

مهندسی نرم افزار ۲

NINTH EDITION

Software Engineering

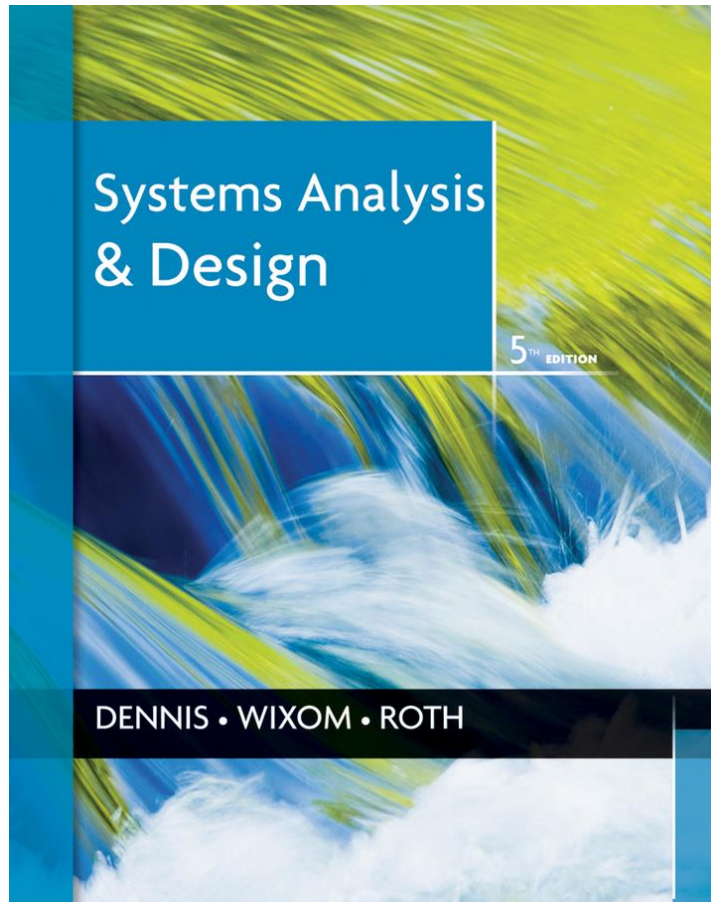
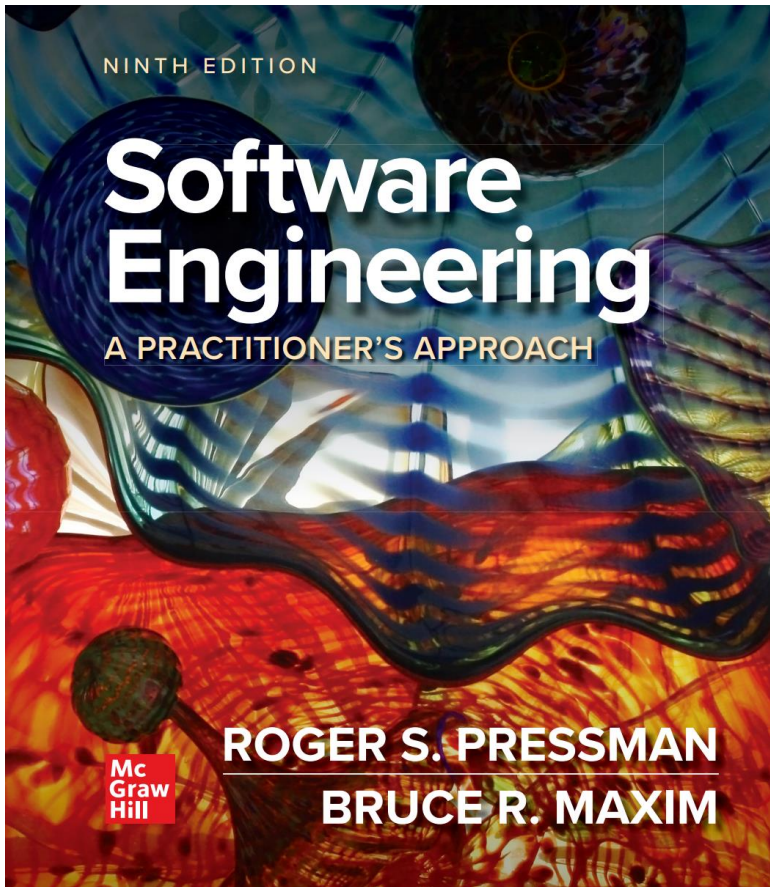
A PRACTITIONER'S APPROACH

ROGER S. PRESSMAN

BRUCE R. MAXIM

Mc
Graw
Hill

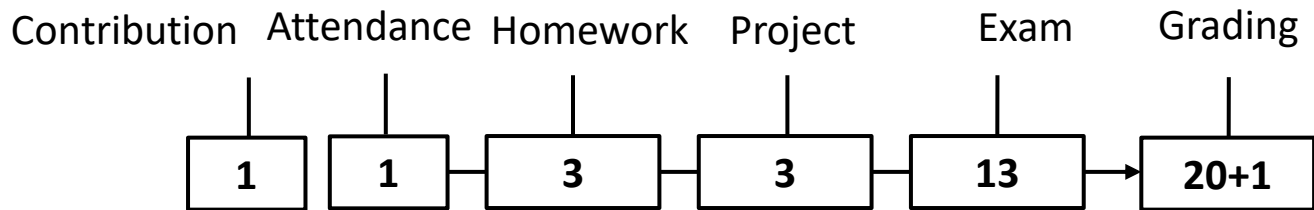
منابع



فهرست مطالب

- مقدمه‌ای بر مهندسی نرم افزار
- فرایندهای نرم افزار
- مدل سازی
 - مدل کردن نیازمندی‌ها
 - مفاهیم طراحی
- کیفیت و امنیت
 - تضمین کیفیت نرم افزار
 - نیازمندی‌های امنیتی نرم افزار
- مدیریت پروژه
 - مدل توسعه اسکرام و چابک
 - نمودار پرت و گانت
 - مفاهیم مدیریت پروژه

ارزیابی



ارزیابی پروژه

- ارائه پروژه صنعتی:
- نوع ارزیابی
- (۱: کیفیت مطلب
- (۲: مفاد ارائه (اسلایدها، استفاده از تصاویر)
- (۳: کیفیت ارائه (نحوه بیان و ارائه جذاب)

مقدمه‌ای بر مهندسی نرم افزار

نرم افزار چیست ؟

- دستورات برنامه‌های کامپیوتری که در صورت اجرا شدن باعث انجام عمل و کارایی خواسته شده می‌شوند.
- ساختمان داده‌هایی که باعث می‌شود برنامه‌ها بطور مناسبی اطلاعات را دستکاری کنند.
- مستنداتی که توصیف کننده عملکرد برنامه‌ها هستند.
- در مفهوم عام در هر محیطی (کارخانه، آموزش و ...) مستقل از IT یک تفکر گرداننده سیستم موجود می‌باشد که سخت افزار موجود را وادار به حرکت می‌نماید.

نگاہ اجمالی

QUICK LOOK

What is it? Computer software is the product that software professionals build and then support over the long term. It encompasses programs that execute within a computer of any size and architecture, content that is presented as the computer programs execute, and descriptive information in both hard copy and virtual forms that encompass virtually any electronic media.

Who does it? Software engineers build and support software, and virtually everyone in the industrialized world uses it either directly or indirectly.

Why is it important? Software is important because it affects nearly every aspect of our lives and has become pervasive in our commerce, our culture, and our everyday activities.

What are the steps? Customers and other stakeholders express the need for computer software, engineers build the software product, and end users apply the software to solve a specific problem or to address a specific need.

What is the work product? A computer program that runs in one or more specific environments and services the needs of one or more end users.

How do I ensure that I've done it right? If you're a software engineer, apply the ideas contained in the remainder of this book. If you're an end user, be sure you understand your need and your environment and then select an application that best meets them both.

خصوصیات نرم افزار

۱- نرم افزار توسعه می یابد یا طراحی می شود، اما به مفهوم کلاسیک ساخته نمی شود.

گرچه شباهت هایی بین توسعه نرم افزار و ساخت سخت افزار وجود دارد، اما این دو فعالیت اساساً با هم متفاوتند.

✓ در هر دو آنها کیفیت بالا حاصل طراحی خوب خواهد بود، اما مرحله ساخت در مورد سخت افزار می تواند یک سری مشکلات کیفیتی داشته باشد که در مورد نرم افزار وجود ندارد.

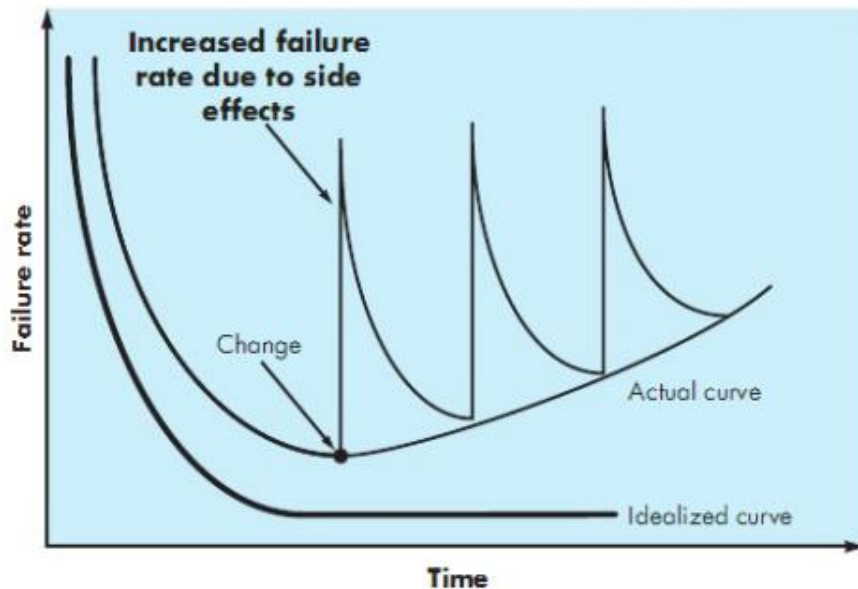
✓ هر دو فعالیت وابسته به مردم است اما ارتباط بین افراد متخصص و کار صورت گرفته کاملاً متفاوت است.

✓ هر دو مستلزم ساختن یک محصول هستند اما روش ها متفاوتند

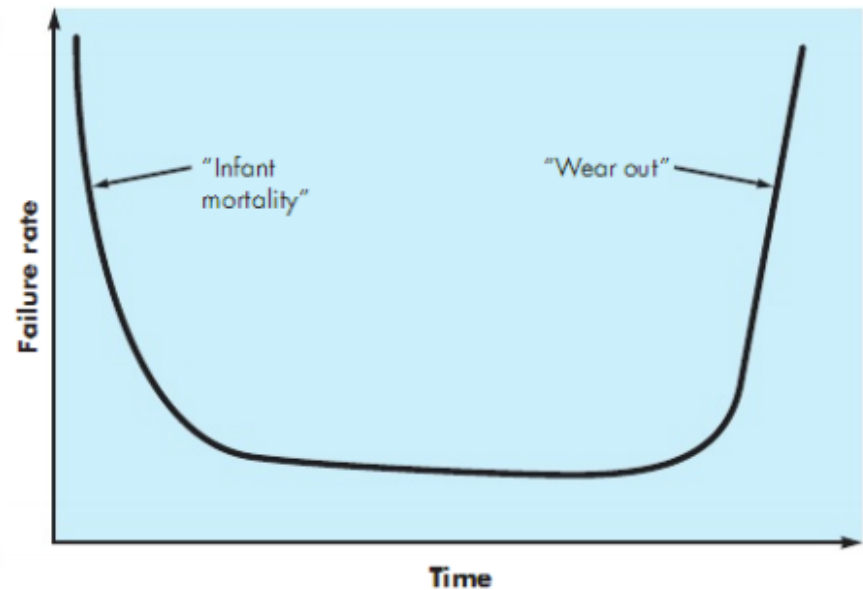
خصوصیات نرم افزار

۲- نرم افزار فرسوده نمی شود اما کیفیت خود را از دست می دهد.

سخت افزار در اوایل عمرش میزان عدم موفقیت نسبتاً بالایی دارد. نقایص اصلاح شده و میزان شکست برای مدتی به سطح ثابتی میرسد و با گذشت زمان، سخت افزار فرسوده می شود و نرخ شکست دوباره افزایش می یابد.



منحنی شکست نرم افزار



منحنی شکست سخت افزار

مهندسی نرم افزار:

مجموعه ای از تکنیک، قواعد، استانداردها و ابزار مهندسی به منظور تولید یک نرم افزار

۱- مقرون به صرفه ($\text{Benefit} > \text{cost}$)

۲- با کیفیت: ارضاء سطح نیازمندیهاست. یک مفهوم نسبیت

است و از یک نرم افزار به نرم افزار دیگر متفاوت است.

مهندسی نرم افزار (ادامه):

$$\text{Cost}_{\text{life cycle}} = \text{Cost}_{\text{Develop}} + \text{Cost}_{\text{maintenance}}$$

$$\text{Cost}_{\text{maintenance}} = 4\text{Cost}_{\text{Develop}}$$

- به طور میانگین ۷۰ درصد هزینه تولید نرم افزار مربوط به نگهداری آن است.
 - کیفیت بالای تولید و توسعه نرم افزار
 - بهتر شدن ابزارها و تجربه‌های ساخت نرم افزار
 - مطالعات جدیدتر درصد فعالیت‌های مربوط به تعمیر و نگهداری نرم افزار که مرتبط با رفع اشکال هستند را نزدیک به ۲۱٪ گزارش کرده‌اند.

چرا اصول طراحی نرم افزار؟

نظریه مهندسی نرم افزار در سال ۱۹۶۸ در کنفرانس «بحران نرم افزار» مطرح گردید. در آن زمان تولید نرم افزارها به صورت غیر رسمی (سلیقه‌ای) بود. بنابراین نرم افزارهای تولید شده دارای اشکالات زیر بودند:

- انجام پروژه‌های بزرگ **سال‌های طولانی** طول می‌کشید.
- **هزینه واقعی** نرم افزار **بیش از مقادیر تخمین زده** بود. (هزینه تولید نرم افزار رو به افزایش بود)
- **عدم تطابق** نرم افزار تحویل شده با مشخصات تعیین شده.
- قابل اعتماد نبودند (**کیفیت پایین** نرم افزار).
- نگهداری آن‌ها دشوار بود (**نگهداری پر هزینه** نرم افزار).
- نداشتن سنجش میزان پیشرفت کار (۹۰ خط از ۱۰۰ خط برنامه ۹۰٪ برنامه نیست).
- پیشرفت سریع سخت افزار

مهندس نرم افزار

- مهندس نرم افزار کسی است که بتواند از ابزارها و تکنیک های موجود به کمک علم خود بهره گیرد و پس از تجزیه و تحلیل مساله آن را پیاده سازی و مدیریت نماید.

دوره تکامل نرم افزار

• دوره اول ۱۹۶۵ - ۱۹۵۰

□ نرم افزار های تک کاربره بودند

□ تمام فرآیند توسعه نرم افزار متکی به یک نفر بود.

□ یک نفر هم نرم افزار را توسعه میداد هم اجرا و هم

پشتیبانی میکرد.

دوره تکامل نرم افزار (ادامه)

- دوره دوم ۱۹۶۵ - ۱۹۷۵
 - ایده چند برنامه ای (multi programming) و سیستم های چند کاربره (multi User)
 - روش های محاوره ای در نرم افزار
 - توسعه سیستم های بلادرنگ و تولید خروجی در چند میلی ثانیه به جای چند دقیقه
 - نرم افزار ها کمی پیچیده تر شدند و انبوهی از نرم افزارهای بدون استاندارد تولید شدند
- اولین نسل از سیستمهای مدیریت بانک اطلاعات بحران نرم افزار به وقوع پیوست.

دوره تکامل نرم افزار (ادامه)

- دوره سوم ۱۹۸۵ - ۱۹۷۵
 - ریز پردازنده‌های کامپیوترهای شخصی کاربرد گسترده‌ای پیدا کرد.
 - توسعه و مطرح شدن سیستم‌های تعبیه شده (Embedded).
 - توسعه‌ی شبکه‌های محلی (LAN)
 - نرم افزارها به خود حالت توزیع شده گرفتند.

دوره تکامل نرم افزار (ادامه)

- دوره چهارم ۱۹۸۵ – تا کنون
 - مفهوم برنامه نویسی شی گرا
 - قابلیت استفاده مجدد از مولفه ها
 - پیدایش نرم افزارهای خبره و پردازشات موازی

داستان نرم افزار

- بسیاری از عوامل ایجاد مشکلات نرم افزاری از طریق علم داستان سرایی که در اوایل تاریخ توسعه نرم افزار بوجود آمد، دنبال می شود.
- داستانهای نرم افزاری اطلاعات غلط و گیج کننده ای را انتشار دادند.
- امروزه اکثر افراد حرفه ای این داستانها را حاوی صفات گمراه کننده ای می دانند که مشکلات زیادی را برای مدیران و افراد متخصص ایجاد می نماید.

➤ انواع اشتباهات، تصورات و داستانهای نرم افزاری

1. داستانهای مدیریتی

2. داستانهای مشتری

3. داستانهای متخصصین و مجریان

داستان نرم افزار (ادامه)

• داستانهای مدیریتی

• مدیران مسئول نرم افزار، مانند اکثر مدیران دیگر، اغلب برای مواردی از قبیل تامین بودجه، جلوگیری از لغزش زمانبندی پروژه و ارتقاء کیفیت، تحت فشارند.

• **داستان:** قبلا کتابی در اختیار داشتیم که حاوی استانداردها و رویه هایی برای تولید نرم افزار بود ولی آیا کتاب تمام آنچه را افراد نیاز دارند، در اختیار دارد.

• **واقعیت:** (در بسیاری از موارد جواب سئوالات زیر **خیر** است)

• کتاب استانداردها می تواند وجود داشته باشد

1. آیا افراد از این کتاب مطلع هستند؟

2. آیا از کتاب استفاده می شود؟

3. آیا کتاب، منعکس کننده مهندسی نرم افزار مدرن است؟

4. آیا کتاب کامل است؟

5. آیا کتاب، زمان تولید نرم افزار را کاهش می دهد؟

6. آیا کتاب، باعث ارتقاء سطح کیفی نرم افزار می شود؟

داستان نرم افزار (ادامه)

• داستانهای مشتری

• مشتریانی که تقاضای سیستم نرم افزاری دارند، فرد همکار، یک گروه کاری، واحد بازاریابی یا فروش شرکت خارجی باشند که نرم افزار را از طریق قرارداد درخواست نموده اند.

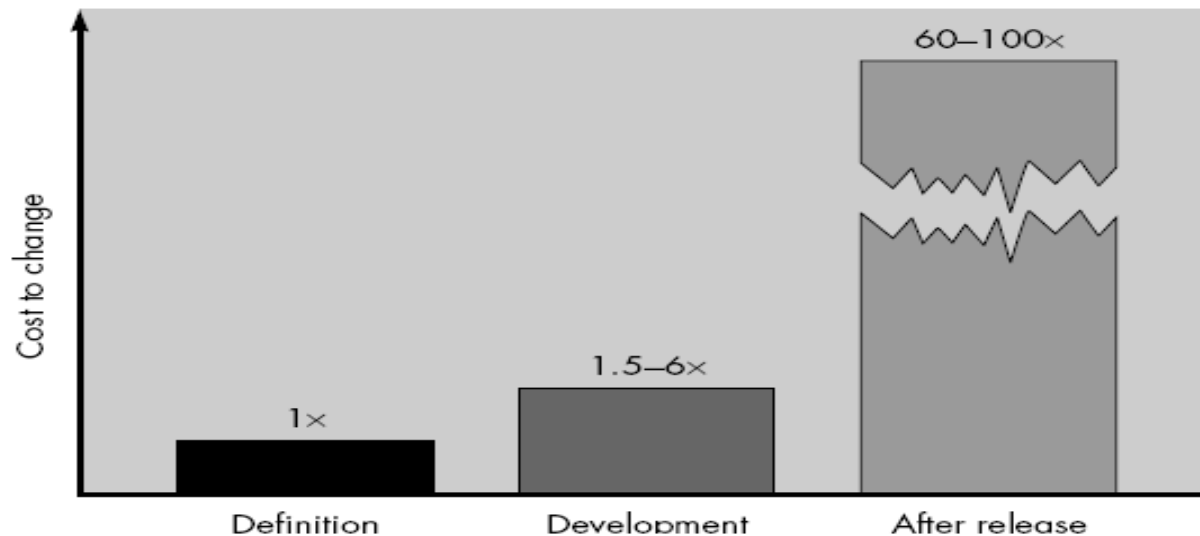
• **داستان:** نیازسنجی (مهندسی نیاز) پروژه دائم در حال تغییر است، اما این تغییرات به راحتی سازماندهی می شوند، زیرا نرم افزار قابل انعطاف است.

• **واقعیت:** با اطمینان میتوان گفت که نیازسنجی نرم افزار تغییر میکند، اما برخورد با این تغییر برحسب زمان بیان آن متفاوت خواهد بود.

•

داستان نرم افزار (ادامه)

• تاثیر تغییرات نیازهای کاربران نسبت به زمان، متفاوت است



• هر تغییر باعث افزایش ناگهانی درخواست برای منابع بیشتر و اصلاحات عمده، بنابراین افزایش هزینه را به دنبال دارد که این افزایش در مراحل (فازهای) مختلف تولید و توسعه نرم افزار متفاوت می باشد.

داستان نرم افزار (ادامه)

• داستانهای متخصصین

• پس از گذشت ۵۰ سال از بحران نرم افزار، هنوز یکسری متخصصین به داستان های زیر اعتقاد دارند.

• **داستان:** وقتی نرم افزار تولید و به مشتری تحویل داده شد، کارمان تمام شد.

• **واقعیت:** آمار نشان داده که ۶۰ تا ۸۰ درصد از کل تلاشهای صورت گرفته برای تولید نرم افزار، تازه بعد از آنکه به دست مشتری می رسد، انجام می شود.

• **داستان:** تا زمان اجرای نرم افزار، راهی برای بررسی کیفیت آن وجود ندارد.

• **واقعیت:** یکی از موثرترین مکانیزمهای تضمین کیفیت نرم افزار، که میتوان از شروع پروژه بکار گرفت، **بازبینی فنی رسمی** است که بعنوان فیلترهای کیفیتی محسوب میشود و موثرتر از تست نرم افزار برای یافتن گروه های خاصی از خطاها و نقایص نرم افزار می باشد.

انواع نرم افزار

- مستقیماً با سخت افزار در ارتباط اند؛ سیستم عامل ، کامپایلرها
- مجموعه‌ای از برنامه‌هایی که برای دادن سرویس به برنامه‌های دیگر نوشته شده‌اند.

سیستمی

- انجام امور علمی، فنی و مهندسی؛ **Mathimatica, CAD**
- این نرم‌افزارها الگوریتم‌های خاصی را روی اعداد اجراء می‌کنند.

علمی

- عملاً متکی به یک پایگاه داده می‌باشند. (پردازش تراکنش بانکی)
- برنامه مستقل هستند که برای حل یک نیاز شغلی مورد استفاده قرار می‌گیرند.

کاربردی و تجاری

- این گونه نرم‌افزارها از الگوریتم‌های غیر عددی برای حل مسائل استفاده می‌نمایند.
- سیستم‌های خبره، تشخیص الگو، شبکه عصبی مصنوعی و ...

هوش مصنوعی

- صفحات وب که با استفاده از مرورگرها بازیابی می‌شوند.
- این نرم‌افزارها بر روی کامپیوترهایی قرار دارند که به شبکه‌ها متصل هستند.

وب

• ...

و

نیازمندیها (Requirement)

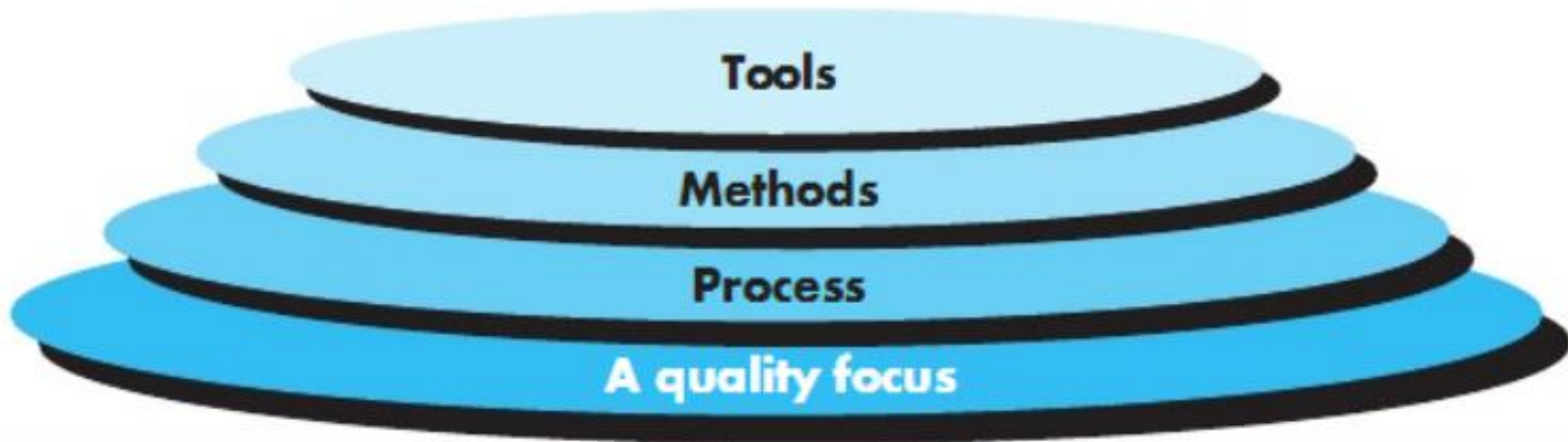
- **Functional (عملیاتی – کارکردی):**
 - ماهیتی کسب و کار دارد. در واقع به کارکردهایی از Business که بایستی در یک نرم افزار Automate گردد.
 - تصویری از **فرآیندهای کسب و کار** سازمان
 - مثلاً در یک نرم افزار مالی توانایی محاسبه تراز مالی ، سند زنی و...
 - در یک نرم افزار آموزشی امکان ثبت نام، انتخاب واحد، حذف اضافه، صدور کارنامه ،

نیازمندیها (Requirement) ادامه

• Non-functional (غیر عملیاتی – غیر کارکردی):

- این نیازمندیها به خصوصیات کیفی و استانداردها مانند امنیت (Security)، کارایی (Performance) دسترس پذیری، زمان پاسخ.
- مستقیماً به کسب و کار مربوط نیست و تعدادش از یک نرم افزار به نرم افزار دیگر متفاوت است.

Software engineering is a layered technology



لایه تاکید بر کیفیت

- این لایه نشان می دهد که یک محصول نرم افزاری باید از یک سری استاندارد که متضمن کیفیت (Quality) بالای محصول است تبعیت میکند. بدیهی است که هیچ شرکت نرم افزاری دوست ندارد نرم افزار بی کیفیت تحویل مشتری دهد.

لایه تاکید بر کیفیت

- کیفیت نرم افزار به **دو** رده مرتبط اما مجزای زیر اشاره دارد:

(۱) کیفیت عملیاتی نرم افزار (Software Functional Quality):

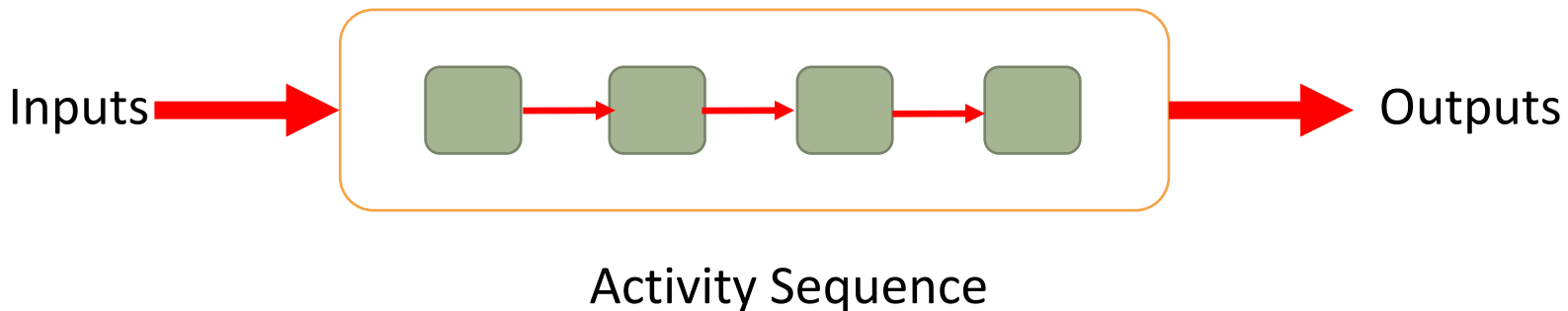
شاخصی جهت نشان دادن میزان تطابق نرم افزار با **نیامندی های عملیاتی** تعریف شده برای نرم افزار است.

(۲) کیفیت ساختاری نرم افزار (Software Structural Quality):

که منعکس کننده میزان دست یابی به نیازمندی های **غیر عملیاتی** مانند استحکام (Robustness) و قابلیت نگهداری (Maintainability) نرم افزار است.

لایه فرآیندها

- اساس مهندسی نرم افزار فرآیند می باشد. لایه فرآیند مانند چسبی است که لایه های فناوری را با هم نگه داشته.
- این لایه خود در برگیرنده **لایه تکنولوژی های** که در طی زمان توسعه نرم افزار را فراهم می کند.
- تعریف فرآیند: **دنباله ای** (نه مجموعه) از فعالیت ها که برای یک هدف خاص تعریف می شود



فرآیند توسعه نرم افزار Software Development Process

- لایه فرآیند یک **چارچوبی** را برای تحویل موثر نرم افزار مهندسی تعریف می کند.

- این لایه اساس کنترل مدیریت پروژه های نرم افزاری را شکل می دهد و مفادی را که در آن روشهای فنی (Methods) اعمال میشوند را ایجاد کرده و محصولات کاری (مدل ها، اسناد، داده ها، گزارش ها، فرم ها و غیره) تولید می شوند،
 - نقاط قوت ایجاد می شوند، کیفیت تضمین شده است، و تغییرات را به درستی مدیریت میکند.

یک چارچوب فرآیند عمومی شامل پنج فعالیت است

- ۱. ارتباطات. قبل از شروع هر کار فنی، برقراری ارتباط و همکاری با مشتری (و سایر ذینفعان) بسیار مهم است. هدف درک اهداف ذینفعان برای پروژه و جمع آوری نیازهایی است که به تعریف ویژگی‌ها و عملکردهای نرم افزار کمک می‌کند. (جمع آوری نیازمندی‌ها، امکان سنجی)
- ۲. برنامه ریزی: کار مهندسی نرم افزار را با تشریح وظایف فنی که باید انجام شود، خطرات احتمالی، منابعی که مورد نیاز خواهد بود، محصولات کاری که باید تولید شود و برنامه کاری را تعریف می‌کند.
- ۳. مدل سازی. یک مهندس نرم افزار با ایجاد مدل‌هایی برای درک بهتر نیازمندی‌های نرم افزار و طراحی که به آن نیازها می‌رسد انجام می‌دهد.

یک چارچوب فرآیند عمومی شامل پنج فعالیت است (ادامه)

- ۴. ساخت و ساز. آنچه طراحی می کنید باید ساخته شود. این فعالیت ترکیبی از تولید کد (چه دستی یا خودکار) و آزمایشی است که برای کشف خطاها در کد انجام می شود.
- ۵. توسعه: (به عنوان یک موجودیت کامل یا به عنوان یک افزایش جزئی تکمیل شده) به مشتری تحویل داده می شود که توسط او ارزیابی شود و بر اساس ارزیابی بازخورد ارائه کند.

فعالیت‌های چتری

- فرآیندهای مهندسی نرم‌افزار با مقداری از فعالیت‌های چتری تکمیل می‌شود که عبارتند از:
 ۱. **ردیابی و کنترل پروژه نرم افزار:** ارزیابی پیشرفت در برابر طرح پروژه و حفظ برنامه زمانبندی.
 ۲. **مدیریت ریسک:** ارزیابی ریسک‌هایی که ممکن است بر نتیجه پروژه یا کیفیت محصول تأثیر بگذارد.
 ۳. **تضمین کیفیت نرم افزار:** تعریف و انجام فعالیت‌هایی (آزمون‌هایی) برای اطمینان از کیفیت نرم افزار.
 ۴. **بررسی‌های فنی:** محصولات کاری مهندسی نرم افزار را برای کشف و حذف خطاها ارزیابی کنید.

فعالیت‌های چتری (ادامه)

- ۵. **اندازه‌گیری:** تعریف و جمع‌آوری نشان‌گرهای کارایی شامل (زمان لود، نرخ شکست، نرخ محاوره با کاربر) که به ارائه نرم افزار به نیازهای ذینفعان کمک می‌کند.
- ۶. **مدیریت پیکربندی نرم افزار:** اثرات تغییر را در طول فرآیند نرم افزار مدیریت کنید.
- ۷. **مدیریت قابلیت استفاده مجدد:** معیارهایی را برای استفاده مجدد از محصول کاری تعریف کنید و مکانیسم‌هایی را برای اجزای قابل استفاده مجدد ایجاد کنید.
- ۸. **آماده‌سازی و تولید محصول کاری:** انجام فعالیت‌های مورد نیاز برای ایجاد محصولات کاری مانند مدل‌ها، اسناد، گزارش‌ها، فرم‌ها و لیست‌ها.

"انطباق پذیری فرآیند مهندسی نرم افزار"

- - این فرآیندها باید سازگار چابک و سازگار با مشکل، پروژه، تیم و سازمان باشد و وحی منزل نیست.
- - یک فرآیند برای یک پروژه ممکن است به طور قابل توجهی با پروژه دیگر متفاوت باشد. تفاوت ها می تواند شامل موارد زیر باشد:
 - - جریان فعالیت ها، اقدامات، وظایف و وابستگی های متقابل آنها
 - - چگونه اقدامات و وظایف در هر چارچوب فعالیت تعریف می شود
 - - شناسایی و نیاز محصولات کاری
 - - بکارگیری فعالیت های تضمین کیفیت، کاربرد فعالیت های ردیابی و کنترل پروژه
- - جزئیات و دقت در شرح فرآیند و مشارکت مشتری و سایر ذینفعان
- - استقلال داده شده به تیم نرم افزار و تجویز سازماندهی و نقش های

تیم

لایه ی روش‌ها

- برای توسعه ی هر محصول نرم افزاری یک روش پنج مرحله ای وجود دارد که عبارتند از:
 - تجزیه و تحلیل خواسته ها Requirements Analysis
 - طراحی Design
 - پیاده سازی Implementation
 - آزمون Test
 - پشتیبانی Maintenance

تفاوت لایه روش‌ها و فرآیند

- فرآیند نرم افزار مجموعه ای از مراحل یا مجموعه ای از فعالیت هایی است که در طول توسعه نرم افزار استفاده می شود. برای مثال، اگر می‌خواهیم نرم‌افزار را با موفقیت توسعه دهیم، باید مجموعه‌ای از قوانین یا فعالیت‌هایی مانند برنامه‌ریزی، جمع‌آوری نیازمندی‌ها، طراحی، پیاده‌سازی، آزمایش و استقرار را دنبال کنیم.
- لایه روش‌ها: این لایه دانش فنی را برای توسعه نرم افزار فراهم می کند که شامل روش‌ها، دانش فنی و "چگونگی" به منظور توسعه نرم افزار است. بیشتر بر روی مرحله خاص در طول توسعه تمرکز دارد.
 - متدولوژی‌های مختلفی مانند چابک، آبشار، روش مارپیچی و غیره وجود دارد.

نمونه‌های روش‌های توسعه نرم‌افزار

1. **آبشاری:** این یک رویکرد خطی و متوالی است که در آن هر مرحله از فرآیند توسعه به ترتیب خاصی انجام می‌شود.
 2. **چابک:** این یک رویکرد تکراری و تطبیقی است که پروژه به واحدهای کوچک کار یا «داستان‌های کاربر» تقسیم می‌شود.
 3. **اسکرام:** این زیرمجموعه‌ای از چابک است که کار را به صورت اسپرینت‌های با زمان محدود سازماندهی می‌کند.
- **توسعه سریع برنامه RAD:** این یک رویکرد سریع و انعطاف پذیر است که بر نمونه سازی سریع و تحویل تکراری تأکید دارد.
 - تیم وارد فازهای نمونه‌سازی سریع می‌شود.
 - در هر فاز، آن‌ها نمونه اولیه‌ای از بخشی از سیستم را می‌سازند، سپس آن را با ذینفعان بررسی می‌کنند تا بازخورد بگیرند. پس از نمونه‌سازی و تأیید تمام بخش‌های سیستم، تیم آن‌ها را در یک سیستم کامل ادغام می‌کند.

لایه ابزار

- ابزارهای مهندسی نرم افزار پشتیبانی خودکار یا نیمه خودکار را برای فرآیند و روش ها ارائه می دهند.
- این گونه نرم افزارها به مهندسين نرم افزار کمک می کنند تا بتوانند سیستم های مورد نیاز را مهندسی نمایند.
- ابزاریهای خودکار اصطلاحاً ابزارهای مهندسی نرم افزار به کمک کامپیوتر گفته می شود که ترجمه عبارت “Case Tools” می باشد.
- (Case) computer-aided software engineering
- یکی از معروفترین CASE Tools در دنیا فرآیند یکپارچه ی ارائه شده توسط شرکت Rational که RUP نامیده میشود.

ابزار مهندسی نرم افزار (ادامه)

مصاحبه های تحلیلی → Word

برای کد نویسی → V Stadoi ۲۰۱۰

برای مدل سازی → UML {
 Rational Rose
 My Visio

برنامه ریزی و زمان بندی → My Project

لایه ی روش‌ها – تجزیه و تحلیل خواسته ها

- سیستم سنتی که تا کنون حاکم بوده بررسی شود و نیازهای مشتری لیست شود.
- لیست تولید شده در جلسه ای که با مشتری برگزار خواهد شد مورد بازبینی با تیم توسعه قرار گیرد.
- در نیازسنجی باید به اجزا زیر بیشتر دقت کنیم:
 - ورودی ها و خروجی های سیستم
 - پردازشهای که روی آنها انجام میشود
 - رابط ها : انسان به محیط، انسان به انسان، و محیط با محیط
 - مستندات نیازها

لایه ی روش‌ها - طراحی

- در مرحله طراحی سعی میشود خواسته های به دست آمده از مرحله ی تحلیل به صورتی (با نمودارهای خاص) نمایش داده میشود که قبل از کدنویسی بتوان آن را مورد ارزیاب قرار داد.
- باید به چهار ویژگی توجه کرد:
 - ساختمان داده
 - معماری نرم افزار
 - نمایش واسطها
 - و جزئیات رویه ای (الگوریتم ها)

مثال : لایه ی روش ها - طراحی

1. طراحی معماری:

2. هدف: طراحی معماری اجزای اصلی سیستم، مسئولیت ها، ویژگی ها و تعاملات آنها را مشخص می کند. این یک انتخاب ساختاری سطح بالا است.

1. سیستم را به اجزای اصلی تجزیه کنید.

2. مسئولیت های عملکردی را به این اجزا اختصاص دهید.

3. رابط های کامپوننت را تعریف کنید.

4. مقیاس پذیری، عملکرد، قابلیت اطمینان و ویژگی های مصرف منابع را در نظر بگیرید.

5. ارتباط و تعامل بین اجزا را نشان می دهد.

3. مثال: سامانه کتابفروشی آنلاین:

1. ما در حال طراحی یک سیستم کتابفروشی آنلاین هستیم که به کاربران امکان می دهد کتاب ها را مرور، جستجو و خریداری کنند. تصمیمات معماری ما نحوه عملکرد سیستم را شکل می دهد.

2. استفاده از معماری ۳ لایه - لایه نمایش - لایه منطقی - لایه دسترسی به دیتابیس

لایه ی روش‌ها - آزمون

- در مرحله آزمون سعی میشود
- **منطق اجزای داخلی نرم افزار مورد آزمایش** قرار گیرد (آزمودن تمام دستورات برنامه)
- **منطق اجزای خارجی** عملیاتی برنامه مورد آزمایش قرار گیرد. (حصول اطمینان از اینکه ورودی های نرم افزار منجر به ایجاد خروجی های درستی میشود.
- انواع آزمون: تست واحد، تست یکپارچگی، آزمون پذیرش، آزمون کارایی، آزمون امنیتی، آزمون کاربردپذیری

Online Bookstore System: Testing

1. Unit Testing: آزمون واحد

- تک تک اجزا (واحد‌ها) را به صورت مجزا بررسی کنید.
- مثال: الگوریتم جستجوی کتاب را برای نتایج دقیق آزمایش کنید.

2. Integration Testing: آزمون یکپارچگی

- اعتبار سنجی تعاملات بین اجزای مختلف
- مثال: اطمینان حاصل کنید که افزودن کتاب به سبد خرید پایگاه داده را به درستی به روز می‌کند.

3. Acceptance Testing: آزمون پذیرش

- اعتبارسنجی سیستم در برابر الزامات تجاری
- مثال: فرآیند خرید نهایی (جستجو، سبد خرید، پرداخت) را آزمایش کنید.

Online Bookstore System: Testing

4. Performance Testing: آزمون کارایی

- پاسخگویی، مقیاس پذیری و استفاده از منابع را ارزیابی کنید.
- مثال: زمان پاسخگویی را در اوج ترافیک اندازه گیری کنید..

5. Security Testing: آزمون امنیت

- آسیب پذیری ها را شناسایی کنید و از داده ها محافظت کنید.
- مثال: اطمینان از انتقال ایمن اطلاعات حساس.

6. Usability Testing: آزمون استفاده پذیری

- کاربر پسندی را ارزیابی کنید.
- مثال: سهولت پیمایش و جریان خرید را ارزیابی کنید.

لایه ی روش‌ها - پشتیبانی

- شامل یک سری تغییرات است. این تغییرات ۴ نوع است.
- تغییرات **تصحیحی**: معیابی که مشتری گزارش کرده رفع شده نرم افزار تصحیح میشود.
- تغییرات **تطابقی**: اصلاحاتی روی نرم افزار انجام میشود که آن را با تغییرات خارجی خود سازگار میکند. بیشتر بخاطر تغییر محیط نرم افزار به مرور زمان است (CPU و سیستم عامل)
- تغییرات **بهبود دهنده**: به موازات استفاده مشتری متوجه یک سری تغییراتی برای بهبود نرم افزار میشود.
- تغییرات **پیشگیرانه**: تغییراتی به نرم افزار اعمال میشود که بتوان سه تغییر بالایی را آسان تر به نرم افزار اعمال کرد. به این تغییرات مهندسی مجدد نرم افزار می گویند.

مثال: لایه‌ی روش‌ها – پشتیبانی

- تعمیر و نگهداری نرم افزار برای کتابفروشی آنلاین
 - ۱. تعمیر و نگهداری اصلاحی:
 - - خطاها و باگ‌ها را به سرعت برطرف کنید.
 - - مثال: رفع عدم دقت الگوریتم جستجو.
- ۲. نگهداری پیشگیرانه:
 - - به طور فعال عمر طولانی را تضمین کنید.
 - - مثال: درگاه‌های پرداخت قدیمی را ارتقا دهید.
- ۳. تعمیر و نگهداری بهبود دهنده:
 - - بهبود ویژگی‌ها و عملکرد.
 - - مثال: بهبود دقت توصیه کتاب. - جستجوی فازی در بخش جستجو کتاب‌ها
- ۴. تعمیر و نگهداری تطبیقی:
 - - سازگاری با تغییرات (به عنوان مثال، قوانین مالیاتی).
 - - مثال: به روز رسانی ماژول قیمت گذاری.

اصول کلی مهندسی نرم افزار

- ۱. ارزش افزوده. قبل از اینکه چیزی را در نیازمندی‌های سیستم مشخص کنید، این سوال را بپرس آیا ارزش افزوده به سیستم دارد؟ مثال ارائه ویژگی سفارشی سازی رنگ نرم افزار؟
- ۲. "KISS (Keep It Simple, Stupid!) طراحی باید تا حد امکان ساده باشد، اما نه ساده تر به معنی کاستن از فیچرها. این منجر به نرم افزارهایی با قابلیت نگهداری بیشتر و خطای کمتری می شود.
- ۳. "حفظ چشم انداز": چشم انداز روشن برای موفقیت یک پروژه نرم افزاری ضروری است. یک معمار توانمند که می تواند چشم انداز را حفظ کند و انطباق را به اجرا بگذارد، یک پروژه نرم افزاری موفق را تضمین می کند.

تمرین مهندسی نرم افزار به عنوان یک کل (ادامه)

- ۴. "آنچه شما تولید می کنید، دیگران مصرف می کنند": همیشه مشخص کنید، طراحی کنید، مستند کنید، و پیاده سازی کنید و بدانید که شخص دیگری باید بفهمد چه کاری انجام می دهید.
- ۵. «به سوی آینده باز باشید»: سیستم ها باید آماده سازگاری با تغییرات باشند. هرگز خود را در گوشه ای طراحی نکنید. همیشه بپرسید «اگر چه می شد» و برای همه پاسخ های ممکن آماده شوید.
- ۶. **استفاده مجدد باعث صرفه جویی** در زمان و تلاش می شود. برنامه ریزی از قبل برای استفاده مجدد هزینه را کاهش می دهد و ارزش اجزای قابل استفاده مجدد و سیستم هایی که در آنها گنجانده شده اند را افزایش می دهد.
- ۷. «**فکر کنید!**»: قرار دادن فکر واضح و کامل قبل از عمل تقریباً همیشه نتایج بهتری به همراه دارد. وقتی فکر روشنی وارد سیستم شد، ارزش آشکار می شود.

پایان بخش اول •