی مشتریان و روحیه پایین تیم مواجه شوند.

2. طراحی معماری مقدماتی

تصمیمات لازم برای توسعه یک طراحی معماری قوی باید در مراحل اولیه انجام شود. این طراحی باید به گونه‌ای باشد که بتواند به راحتی تغییرات را در طول فرآیند توسعه بپذیرد.

3. برآورد منابع

یکی از جنبه‌های چالش‌برانگیز استفاده از پروتوتایپ‌سازی چرخشی یا چابک، برآورد زمان لازم برای تکمیل پروژه است. برآوردها باید به طور مداوم به‌روزرسانی شوند تا تغییرات در نیازمندی‌ها و زمان‌بندی پروژه را منعکس کنند.

مدل فرآیند پیشنهادی نرم افزار

4. ساخت اولین نمونه اولیه

توسعه پروتوتایپ اول شامل شناسایی ویژگی‌ها و عملکردهای مهم برای ذینفعان و ایجاد یک پروتوتایپ کاغذی برای بررسی اولیه است. این پروتوتایپ‌ها به ذینفعان کمک می‌کنند تا ایده‌های خود را به وضوح بیان کنند و تغییرات لازم را شناسایی کنند.

5. ارزیابی نمونه اولیه

تست پروتوتایپ در حین ساخت آن انجام می‌شود و شامل جمع‌آوری بازخورد از ذینفعان است. این ارزیابی به تیم کمک می‌کند تا تصمیم بگیرد که آیا ادامه توسعه لازم است یا خیر.

6. تصمیم ادامه یا توقف

پس از ارزیابی پروتوتایپ، ذینفعان تصمیم می‌گیرند که آیا توسعه نرم‌افزار ادامه یابد یا خیر. این تصمیم بر اساس کیفیت پروتوتایپ و ارزیابی ریسک‌ها اتخاذ می‌شود.

مدل فرآیند پیشنهادی نرم افزار

7. تکامل نمونه اولیه

پس از ارزیابی پروتوتایپ، تیم باید به جمع‌آوری بازخورد و برنامه‌ریزی برای ساخت پروتوتایپ جدید بپردازد. این فرآیند شامل تعیین دامنه جدید پروتوتایپ و ارزیابی ریسک‌های مرتبط با ادامه توسعه است.

8. انتشار نمونه اولیه

پروتوتایپ‌های تکاملی باید تحت آزمایش پذیرش کاربر قرار گیرند تا اطمینان حاصل شود که نرم‌افزار به درستی عمل می‌کند و نیازهای کاربران را برآورده می‌سازد.

9. نگهداری نرم افزار

نگهداری شامل فعالیت‌هایی است که برای حفظ نرم‌افزار پس از تحویل آن به کاربران نهایی انجام می‌شود. این فعالیت‌ها شامل اصلاحات، به‌روزرسانی‌ها و بهبودهای نرم‌افزار است.

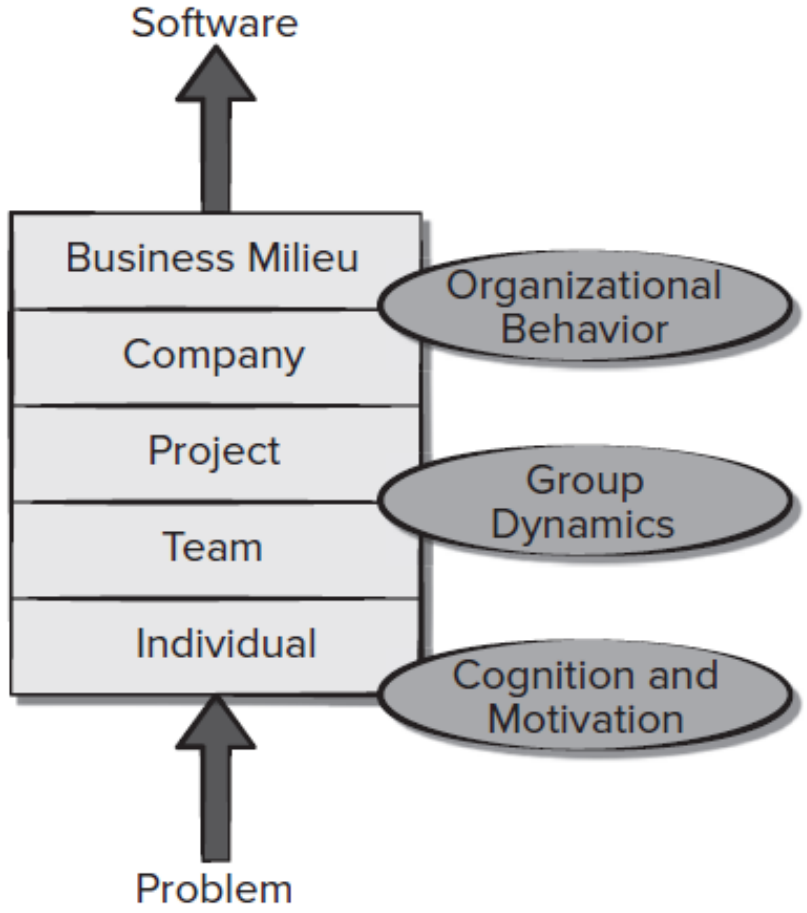
مهندسی نرم افزار

Software Engineering

۵ جنبه‌های انسانی مهندسی نرم‌افزار

Mahmoud Matinfar
matinfar.ir
@matinfar_ir
matinfar.ir@gmail.com

5.2. روانشناسی مهندسی نرم افزار



در یک مقاله بنیادی در مورد روانشناسی مهندسی نرم‌افزار، بیل کورتس و دایان والز مدل رفتاری لایه‌ای برای توسعه نرم‌افزار پیشنهاد می‌کنند. این مدل به سه سطح اصلی تقسیم می‌شود:

سطح فردی: در این سطح، روانشناسی مهندسی نرم‌افزار بر شناسایی مشکل، مهارت‌های حل مسئله و انگیزه برای تکمیل راه‌حل در چارچوب‌های تعیین‌شده تمرکز دارد. مهندسان نرم‌افزار باید توانایی شناسایی مشکلات و استفاده از مهارت‌های لازم برای حل آن‌ها را داشته باشند. همچنین، انگیزه و تعهد به انجام کار در زمان مقرر از اهمیت بالایی برخوردار است.

سطح تیم و پروژه: در این سطح، دینامیک گروهی به عنوان عامل غالب در موفقیت پروژه مطرح می‌شود. ساختار تیم و عوامل اجتماعی نقش مهمی در موفقیت دارند. ارتباطات گروهی، همکاری و هماهنگی به اندازه مهارت‌های فردی اعضای تیم اهمیت دارند. این سطح به تعاملات بین اعضای تیم و چگونگی کارکرد آن‌ها در کنار یکدیگر می‌پردازد.

لایه‌های بیرونی: در این لایه، رفتار سازمانی بر اقدامات شرکت و پاسخ آن به محیط کسب‌وکار تأثیر می‌گذارد. این لایه شامل عواملی است که بر روی تصمیم‌گیری‌ها و عملکرد کلی تیم تأثیر می‌گذارد.

5.3. تیم نرم افزاری

تیم ژله‌شده (Jelled Team): تیمی که اعضای آن به‌شدت به هم پیوسته‌اند و عملکردشان از مجموع عملکرد فردی‌شان بهتر است.

ویژگی‌های تیم‌های مؤثر:

احساس هدف: داشتن هدف مشترک و تعریف واضح از موفقیت

احساس مشارکت: ارزش‌گذاری به مهارت‌ها و مشارکت هر عضو.

اعتماد: اعتماد به مهارت‌ها و شایستگی‌های هم‌تیمی‌ها.

احساس بهبود: بازنگری و بهبود مستمر.

سموم تیمی (Team Toxicity): عواملی مانند جو کاری شلوغ، ناامیدی، فرآیندهای ناکارآمد، تعریف نشدن نقش‌ها و مواجهه مداوم با شکست می‌توانند تیم را مسموم کنند.

5.5. ساختار تیم

ساختار تیم نرم‌افزاری به عوامل مختلفی بستگی دارد، از جمله سبک مدیریت سازمان، تعداد اعضای تیم و سطح مهارت‌های آن‌ها، و همچنین دشواری کلی مشکل. در این بخش، به بررسی عواملی که باید در برنامه‌ریزی ساختار تیم‌های مهندسی نرم‌افزار در نظر گرفته شوند، پرداخته می‌شود:

عوامل مؤثر در ساختار تیم

دشواری مشکل: پیچیدگی و چالش‌های فنی که تیم با آن مواجه است، تأثیر زیادی بر ساختار تیم دارد. مشکلات پیچیده‌تر ممکن است نیاز به تخصص‌های مختلف و همکاری نزدیک‌تری داشته باشند.

اندازه برنامه: اندازه نهایی برنامه‌ها، که می‌تواند به صورت خطوط کد یا نقاط عملکرد اندازه‌گیری شود، بر روی نحوه تقسیم کار و ساختار تیم تأثیر می‌گذارد.

مدت زمان همکاری تیم: طول مدت زمانی که تیم قرار است با هم کار کند، می‌تواند بر روی نحوه سازماندهی و ساختار تیم تأثیر بگذارد. تیم‌هایی که برای مدت طولانی‌تری با هم کار می‌کنند، ممکن است نیاز به ساختارهای پیچیده‌تری داشته باشند.

قابلیت مدولار کردن مشکل: اگر مشکل قابل تقسیم به بخش‌های کوچکتر باشد، تیم می‌تواند به راحتی وظایف را بین اعضا تقسیم کند.

کیفیت و قابلیت اطمینان مورد نیاز: نیاز به کیفیت و قابلیت اطمینان بالاتر ممکن است نیاز به نظارت و کنترل بیشتری در ساختار تیم داشته باشد.

سختی تاریخ تحویل: تاریخ‌های تحویل سخت و غیرقابل تغییر می‌تواند فشار بیشتری به تیم وارد کند و نیاز به ساختارهای مدیریتی قوی‌تر داشته باشد.

درجه اجتماعی بودن (ارتباطات): پروژه‌هایی که نیاز به ارتباطات و همکاری بیشتری دارند، ممکن است نیاز به ساختار تیمی متفاوتی داشته باشند.

5.5. تاثیر رسانه های اجتماعی

مقدمه

ایمیل، پیام‌رسانی متنی و جلسات ویدیویی به ابزارهای رایجی در کار مهندسی نرم‌افزار تبدیل شده‌اند. با این حال، این ابزارها صرفاً جایگزین‌هایی مدرن برای ارتباطات چهره‌به‌چهره هستند.

رسانه‌های اجتماعی (Social Media) چیزی فراتر از این ابزارها هستند و نقش متفاوتی در ارتباطات و همکاری تیم‌های نرم‌افزاری ایفا می‌کنند. رسانه‌های اجتماعی فراتر از ابزارهای ارتباطی سنتی هستند و می‌توانند به بهبود همکاری و ارتباطات در تیم‌های نرم‌افزاری کمک کنند. این ابزارها به ویژه برای تیم‌های بزرگ و پراکنده جغرافیایی مفید هستند.

امنیت و حریم خصوصی باید در استفاده از رسانه‌های اجتماعی مورد توجه قرار گیرد تا از افشای اطلاعات محرمانه جلوگیری شود.

نقش رسانه‌های اجتماعی در مهندسی نرم‌افزار

فرآیندهای اجتماعی مرتبط با توسعه نرم‌افزار به توانایی مهندسان در یافتن و ارتباط با افرادی که اهداف مشترک و مهارت‌های مکمل دارند، وابسته است.

رسانه‌های اجتماعی به مهندسان کمک می‌کنند تا:

- با افراد هم‌فکر و دارای مهارت‌های مکمل ارتباط برقرار کنند.
- ترجیحات ارتباطی و تیمی هر عضو را هماهنگ کنند.
- در طول چرخه حیات نرم‌افزار همکاری و هماهنگی داشته باشند.
- از موفقیت محصول خود در بازار حمایت کنند.

5.5. تاثیر رسانه های اجتماعی

ویژگی‌های رسانه‌های اجتماعی در مهندسی نرم‌افزار

ارتباطات بهبود یافته: رسانه‌های اجتماعی مانند فیس‌بوک، لینکدین، اسلک و توییتر به مهندسان نرم‌افزار این امکان را می‌دهند که با یکدیگر و با ذینفعان پروژه ارتباط برقرار کنند. این ارتباطات می‌تواند به تسهیل همکاری و هماهنگی در طول چرخه حیات نرم‌افزار کمک کند.

ایجاد شبکه‌های اجتماعی: این ابزارها به مهندسان نرم‌افزار اجازه می‌دهند تا با افرادی که اهداف مشابه و مهارت‌های مکمل دارند، ارتباط برقرار کنند. این شبکه‌ها می‌توانند به اشتراک‌گذاری دانش و تجربیات کمک کنند و به حل مشکلات پیچیده‌تر یاری رسانند.

فراهم کردن بازخورد سریع: رسانه‌های اجتماعی امکان دریافت بازخورد سریع و مکرر از ذینفعان و اعضای تیم را فراهم می‌کنند. این بازخوردها می‌توانند به بهبود کیفیت محصول و تطابق بهتر با نیازهای مشتریان کمک کنند.

تسهیل همکاری در تیم‌های جهانی: با توجه به اینکه تیم‌های جهانی معمولاً با چالش‌های ارتباطی و فرهنگی مواجه هستند، رسانه‌های اجتماعی می‌توانند به کاهش این چالش‌ها کمک کنند و ارتباطات را تسهیل کنند.

چالش‌های استفاده از رسانه‌های اجتماعی

محرمانگی اطلاعات: بسیاری از کارهای مهندسی نرم‌افزار ممکن است محرمانه باشند و افشای آن‌ها می‌تواند به سازمان آسیب برساند.

موازنه بین فواید و خطرات: باید بین مزایای استفاده از رسانه‌های اجتماعی و خطرات ناشی از افشای اطلاعات محرمانه تعادل برقرار کرد.

**

5.6. تیم های جهانی

در حوزه نرم‌افزار، جهانی‌سازی به معنای انتقال کالاها و خدمات فراتر از مرزهای ملی نیست. در چند دهه گذشته، تعداد فزاینده‌ای از محصولات نرم‌افزاری بزرگ توسط تیم‌های نرم‌افزاری که معمولاً در کشورهای مختلف مستقر هستند، ساخته شده‌اند. این تیم‌های توسعه نرم‌افزار جهانی Global Software Development (GSD) با چالش‌های منحصر به فردی مواجه هستند که شامل هماهنگی، همکاری، ارتباطات و تصمیم‌گیری‌های تخصصی می‌شود.

چالش‌های تیم‌های GSD

تیم‌های GSDبا چالش‌های خاصی در زمینه هماهنگی، همکاری و ارتباطات روبرو هستند که می‌تواند تأثیر عمیقی بر تصمیم‌گیری‌ها داشته باشد. این چالش‌ها شامل چهار عامل زیر است:

پیچیدگی مشکل: مشکلات نرم‌افزاری معمولاً پیچیده هستند و نیاز به درک عمیق و تخصص‌های مختلف دارند.

عدم قطعیت و ریسک مرتبط با تصمیم‌گیری: تصمیم‌گیری در شرایط عدم قطعیت می‌تواند به نتایج غیرمنتظره منجر شود.

قانون عواقب ناخواسته: تصمیمات مرتبط با کار ممکن است تأثیرات ناخواسته‌ای بر دیگر اهداف پروژه داشته باشند.

دیدگاه‌های مختلف نسبت به مشکل: اعضای تیم ممکن است دیدگاه‌های متفاوتی نسبت به یک مشکل داشته باشند که می‌تواند به نتایج مختلفی منجر شود.

مهندسی نرم افزار

Software Engineering

۶ توسعه نرم افزار چابک

Mahmoud Matinfar
matinfar.ir
@matinfar_ir
matinfar.ir@gmail.com

مفاهیم چابک

Agile

ارائه چابک

در دهه 1990، توسعه نرم افزار با بحران روبرو شد که با عنوان "بحران توسعه برنامه" یا "تأخیر تحویل برنامه" شناخته می‌شود. در این برهه تیم ها قادر به پاسخگویی سریع به خواسته‌ها و نیازهای مشتری نبود و زمان تخمینی بین نیاز تجاری و تحویل برنامه واقعی حدود سه سال به طول می‌انجامید

بهتر است بدانید که مدل‌های توسعه سنتی بر اساس یک رویه زمانی بنا شده بودند، در این وضعیت توسعه به ترتیب اتفاق می‌افتاد و محصول نهایی تا آخرین مرحله برای مشتریان آشکار نمی‌شد.

در نتیجه طولانی شدن فرآیند تحویل، محصول نهائی یا فناوری قبل و بعد تفاوت زیادی بود و یا نیازهای مشتری در این مدت تغییر کرده بود؛

از طرفی در این دوره، شرکتها خصوصاً شرکتهای تولید کننده نرم افزار، به شدت بی پول شده بودند، لذا به دنبال کاهش هزینه های خود بودند؛

بسیاری از پروژه های نرم افزاری در آن زمان، ماهیت R& Dداشت و از ابتدا مشخص نبود که سودده میشود؛

شکست های مداوم پروژه ها منجر به نا امیدی رهبران صنعت توسعه نرم افزار شده بود.

در آن دوره، سیستم های مدیریت پروژه عموما با شیوه Waterfallپیش می رفت تا اینکه...

تعریف چابک

واژه چابك در فرهنگ لغت، به معنای حرکت سریع، چالاك است.

چابک یک رویکرد برای مدیریت پروژه بر پایه یک رویکرد افزایشی و تکراری است که روی تقسیم و تجزیه پروژه های بزرگ به وظایف قابل کنترل و کوچکتر تمرکز دارد و هدف اصلی آن، دستیابی به یک هدف طی چند مرحله کوچک است.

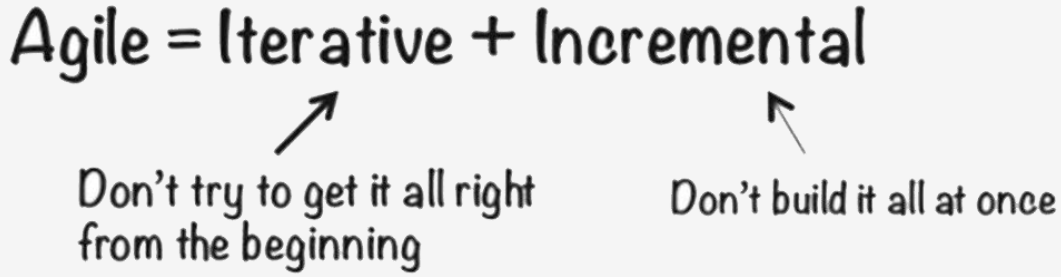
پروژه هایی که با تکرارهای کوتاه در طول چرخه عمر پروژه تکمیل میشوند.

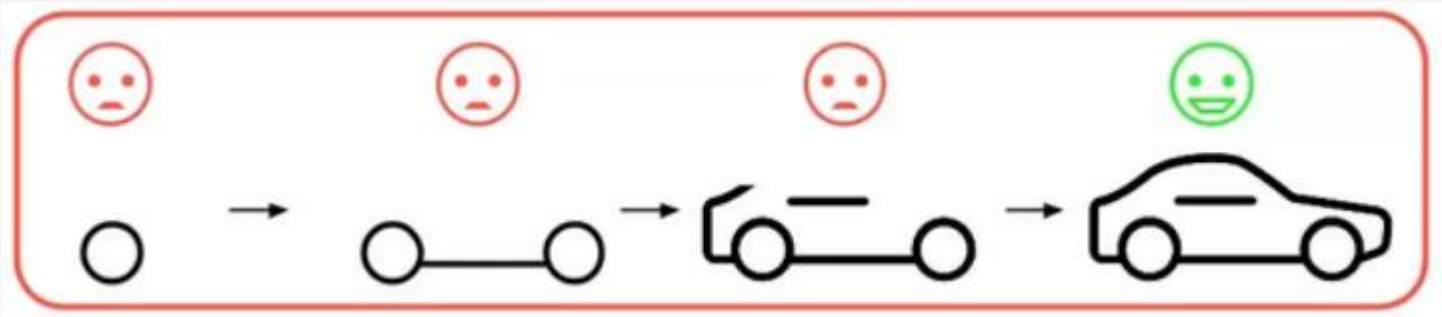
این رویکرد به تیمها اجازه میدهد با تمرکز بر اهداف کلی و با تعامل مستمر با مشتریان، بهبودهای مداومی در فرآیند کار و تحویل محصول داشته باشند و در نتیجه ضمن ارائه سریعتر، محصولی مطابق با نیاز مشتری و بازار عرضه کنند.

این روش مدیریت پروژه متمرکز بر تعامل، همکاری تیمی و انعطاف پذیری و بر مبنای فلسفه رضایت مشتری، تحویل نرم‌افزار به صورت افزایشی، تیم‌های نرم‌افزاری خود سازمانده (متشکل از مهندسین نرم‌افزار و ذینفعان) و نرم افزار کارکننده بنا شده است. مدیریت پروژه چابک در ابتدا برای شرکتهای نرم افزاری استفاده شد، سپس توسط صنایع مختلف، از خدمات مالی گرفته تا حمل و نقل، مورد استفاده قرار گرفته است

در طول 10 سال گذشته، استفاده از روشهای مدیریت پروژه چابک رشد فراوانی کرده و طی 3 سال اخیر این رشد سرعت یافته است.

چابک رویکردی تکراری و افزایشی





در مدل افزایشی، شما کل راه حل را به صورت قسمت‌هایی می سازید، اما در پایان هر بخش، چیزی برای بررسی یا بازخورد ندارید. برای تحویل محصول نهایی باید تا مرحله نهایی فرآیند افزایشی صبر کنید.

در مدل تکراری، شما ایده را تکرار می‌کنید و با تکرار روی نسخه‌ها، آن را بهبود می‌بخشید. هنگامی که از یک نسخه به نسخه دیگر منتقل می شوید، تصمیم می گیرید (بر اساس بازخورد) به چه چیزی در نسخه جدید به عنوان انتخاب بهتر نیاز دارید و چه چیزی را باید دور بیندازید.

چرا چابک؟

به عبارتی چه شد که این 17 نفر (بعلاوه چند ده نفر دیگر که به صورت غیر مستقیم در این بیانیه تاثیر داشتند) در سال 2001 اقدام به انتشار چنین روشی کردند!؟

جواب کاملا واضح است: ضعف های موجود در روش های سنتی باعث معرفی Agile شد.

روش های سنتی دارای ایراداتی بودند که از آن جمله می توان به موارد ذیل اشاره کرد:

- صرف زمان زیاد و در واقع اتلاف زمان برای طراحی Up-front در فاز اولیه پروژه
- مشخص نبودن و واضح نبودن نیازمندی ها بدلیل ارتباطات کاغذی به جای ارتباطات چهره به چهره
- بالا بودن هزینه تغییرات
- به طول انجامیدن پروژه و گذر از زمان تعیین شده در حد بسیار زیاد در بسیاری از پروژه ها
- عدم وجود زمان برای آزمایش محصول و متعاقبا محصولات پر از باگ و بدون کیفیت
- عدم شفافیت در پروسه توسعه و تولید
- داشتن ریسک بالا

بیانیه چابک

ارزش ها و اصول Agile در سال 2001 توسط 17 نفر از اساتید معتبر جهانی صنعت توسعه نرم افزار طی یک بیانیه با عنوان "**بیانیه چابک**" یا "**مانیفست چابک**" تنظیم و ارائه گردید.

ما با توسعه نرم افزار و کمک به دیگران در انجام آن، در حال کشف راه های بهتری برای توسعه نرم افزار هستیم.

از این طریق باید دست یابیم به ارزش:

افراد و تعاملات بالاتر از فرآیندها و ابزارها

نرم افزار کارکننده بالاتر از مستندات جامع

مشارکت مشتری در انجام کار بالاتر از قرارداد کار

پاسخگویی به تغییرات بالاتر از پیروی یک طرح

با وجود اینکه موارد سمت چپ نیز ارزشمند هستند ولی ما برای موارد سمت راست ارزش بیشتری قائل هستیم.

ارزش های چابک

افراد و تعاملات بالاتر از فرآیندها و ابزارها

افرادی که پشت این روند و فرآیندها هستند اهمیت و تاثیر بیشتری دارند.

نرم افزار کارکننده بالاتر از مستندات جامع

در گذشته توسعه دهندگان نرم افزار سالها برای تهیه مستندات دقیق وقت صرف می کردند و همه این اتفاقات قبل از اینکه حتی یک خط از برنامه نویسی خود را بنویسند اتفاق می افتاد. اگرچه اسناد و مدارك ضرر ی ندارد، اما در نهایت تیم ها باید بر روند کار تمرکز کنند و نرم افزارهای با کیفیت و عملکردی را در اختیار مشتریان قرار دهند.

مشارکت مشتری در انجام کار بالاتر از قرارداد کار

در گذشته مهمترین جنبه یک پروژه قرارداد بود. شما می توانید قرارداد دقیق خود را با مشتری خود تنظیم کنید و جزئیات دقیق محصول نهایی را ارائه دهید. به جای استفاده از این روش منسوخ ، تمرکز باید بر روی توسعه مداوم محصول یا برنامه موردنظر باشد. به همین دلیل این رویکرد بهتراست که با همکاری مشتری یا ذینفعان پروژه محصول نهایی مورد نیاز را طراحی کنیم.

پاسخگویی به تغییرات بالاتر از پیروی یک طرح

برای توسعه نرم افزار همواره ممکن است تغییرات مفیدی از سوی ذی نفعان و مشتریان مورد نیاز باشد.تیم های نرم افزاری در رویکرد اجایل باید همیشه توانایی تغییر جهت کار خود در هرزمان را داشته باشند. بیانیه چابک بر اهمیت مشتری مداری در این اصل تأکید می کند. تیم چابک باید بعد از تحویل محصول نهایی به مشتری ، انتظار برخی تغییرات و بازبینی ها را داشته باشد و از آنها برای بهبود کیفیت استفاده کند.

اصول چابک

اصل اول-رضایت مشتری

✓ بالاترین اولویت ما جلب رضایت مشتری با تحویل زود و مداوم نرم افزاری ارزشمند می‌باشد.

اصل دوم-گشاده رویی در مقابل تغییرات

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

اصل سوم-تحویل پیوسته نرم افزار

✓ تحویل پیوسته و زود به زود نرم‌افزار قابل استفاده دو سه هفته یک بار تا دو سه ماه یک بار با ترجیح بر فاصله‌های زمانی کوتاه‌تر.

اصل چهارم-همکاری و تعامل

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

اصل پنجم-تیم با انگیزه

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

اصل ششم-تعاملات رو در رو

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

اصول چابک

اصل هفتم-نرم افزار کارکننده

✓ نرم افزار قابل استفاده و کارکننده اصلی ترین معیار سنجش پیشرفت است

اصل هشتم-توسعه پایدار

✓ 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

اصل نهم-برتری فنی و طراحی خوب

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

اصل دهم-سادگی

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

اصل یازدهم-تیم‌های خودسازمان‌ده

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

اصل دوازدهم-بحث و تبادل نظر

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

روش های چابک

برای پیاده سازی توسعه نرم افزار چابک روشهای مختلفی وجود دارد. در حقیقت Agile یک تفکر در زمینه توسعه نرم افزار می باشد و برای پیاده سازی این تفکر در پروژه های توسعه نرم افزار، روش های مختلفی ارائه شده است.

اسکرام (scrum)

کانبان (Kanban)

عبارت ژاپنی «کانبان» (kanban) به معنای «تخته بصری» یا «نشان»، از دهه ۱۹۵۰ میلادی تاکنون برای توصیف یک فرایند استفاده شده است. این فرایند برای نخستین بار توسط تویوتا توسعه یافت و به کار گرفته شد. در روش کانبان تمرکز اصلی بر «تداوم و پیوستگی» است. کل پروژه بر روی تخته کانبان تصویر می‌شود؛ تخته‌ای که در آن کارهای در حال انجام، کارهای انجام شده و کارهایی که باید انجام شوند در لیست‌های جداگانه‌ای نمایش داده می‌شوند. این روش، با تقسیم پروژه به بخش‌های کوچکتر، برخورد با هر مرحله را بهبود می‌بخشد و همواره بر همکاری مداوم و بهبود مستمر تاکید می‌کند.

ناب (Lean)

مدیریت ناب (Lean Management) یک روش شناخته در مدیریت است که برای بهبود کارایی و کیفیت در سازمان‌ها استفاده می‌شود. این روش در ابتدا توسط شرکت تویوتا موتور در دهه ۱۹۵۰ و به منظور بهبود کارآیی و بهره‌وری در فرآیندهای تولید توسعه یافت. با این رویکرد، سازمان‌ها در راستای ایجاد ارزش برای مشتریان، فرایندهای خود را بهینه می‌کنند و هدررفت‌ها و فعالیت‌هایی که ارزش افزوده‌ای ندارند را حذف می‌کنند. لین، به جای اینکه یک روش مبتنی بر فرایند و فهرستی از ملزومات باشد، برای خودش اصولی دارد. شما این روش را انتخاب می‌کنید چون دنبال کار بیشتر با زیادی کاری کمتر هستید.

روش های چابک

کریستال (Crystal)

این شیوه چابک بیشتر بر استقلال تیم‌های توسعه تمرکز دارد و آن‌ها را به بهبود مستمر محصول و حل مشکلات خود تشویق می‌کند. در این متد، افراد و تعاملات بین آن‌ها ارزش بیشتری از رویه‌ها و ابزارها دارند. کریستال بر روی اصولی مانند افراد، تعامل، جامعه، مهارت، استعداد و ارتباطات متمرکز شده و هدف از آن ارائه بهترین فرآیند ممکن برای توسعه نرم افزار است. هسته اصلی این فرآیند توسعه تعامل و همزیستی است که باید بین افرادی که به پروژه ها و فرایندها اختصاص داده می شوند وجود داشته باشد تا کارآیی در توسعه ایجاد شود.

XP (Extreme Programming)

یکی از روش های چابک توسعه نرم افزار است که هدف آن افزایش کیفیت نرم افزار و پاسخگویی به نیازهای در حال تغییر کاربران است. Xpروشی کارآمد، انعطاف‌پذیر، کم خطر، علمی و قابل پیش بینی برای تولید یک نرم افزار است. برنامه‌نویسی مفرط یا XP توسط تیم پروژه‌های کوچک برای توسعه محصولات کوچک و متوسط بکار گرفته می‌شود؛ به خصوص در پروژه‌هایی که الزامات محصول به سرعت تغییر می‌کنند.

SAFe

چارچوب چابک مقیاس پذیر (Scaled Agile Framework) یاSAFe ، به معنی مجموعه‌ای از الگوهای سازمانی و گردش کار برای اجرای روش‌های چابک در مقیاس سازمانی است. این چارچوب یک مجموعه دانش است که شامل هدایت‌های ساختار یافته در مورد نقش‌ها و مسئولیت‌ها، نحوه برنامه ریزی، مدیریت کار و ارزش کارهایی است که باید حفظ شوند. SAFe یکی از محبوب‌ترین Framework های Agileبه شمار می‌رود و هم ترازی، همکاری و تحویل را در میان تعداد زیادی از تیم‌های چابک ترویج می‌کند.

PDF-XChange Editor
Click to BUY NOW!
www.pdf-xchange.com

اسکرام

Scrum

اسکرام یک چارچوب (FrameWork) سبک وزن است که به افراد، تیمها و سازمانها کمک میکند تا از طریق یافتن راه حلهای تطبیق پذیر برای مشکلات پیچیده، ارزش خلق کنند.

اسکرام یکی از روش‌های توسعه نرم افزار و مدیریت پروژه است که می‌تواند در بسیاری از پروژه‌های نرم افزاری و غیر نرم افزاری استفاده شود. تفکر اصلی این چارچوب بر مبنای اجایل است. در واقع اسکرام زیر مجموعه تفکر چابک محسوب می‌شود.

این روش مخصوصاً زمانی کاربرد دارد که دانش اولیه ما در مورد حل یک مسئله به اندازه کافی نبوده و به تدریج به آن اضافه می‌شود.

بر اساس گزارش‌های انجام شده بیشتر از ۷۰ درصد از گروه‌هایی که در زمینه تولید نرم افزار فعالیت می‌کنند از این چارچوب برای انجام فعالیت خود استفاده می‌کنند. گاهی اوقات نیز به صورت ترکیبی از این چارچوب در کنار روش‌های دیگر استفاده می‌شود. این چارچوب برای انجام پروژه‌های پیچیده بسیار مفید است.

اسکرام برای بهینه‌سازی امکان پیش‌بینی پذیری و کنترل ریسک از یک روش چرخشی-افزایشی استفاده می‌کند. اسکرام گروه هایی از افراد را به کار میگیرد که درمجموع همه مهارت ها و تخصص های لازم برای انجام کار را دارند و در صورت نیاز آن مهارت‌ها را با یکدیگر به اشتراک گذاشته و یا کسب میکنند.

ویژگی های پایه اسکرام

اسکرام از روش‌ها و نقش‌های مختلفی تشکیل می‌شود اما دارای سه ویژگی پایه و اساسی است. این ویژگی‌ها شامل موارد زیر هستند:

شفافیت (Transparency): کار و فرآیندِ انجام شده، باید هم برای کسانی که کار را انجام میدهند و هم کسانی که نتیجه آن را دریافت میکنند آشکار باشد.

شفافیت بازرسی را ممکن می سازد. بازرسی بدون شفافیت گمراه کننده و بی فایده است.

بازرسی (Inspection): تمام بخش‌ها و جنبه‌های مختلف پروژه و میزان پیشرفت در مسیر اهداف توافق شده، باید به طور مستمر و با دقت بازرسی شوند تا انحرافات نامطلوب یا مشکلات بالقوه شناسایی شوند.

بازرسی بستری است برای سازگاری که بدون آن عملا بی معنی است.

سازگاری (Adaptation): اگر در مرحله بازرسی تشخیص داده شود که هر یک از جنبه‌های یک فرآیند، از محدوده‌های قابل قبول منحرف شود یا محصول به دست آمده قابل قبول نباشد، باید فرایند اعمال شده یا آنچه تولید شده است سازگار شود. برای به حداقل رساندن انحراف در آینده، سازگاری باید هرچه سریعتر انجام شود.

ارزش های اسکرام

اسکرام براساس پنج ارزش معنا پیدا می‌کند: تعهد، تمرکز، بازبودن، احترام و شجاعت.

1. **تعهد (Commitment)**

تیم اسکرام در حمایت از یکدیگر و رسیدن به اهدافشان باهم متعهدند.

۲. تمرکز (Focus)

تمرکز اصلی آنها بر روی کارهای اسپرینت است تا بهترین پیشرفت ممکن را در راستای این اهداف داشته باشند.

۳. بازبودن (Openness)

تیم اسکرام و ذینفعانشان در مورد کار و چالشهایش باز عمل میکنند.

۴. احترام (Respect)

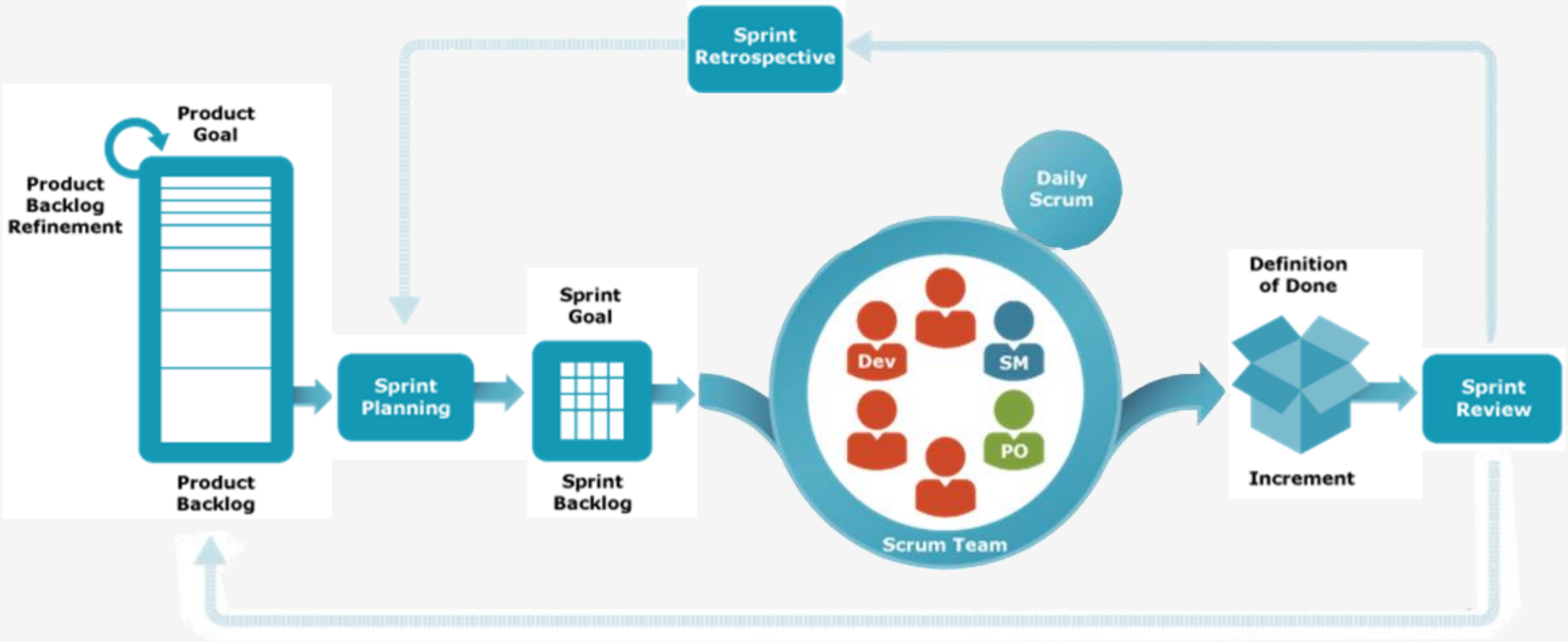
اعضای تیم اسکرام برای اینکه افرادی توانمند و مستقل باشند به هم احترام میگذارند و به همین ترتیب از سوی افرادی که با آن ها کار میکنند دارای احترام هستند.

۵. شجاعت (Courage)

اعضای تیم اسکرام شجاعت الزم برای انجام کار درست و کار کردن بر روی مسائل دشوار را دارند.

این ارزشها به تیم اسکرام در کار، اعمال و رفتارشان جهت میدهد. تصمیماتی که گرفته میشود، گامهایی که برداشته میشود و نحوه استفاده از اسکرام باید تقویت کننده این ارزشها باشد، نه کاهنده یا تضعیف کننده آنها. اعضای تیم اسکرام این ارزشها را در حین کار با رویدادها و مصنوعات اسکرام، فراگرفته و کشف می کنند. هنگامیکه این ارزشها در تیم اسکرام و افرادی که با آنها کار میکنند نهادینه شود، ارکان تجربه گرایانه اسکرام، شفافیت، بازرسی و سازگاری به وجود آمده و اعتماد ساخته میشود.

چارچوب اسکرام



تیم اسکرام

- واحد بنیادین اسکرام یک تیم کوچک از افراد، یا همان تیم اسکرام است. تیم اسکرام متشکل از یک Scrum Master، یکOwner Product (مالک محصول) و Developers (توسعه دهندگان) است.
- در یک تیم اسکرام هیچگونه زیرتیم یا سلسله مراتبی وجود ندارد. این تیم یک واحد منسجم از حرفه ای هاست که در هرلحظه بر روی یک مقصود که همان هدف محصول است، متمرکزند.
- تیمهای اسکرام فراوظیفه ای هستند. به این معنا که اعضایش تمام مهارت های لازم برای خلق ارزش در هر اسپرینت را دارند.
- آنها خودمدیریت نیز هستند. به این معنا که درون تیم تصمیم میگیرند که چه کسی، چه کاری را در چه وقت و چگونه انجام دهد.
- تیم اسکرام آنقدر کوچک است که چابک بماند و آنقدر بزرگ است که بتواند کار قابل توجهی را در یک اسپرینت کامل کند. معمولا ۱۰ نفر یا کمتر.
- اگر تیمهای اسکرام بیش از اندازه بزرگ شوند، باید در قالب چند تیم منسجم اسکرام که همه بر روی همان محصول متمرکزند، مجدد سازماندهی شوند. پس همگی باید در یک هدف محصول، یک بک لاگ محصول و یک مالک محصول شریک باشند.
- تیم اسکرام مسئول تمام فعالیتهای مرتبط با محصول است. از تعامل با ذینفعان تا ارزیابی، نگهداری، بهره برداری، آزمایش، تحقیق و توسعه و هر چیز دیگری که ممکن است لازم شود.
- کار کردن با ضرب آهنگ پایدار در اسپرینتها، تمرکز و ثبات تیم اسکرام را بهبود میدهد.
- در طول هر اسپرینت، کل تیم اسکرام در قبال ایجاد یک فرآورده ارزشمند و قابل استفاده پاسخ گوست.

تیم اسکرام > Product Owner (مالک محصول)

مالک محصول یک تصویرساز از محصول آینده است و به عنوان صدای مشتری یا کاربر محصول عمل می کند. تمرکز مالک محصول بر جنبه تجارتی و کسب و کار توسعه محصول است. مالک محصول فردی است که تصمیم می‌گیرد محصول باید چه ویژگی‌های و قابلیت‌هایی داشته باشد و این ویژگی‌ها و قابلیت‌ها چه ترتیب و اولویتی دارند.

مالک محصول پاسخگوی به حداکثر رساندن ارزش محصولی است که از کار تیم اسکرام نتیجه می شود. روش انجامش ممکن است بر اساس نوع سازمان، تیمهای اسکرام و افراد آن بسیار متفاوت باشد.

مالک محصول در مورد مدیریت مؤثر Product Backlog هم پاسخگوست که شامل موارد زیر است:

- توسعه هدف محصول و تعامل صریح بر سر آن؛
- ایجاد اقلام Product Backlog و تعامل روشن بر سر آنها؛
- رتبه بندی اقلام Product Backlog؛
- اطمینان از شفافیت، قابل مشاهده و درک بودن Product Backlog

مالک محصول ممکن است این کارها را خودش انجام دهد یا مسئولیتش را به دیگران واگذار کند. درهرصورت مالک محصول پاسخگو می باشد.

مالک محصول یک نفر است، نه یک کمیته. ممکن است نیازهای ذینفعان زیادی را در Product Backlog اعمال کند. آنهایی که خواهان تغییر Product Backlog هستند باید این کار را با تلاششان در متقاعد کردن Product Owner انجام دهند.

تیم اسکرام > Scrum Master (اسکرام مستر)

اسکرام مستر مسئول استقرار اسکرام است. اسکرام مسترها این کار را با کمک کردن به افراد درون تیم اسکرام و سازمان در جهت درک تئوری اسکرام و تمرین شیوه هایش، انجام میدهند.

اسکرام‌مستر عموما با عنوان رهبر خدمتگزار نیز شناخته می‌شود و مربی تیم چابک است. نقش ها و مسئولیت های اسکرام مستر شبیه مدیر پروژه است. تاکید اسکرام مستر بر تسهیل امور تیم و کسب اطمینان از این امر است که چارچوب اسکرام در تیم و پروژه دنبال می شود.

کار اسکرام‌مستر این است که به همه کمک کند. اسکرام‌مستر با همه اعضای تیم ارتباط نزدیک دارد و برای هرکدام از آن‌ها، مجموعه‌مسئولیت‌های مختلفی مشخص می‌کند. اسکرام مستر از همه اعضاء نسبت به اسکرام دانش بیشتری دارد. او باید ارزش‌ها و اصل‌ها و روش‌های اسکرام را درک و برای دیگران تحلیل کند.

وظیفه اصلی اسکرام مستر این است که:

موانع سازمانی و یا هرمانع دیگری که مانع افراد در انجام کارهایشان می شود را برطرف کند.

رویدادهای تیمی را تسهیل کند و فرآیند ها را بهبود ببخشد.

روند پیشرفت اسپرینت ها را پیگیری کند.

تیم اسکرام > Developers (توسعه دهندگان)

توسعه دهندگان متخصصانی در تیم اسکرام هستند که در پایان هر اسپرینت، Incrementیا بخش افزایشی محصول بالقوه را تحویل می‌دهند.

این توسعه دهندگان هستند که کار اصلی از تحلیل گرفته تا طراحی، کد زدن و آزمایش را انجام می دهند.

مهارتهای خاص موردنیاز برای توسعه دهندگان اغلب گسترده بوده و نسبت به حوزه کار متفاوت است. بااین حال، توسعه دهندگان همیشه در مورد موارد زیر پاسخگو هستند:

- ایجاد یک برنامه برای اسپرینت، یا همان Sprint Backlog؛
- سازگار کردن روزانه برنامه شان نسبت به هدف اسپرینت؛
- تأمین کیفیت از طریق وفادار ماندن به تعریف تکمیل شده؛
- پاسخگو نگه داشتن یکدیگر به عنوان افرادی حرفه ای.

اسپرینت (Sprint)

اسپرینت، نبضِ تپندهٔ اسکرام است، جایی که ایده ها به ارزش تبدیل میشوند.

اسپرینت یک بازه زمانی کوتاه است که تیم اسکرام در این بازه زمانی، مقدار مشخصی از محصول یا پروژه را تکمیل می‌کند.

در روش اسکرام، محصول در دوره‌های زمانی تکراری به نام اسپرینت ساخته می‌شود که پروژه‌های عظیم و پیچیده را به بخش‌های کوچک‌تری تقسیم‌بندی می‌کند.

اسپرینتها رویدادهایی با طولِ ثابتِ یکماهه یا کمترند (معمولا یک هفته الی 4 هفته) تا ثبات ایجاد شود. اسپرینت جدید بلافاصله بعد از به سرانجام رسیدن اسپرینت قبلی آغاز میشود.

در طولِ اسپرینت:

- هیچ تغییری ایجاد نمیشود که هدف اسپرینت را به خطر بیندازد؛
- کیفیت کاهش نمی یابد؛
- بک‌لاگ محصول در صورت لزوم پالایش میشود؛
- محدوده کار ممکن است با توجه به یادگیری های جدید با مالک محصول مجدداً مورد مذاکره و تصحیح قرار گیرد.

یک اسپرینت زمانی که هدفش منسوخ شده یا دیگر معتبر نباشد میتواند لغو شود. تنها مالک محصول اختیار لغو اسپرینت را دارد.

رویدادهای اسکرام > Sprint Planning (برنامه‌ریزی اسپرینت)

Sprint Planning رویدادیست که در ابتدای اسپرینت با حضور اعضاء تیم اسکرام تشکیل می شود. مالک محصول اطمینان حاصل میکند که شرکت کنندگان آماده بحث و گفتگو درباره مهمترین اقلام بک‌لاگ محصول و چگونگی نگاشت آنها به هدف محصول باشند. تیم اسکرام میتواند افراد دیگری را هم برای مشورت گرفتن به Planning Sprint دعوت کند.

Planning Sprint موضوعات زیر را پوشش میدهد:

موضوع یک: چرا این اسپرینت ارزشمند است؟

موضوع دو: چه چیزی در این اسپرینت میتواند انجام شود؟

موضوع سه: کارهای انتخاب شده چگونه انجام خواهند شد؟

این جلسه دو خروجی مهم دارد:

الف) هدف اسپرینت

ب) اقلام انتخاب شده از بک‌لاگ محصول برای اسپرینت بعلاوه طرح تحویل آنها که همه باهم بک‌لاگ اسپرینت نامیده میشود.

زمانِ بستهٔ Planning Sprint برای یک اسپرینت یکماهه هشت ساعت است. برای اسپرینت‌های کوتاه تر، این رویداد معمولا کوتاه تر است.

رویدادهای اسکرام > Daily Scrum (اسکرام روزانه)

مقصود از Daily Scrum بازرسی پیشرفت کار در جهت رسیدن به هدف اسپرینت و در صورت نیاز سازگاری بکلاگ اسپرینت و تنظیم کردن کارهای برنامه ریزی شده آینده است.

اسکرام روزانه یک رویداد ۱۵ دقیقه ای برای توسعه دهندگان تیم اسکرام است. جهت کاهش پیچیدگی، این رویداد در هر روز کاری یک اسپرینت، در زمان و مکان ثابت برگزار میشود. چنانچه مالک محصول یا اسکرام مستر بر روی اقلامی از بکلاگ اسپرینت فعالانه کار میکنند، ایشان نیز مانند توسعه دهندگان در این رویداد شرکت میکنند.

Daily Scrum ارتباطات را بهبود میبخشد، موانع را شناسایی میکند، تصمیم گیری سریع را ترویج میدهد و درنتیجه نیاز به جلسات دیگر را برطرف میکند. البته Daily Scrum تنها زمانی نیست که توسعه دهندگان مجاز به تنظیم برنامه شان باشند. آنها اغلب در طول روز برای بحث‌های جزئی‌تر در مورد انطباق یا برنامه‌ریزی مجدد کارهای باقیمانده اسپرینت باهم دیدار میکنند.

معمولا اعضاء تیم در این جلسه به 3 سوال مهم پاسخ می دهند:

1. دیروز چه کاری انجام دادم؟
2. امروز چه کاری انجام خواهم داد؟
3. به چه موانع و مشکلاتی برخوردم؟

رویدادهای اسکرام > Sprint Review (بازبینی اسپرینت)

Sprint Review رویدادیست که در انتهای اسپرینت با حضور اعضاء تیم اسکرام، ذینفعان و سایرین تشکیل می شود.

مقصود از Sprint Review، بازرسی برآیند اسپرینت و تعیین سازگاری‌های آینده است. در این رویداد، تیم اسکرام خروجی کار خود را به ذینفعان اصلی ارائه میدهد و پیشرفت انجام شده در جهت هدف محصول به بحث گذاشته میشود.

در طول این رویداد، تیم اسکرام و ذینفعان آنچه را که در اسپرینت انجام شده و تغییراتی که در محیطشان روی داده است را مرور میکنند. شرکت کنندگان بر اساس این اطلاعات در مورد اقدامات آتی باهم تعامل میکنند. بکلاگ محصول نیز ممکن است متناسب با فرصتهای جدید تنظیم شود

Sprint Review یک جلسه کاری است و تیم اسکرام باید از محدود کردن آن صرفاً به یک جلسه نمایش خودداری کند.

Sprint Review (بازبینی اسپرینت) رویداد ماقبل آخر اسپرینت بوده و زمان بستهٔ آن در یک اسپرینت یک ماهه حداکثر چهار ساعت است. برای اسپرینت‌های کوتاه‌تر، این رویداد معمولاً کوتاه‌تر است.

رویدادهای اسکرام > Sprint Retrospective (بازاندیشی اسپرینت)

Sprint Retrospective رویدادیست که در انتهای اسپرینت با حضور اعضاء تیم اسکرام برگزار می شود.

هدف Sprint Retrospective (بازاندیشی اسپرینت)، برنامه‌ریزی و تعریف روشهایی برای افزایش کیفیت و اثربخشی است.

تیم اسکرام چگونگی پیشرفت اسپرینت قبل را با توجه به افراد، تعاملات، فرآیندها، ابزارها و تعریف تکمیل شده بازرسی میکند.

در این جلسه تیم اسکرام در این خصوص که در اسپرینت قبل چه چیزی خوب پیش رفته است، با چه مشکلاتی روبرو شده‌اند و آن مشکلات چگونه حل شده یا نشده‌اند، بحث میکنند.

در این جلسه اعضای تیم در مورد موضوعات زیر صحبت می کنند:

- چه کارهایی در این اسپرینت به‌خوبی انجام شده است؟
- چه کارهایی به‌خوبی پیش نرفته است؟
- به‌منظور بهبود امور، باید چه کارهایی آغاز کنیم؟

Sprint Retrospective (بازاندیشی اسپرینت)، اسپرینت را خاتمه میدهد. زمان بستهٔ Retrospective (بازاندیشی) برای یک اسپرینت یکماهه حداکثر سه ساعت بوده که در مورد اسپرینت‌های کوتاه‌تر، زمان این رویداد معمولاً کمتر است.

مصنوعات اسکرام (Scrum Artifacts)

مصنوعات اسکرام نشان دهنده ارزش یا کاری هستند. آنها طوری طراحی شده‌اند که شفافیت اطلاعات کلیدی را به حداکثر برسانند؛ بنابراین همه افرادی که آنها را بازرسی میکنند، مبنای یکسانی برای سازگاری و انطباق دارند.

مهمترین مصنوعات اسکرام عبارتند از:

- **Product Backlog**
- **Sprint Backlog**
- **Increment**

هر مصنوع دربردارنده یک تعهد است تا متضمن این باشد که با فراهم کردن اطلاعاتی در جهت بهبود میزان شفافیت و تمرکز بتوان پیشرفت را اندازه گیری کرد:

- برای بکلاگ محصول، هدف محصول است.
- برای بکلاگ اسپرینت، هدف اسپرینت است.
- برای Increment، تعریف تکمیل شده است.

این تعهدات به منظور تقویت تجربه گرایی و ارزشهای اسکرام، برای تیم اسکرام و ذینفعانشان وجود دارند.

مصنوعات اسکرام > Product Backlog (بک‌لاگ محصول)

بک‌لاگ محصول فهرستی از ویژگی‌های اولویت‌بندی‌شده است که باید در محصول نهایی لحاظ شود. این تنها منبع کاری است که تیم اسکرام آن را به عهده گرفته است.

اقلامی از بکلاگ محصول که قابلیت تکمیل شدن توسط تیم اسکرام در یک اسپرینت را داشته باشند، به عنوان اقلامِ «آماده» برای انتخاب در یک جلسه Sprint Planning شناخته میشوند.

آنها معمولاً بعد از فعالیتهای پالایش به این درجه از شفافیت میرسند. پالایش بکلاگ محصول عمل شکستن و تعریف کردن اقلام بکلاگ محصول به اقلام کوچکتر و دقیقتر است. این یک فعالیت مداوم برای افزودن جزئیاتی مانند توضیحات، رتبه و اندازه است. این ویژگیها نسبت به حوزه کاری میتوانند متفاوت باشند.

توسعه دهندگانی که کار را انجام خواهند داد مسئول برآوردن کردن کار هستند. مالک محصول ممکن است با کمک در فهم و ایجاد مقایسه، بر روی توسعه دهندگان تأثیر بگذارد.

محصول، وسیله ای برای تحویل ارزش است. مرزی شفاف، ذینفعانی مشخص و تعریف واضحی از کاربران یا مشتریان دارد.

محصول میتواند یک سرویس، محصولی فیزیکی و یا چیزی انتزاعی‌تر باشد.

مصنوعات اسکرام > Sprint Backlog (بک‌لاگ اسپرینت)

بک‌لاگ اسپرینت فهرستی از وظایفی است که تیم اسکرام باید متعهدانه آن‌ها را تا پایان اسپرینت انجام دهد.
در جلسه‌ی برنامه‌ریزی اسپرینت، تیم توسعه این آیتم‌ها را برمبنای اولویت از فهرست بک‌لاگ محصول انتخاب می‌کند.
بکلاگ اسپرینت از هدف اسپرینت (چرایی)، اقلام انتخاب شده از بکلاگ محصول برای اسپرینت (چه ها) و همچنین یک برنامه عملیاتی برای تحویل یک Increment (چگونگی) تشکیل شده است.
بکلاگ اسپرینت برنامه ای است تهیه شده توسط توسعه دهندگان و برای توسعه دهندگان است.
بکلاگ اسپرینت تصویری به شدت شفاف و لحظه ای از کارهایی است که توسعه دهندگان برنامه ریزی میکنند تا با تکمیل آن در طول یک اسپرینت به هدف اسپرینت برسند. پس بکلاگ اسپرینت در تمام طول اسپرینت توسط توسعه دهندگان اصلاح و به روز میشود. این برنامه باید جزییات کافی داشته تا آنها بتوانند در Daily Scrum میزان پیشرفتشان را بازرسی کنند.

مصنوعات اسکرام > Increment (فرآورده)

واژه‌ی Increment به‌معنای افزایش و ارتقا، به مفهوم رسیدن به سطح بعدی است. درواقع، Increment قدمی به‌سمت هدف یا چشم‌انداز است. Increment شامل بخش افزایشی محصول، مجموعه آیتم‌های کلی به‌پایان‌رسیده در طول هر اسپرینت و همچنین اسپرینت‌های تکمیل‌شده‌ی قبلی است.

یک Increment سنگ‌قدمی عینی به سوی هدف محصول است. هر Increment افزوده‌ای است بر همه Increment های قبلی و کاملاً تائید شده که اطمینان میدهد تمامی Increment ها با یکدیگر کار میکنند. به منظور فراهم آوردن ارزش، Increment باید قابل استفاده باشد.

در طول هر اسپرینت، تمام فعالیت‌های ضروری مانند تجزیه‌وتحلیل، طراحی، ساخت، ادغام و آزمایش را تیم اسکرام انجام می‌دهد تا Increment قابل‌استفاده تولید شود که محصول کامل متفاوتی است.

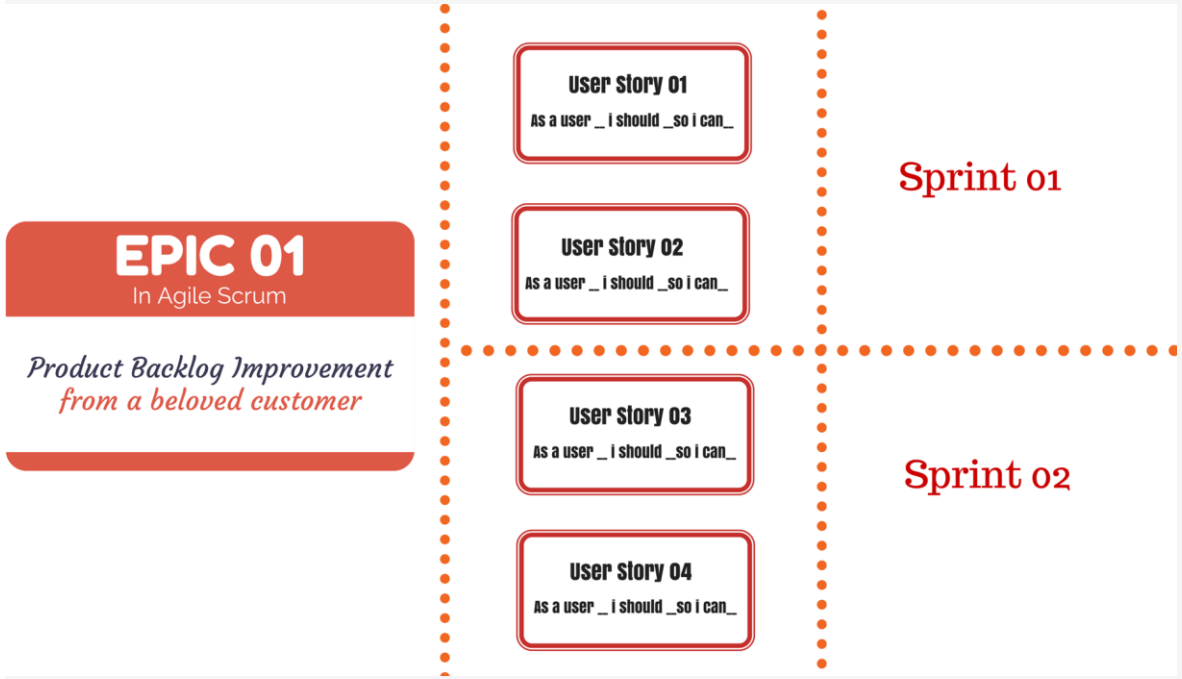
اپیک (EPIC)

Epic در اسکرام، یک وظیفه بزرگ است که می‌تواند به «داستان‌های کاربر» (User Stories) کوچکتر تقسیم شود.

اپیک می‌تواند توضیحی سطح بالا از آنچه مشتری طلب می‌کند باشد. اپیک یک پیش‌نیاز سطح بالا است و به همین خاطر، ابعاد و چشم‌انداز آن می‌تواند در گذر زمان دچار تغییر شود.

یک اپیک می‌تواند میان اسپرینت های مختلف و یا حتی تیم‌های چابک مختلف پخش شود.

اپیک‌ها راهی کارآمد برای سامان‌بخشی به کارها و ساخت سلسله مراتب است. ایده کلی این‌ست که وظایف را به تکه‌های خرد و قابل پیاده‌سازی تقسیم کنید تا پروژه‌های بزرگ واقعا عملی شوند و بتوانید به صورت مداوم برای مشتریان خود ارزش‌سازی کنید. اپیک به تیم‌ها در تقسیم‌بندی کارها و حرکت به سمت هدفی بزرگ‌تر کمک می‌کند.



**

داستان کاربر (User Story)

داستان کاربر یک روش برای توضیح قابلیت یا ویژگی مورد نظر از زبان کسی است که نیازمند آن است (معمولا کاربر یا مشتری).
خاصیت ویژه این توضیح، کوتاه و مختصر بودن، پرداختن به نیاز واقعی کاربر و تعاملی بودن آن است.
داستان یا استوری کاربر (User Story) بخشی از رویکرد توسعه‌ی نرم‌افزار چابک است که جزئیات الزامات یا نیازمندی‌ها را از دیدگاه کاربر نهایی بیان می‌کند. داستان کاربر که در بک لاگ محصول قرار می گیرد شامل چندین نوع کار می باشند و عموما چیزی است که بیش از یک نفر بر روی آن کار می کنند.
داستان کاربر مشخص می‌کند شما چه نوع کاربری هستید و چه چیزی می‌خواهید و چه دلیلی پشت این خواستن وجود دارد.
به‌عبارت ساده، داستان کاربر به تیم چابک کمک می‌کند توصیفی کوتاه و ساده از ویژگی های محصول را از نگاه کاربر ذکر کند.

یوزر استوری‌ها معمولا در قالب جملاتی ساده نوشته می‌شوند و ساختاری مشابه آنچه در ادامه آورده‌ایم دارند:

«به عنوان یک [نقش کاربر]، من [می‌خواهم که...]، [تا این‌طور شود...].»

وظیفه (Task)

یک داستان کاربر به کارهای کوچکتر شکسته می شود که قابلیت انجام توسط یک نفر را داشته باشد که به آن تسک گفته می شود.

تسک کاریست که از یک نوع بوده و توسط یک نفر قابل انجام است.

تَسک ها در جلسه برنامه ریزی اسپرینت، شناسایی شده و بخشی از بک لاگ اسپرینت هستند.

وظایف محدود به یک نوع کار می شوند و فقط توسط یک نفر انجام می شود.

یک تَسک، معمولا چیزی شبیه کد زدن ... ، طراحی ... ، ایجاد داده های تست برای .... ، اتوماسیون ... و غیره است. تمام این موارد چیزهایی هستند که باید یک نفر آنها را انجام دهد.

PDF-XChange Editor
Click to BUY NOW!
www.pdf-xchange.com

کانبان

Kanban