Software Engineering as a Standard

مهندسی نرم افزار

مستندات مورد نیاز پروسه توسعه، تحویل و نگهداری نرم افزار /سایت

تهیه کننده : خشایار جام سحر [1]

jamsahar@gmail.com

نسخه ۲

۱۴۰۲/۱۰/۰۴

2023, Dec 25

[1] - https://www.linkedin.com/in/khashayar-jamsahar-58289743/

فهرست عناوین

لایسنس/مجوز

سند حاضر شامل کلیات/سرفصلهای مهندسی نرم افزار بوده و شامل مستندات لازم برای طراحی، توسعه، تحویل و نگهداری نرم افزار است. این سند تحت لایسنس GNU GPL V3[2] قرار داشته و از آدرس https://github.com/jamsahar/Documents قابل دانلود است.

مقدمه

مهندسی نرم افزار، روشی سیستماتیک، نظام مند و استاندارد برای طراحی و توسعه نرم افزار می باشد. **مهندسی نرم افزار تکنولوژی است** مشتمل بر ۴ لایه زیر:

۱) کیفی(عملکردی)(Non Functional)

a. CIA triad

i. محرمانگی Confidentiality

ii. یکپارچگی/انسجام Integrity[3]

iii. دسترس پذیری Availability[4]

[2] - https://www.gnu.org/licenses/gpl-3.0.html

[3] - Integrity is a nonfunctional aspect of a system to be ***safe, complete, consistent, correct, and free of corruption and errors***.

[4] - **Readiness for correct service**

b. قابل نگهداری Maintainability[5]
   i. انعطاف پذیر Flexiblity[6] و سازگار پذیر Adaptability[7]
   ii. توسعه پذیر Extensibility[8]
   iii. ماجولار Modularity[9]
   iv. مقیاس پذیر Scalability[10]
   v. آزمون پذیری Testability
   vi. پایداری Stability
   vii. تحلیل پذیری Analyzability
c. کارا Efficiency[11]
d. Completeness[12]
e. Portability[13] & Multiplatform
f. Interoperability[14]
g. Reusability[15]
h. پایائی Sustainability[16]
i. مقاومت و تاب آوری Resiliency[17]
j. Reliability[18] Models
   i. Basic Execution Time Model
   ii. Jelinski and Moranda Model
      ۱. Sukert Modified Schick-Wolverton Model
      ۲. Lipow Modified Version of Jelinski-Moranda Geometric Model
      ۳. Schick Wolverton Model
      ۴. GO-Imperfect Debugging Model
      ۵. Jelinski-Moranda Geometric Model
      ۶. Little-Verrall Bayesian Model
      ۷. Shanthikumar General Markov Model
      ۸. Error Detection Model for Application during Software Development
      ۹. The Langberg Singpurwalla Model
      ۱۰. Jewell Bayesian Software Reliability Model

[5] - To undergo modifications and repairs. The software should be simple for the development team to maintain. the capability to be modified for purposes of making corrections, improvements, or adaptations.
[6] - The software should be adaptable to changes. Able to modify on changing needs.
[7] - Adaptation of software system is almost an inevitable process due to the change in customer requirements, the need for faster development of new or maintenance of existing software systems, etc.
[8] - There should be no difficulty in growing the number of functions performed by the software.
[9] - A software product has high modularity if it can be separated into separate independent sections and modified and tested independently.
[10] - It entails the software's capacity to be easily upgraded.
[11] - The software must use storage space and time wisely.
[12] - The design should have all components like data structures, modules, and external interfaces, etc.
[13] - Software portability is the capacity to use the same software in different contexts. The key purpose for porting for sales teams is to reach a larger audience. There are different hardware and software platforms available. More users equal more profit.
[14] - The capacity of different solutions to freely and readily communicate with one another is referred to as software interoperability.
[15] - Software reusability is a property that refers to a software component's predicted reuse potential. Software reuse not only boosts productivity but also enhances the quality and maintainability of software products.
[16] - Apply best practices in architecture, design, construction, and operations of product to reduce impacts.
[17] - Ability to recover quickly from an failure/problem/issue/disaster. Applications can automatically recover from failures and handle increased traffic with the help of load balancers and auto-scaling.
[18] - **Continuity of correct service.** Means that the software should not fail during execution and be free of flaws. Software reliability is defined as a system's or component's ability to perform its required functions under static conditions for a set amount of time.
https://www.javatpoint.com/software-engineering-software-reliability-models

۱۱. Quantum Modification to the JM Model
۱۲. Optimal Software Released Based on Markovian Software Reliability Model
۱۳. A Modification to the Jelinski-Moranda Software Reliability Growth Model Based on Cloud Model Theory
۱۴. Modified JM Model with imperfect Debugging Phenomenon

iii. Goel-Okumoto (GO) Model
iv. Musa-Okumoto Logarithmic Model

k. آزمون پذیری[19]

i. امنیت Safety[20] & Security[21]

ii. قابلیت استفاده Usability[22]

۱. Understandability
۲. Learnability
۳. Operability

iii. یکپارچگی Consistency

iv. میزان انطباق فرآیندهای کارکردی با نیازمندیهای کاربران نهائی

v. اندازه گیری متریکها[23] (Proces, Project, Product) و میزان تاثیر عناصر بر یکدیگر

- استاتیک/داخلی

a. LOC (Line of code)/Person/Time[24]
i. KLOC- Thousand lines of code
ii. NLOC- Non-comment lines of code
iii. KDSI- Thousands of delivered source instruction
b. Number of entities in ER diagram
c. Total number of processes in detailed data flow diagram
d. Function Point Analysis/Count
i. Count the number of functions of each proposed type.
ii. Compute the Unadjusted Function Points (UFP).
iii. Find the Total Degree of Influence (TDI).
iv. Compute Value Adjustment Factor (VAF).
v. Find the Function Point Count (FPC).
e. Halstead's Software Metrics
i. Program length
ii. Vocabulary size
iii. Program volume
iv. Program level
v. Program difficulty
vi. Program effort
vii. Time to implement
viii. Language Level
ix. Intelligence Content
x. Programming Time

[19] - The software should be simple to test.
https://en.wikipedia.org/wiki/Software_testing
https://en.wikipedia.org/wiki/Software_quality_control
https://en.wikipedia.org/wiki/International_Software_Testing_Qualifications_Board https://www.istqb.org/
[20] - The software developed should not be detrimental to the environment or life.
[21] - The software should protect the data from external threats.
[22] - The program should be simple enough for anyone to use.
[23] - Professors Robert Kaplan and David Norton once said that "if you can't measure it, you can't manage it."
https://en.wikipedia.org/wiki/Software_metric
https://www.scaler.com/topics/software-engineering/software-metrics-in-software-engineering/
[24] - Numbers ranging from ~2K-8K LOC/person/year

f. Operational reliability[25]
i. Mean Time to Failure (MTTF)
ii. Mean Time to Repair (MTTR)
iii. Mean Time Between Failure (MTBF)
iv. Mean Up Time (MUT)
v. Mean Down Time (MDT)
vi. Rate of occurrence of failure (ROCOF)
vii. Probability of Failure on Demand (POFOD)
viii. Availability (AVAIL)
ix. Maturity
x. Recoverability
xi. Software Fault Tolerance
۱. Recovery Block
۲. N-Version Software
۳. N-Version Software and Recovery Blocks

- دینامیک/هیبرید

a. Cyclomatic Complexity
b. extent of class usage,
c. Dynamic Coupling[26]
d. Dynamic Lack of Cohesion[27]
e. اندازه گیری complexity[28] و میزان تاثیر گذاری آن بر productivity محصول

۲) پروسه (کارکردی)(Functional)

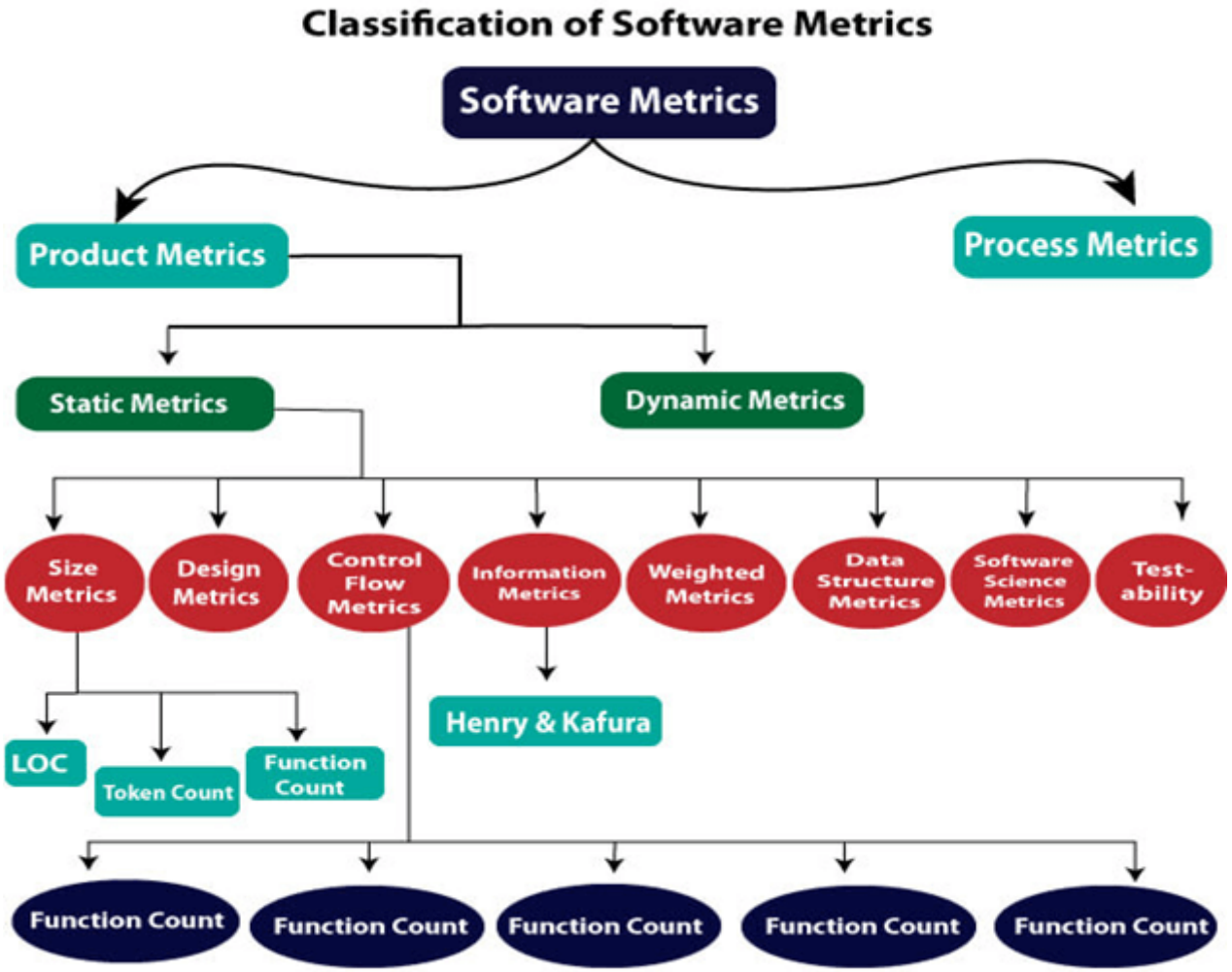
a. **کار درست چیست؟** [۲۹]: طراحی درست برای انجام وظایف (Correctness[30])
b. **چگونگی انجام کار؟** [۳۱]: درستی و صحت پروسه ها، وظایف و فعالیتها
c. میزان انسجام/هماهنگی/انطباق (Consistency) طراحی و انطباق پذیری روشها/متدها با فرآیندهای کارکردی

۳) روشها (متدها)

a. پاسخهایی که برای چگونگی انجام درست هر کاری لازم است
b. ایجاد و بکارگیری تکنیک های لازم برای اجرای موارد فوق
c. توسعه مجموعه ای از ارتباطات، نیازمندیها، تحلیلها، مدلها، ساختارها، تستها، سرویسها، شاخصها، متریکهای اندازه گیری و ...

۴) ابزارها : جهت مکانیزه نمودن موارد فوق

[25] - Software reliability is also defined as the probability that a software system fulfills its assigned task in a given environment for a predefined number of input cases, assuming that the hardware and the input are free of error.
[26] - https://en.wikipedia.org/wiki/Coupling_(computer_programming)
[27] - https://en.wikipedia.org/wiki/Cohesion_(computer_science)
[28] - Cyclomatic complexity (McCabe) M = E – N + 2P https://en.wikipedia.org/wiki/Cyclomatic_complexity
[29] - **Validation:** Are we building the right product?
[30] - The software should match all of the customer's needs. Software design should be correct as per requirement.
[31] - **Verification:** Are we building the product, right?



فعالیتهای مهندسی نرم افزار

فعالیت های توسعه نرم افزار شامل موارد زیر می باشد:

۱) شناسایی نیاز
۲) برنامه ریزی توسعه نرم افزاری
۳) طراحی نرم افزار
۴) پیاده سازی، آزمایش و مستندسازی
۵) گسترش/توسعه و نگهداری نرم افزار

مهندسی نرم افزار[۳۲] به طور کلی به سه فاز تقسیم بندی می شود:

فاز تعریف:

بر چیستی تاکید دارد. چه اطلاعاتی باید پردازش شود، کدام کارایی مطلوب است، چه رفتار های سیستمی قابل انتظار است، چه رابطه هایی را می توان برقرار کرد، چه محدودیت هایی وجود دارد و بطور کلی خواسته های کلیدی سیستم شناسایی می شود. سه کار عمده در این فاز شامل :

۱) مهندسی اطلاعات یا سیستم
۲) طرح ریزی پروژه نرم افزار
۳) تحلیل خواسته ها Software requirements

[32] – مهندسی نرم‌افزار، نحوه نوشتن راه حل نیست، بلکه شناسایی آنچه باید در راه حل باشد است. نگاه کنید به لینکهای زیر:

- https://en.wikipedia.org/wiki/Software_engineering
- https://en.wikipedia.org/wiki/Software_architecture

است.

فاز توسعه Software development :

بر چگونگی تاکید دارد. داده ها چه ساختاری داشته باشند، عملیات درون معماری چگونه پیاده سازی می شوند، جزییات روال ها، ویژگی های واسط ها، زبان برنامه نویسی، نحوه آزمایش ها.

سه کار عمده در این فاز شامل :

۱) طراحی نرم افزار Software design
۲) تولید سورس کد Software development
۳) آزمایش نرم افزار Software testing

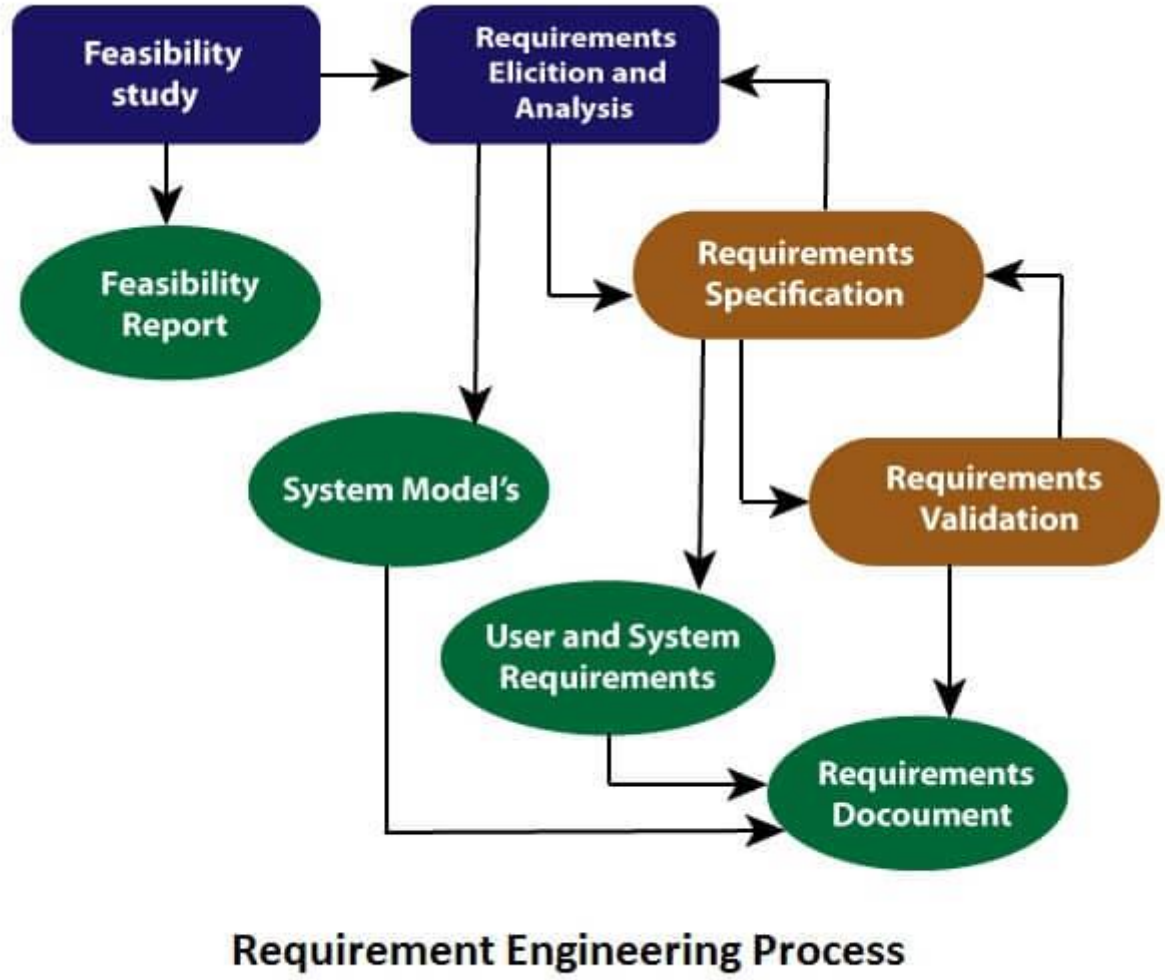
فاز پشتیبانی Software maintenance :

بر تغییراتی تاکید دارد که با تصحیحات مورد نیاز در جهت تکامل محیط نرم افزار در ارتباط هستند. همچنین نیز تغییراتی که ناشی از تغییر خواسته های مشتریان/کاربران هستند.

بیزینس کانسپت

۱) ایده اولیه و نحوه Promoting and Monetizing
۲) بیزینس پلن
۳) سند نیازمندیها: Software Requirement Specification (SRS)
   a. نیاز مندیهای غیرکارکردی
   b. نیاز مندیهای کارکردی کاربران/مشتریان Functional Requirement Specification (FRS)
      i. درک دامنه بیزینس
      ii. تشخیص مشکلات و مسائل موجود
      iii. ارزیابی
      iv. طبقه بندیها
      v. الویتها
      vi. صحت/اعتبار سنجی
      vii. تحلیل سازگاری، رفع ابهام و تناقضات
      viii. مدلسازی فرآیندهای کسب و کار BPMN
      ix. معماری اطلاعات و داده مورد نیاز
      x. Data Flow Diagrams[33]
      xi. Data Dictionaries
      xii. Entity Relationship Diagrams
   c. نیازمندیهای تکنولوژی
   d. نیازمندیهای بازار و کسب و کار
   e. نیازمندیهای سازمان تنظیم مقررات بیزینس مربوطه
۴) انواع نقش ها و کاربران (Admin, Operator, Users, …)
۵) قرارداد(بندها، تبصره ها، پیوستها، فورس ماژور، جرائم و پنالتی ها، ...)

[33]- https://www.scaler.com/topics/software-engineering/data-flow-diagrams/



Requirement Engineering Process

سازمان پروژه

۱) ترکیب نیروهای سازمان پروژه
۲) زمان بندی(شکست) پروژه
۳) نحوه مدیریت ریسک

چک لیست امکانات، نیازمندیها و فرآیندها

۱) سند امکانات[34] و نیازمندیها

a. کارکردی(چه کاری انجام شود(فعل)) Functional
b. غیر کارکردی(چگونه آن کار انجام شود(قید)) NonFunctional
i. تکنولوژی فنی

۱. Responsive web applications
۲. Single-page web applications
۳. Progressive web applications
۴. **Real-time[35] web applications**

[34] – A feature is a set of logically related requirements that allows the user to satisfy an objective. A feature tends to be a higher-level objective than a requirement.
[35] - A real-time web application delivers responses to events in a measurable and acceptable time period, depending on the nature of the event, by means of asynchronous, bidirectional communication between the client and the server. WebSocket is the core

۵. **Reactive[36] web applications**
۶. Mashups and web services

i. اطلاعاتی
ii. عملکردی/کارائی
iii. محیطی(UI/UX)
iv. امنیتی
v. کیفی
vi. مقیاس پذیری
vii. درستی/یکپارچگی integrity

۲) سند فرآیندهای کسب و کار

a. ورودیها
b. پروسه ، گردش کار
c. خروجیها

۳) Workflow

فاز طراحی نرم افزار

۱) **سند معماری(اطلاعات) سازمانی**[۳۷] : پیشتر توسط سازمان بایستی تهیه شده باشد و کلیه نرم افزارها بر اساس آن طراحی و توسعه داده می شود. این سند شامل معماری و طراحی فرآیندها، ساختار داده ها و جریان اطلاعات بین واحدها می باشد.

۲) سند استانداردهای پایه ای تولید و توسعه نرم افزار[۳۸]:

a. استاندارد سند توصیف متدولوژی MDD
b. استاندارد طرح مدیریت پروژه PMP
c. استاندارد طرح تضمین کیفیت QAP
d. استاندارد طرح مدیریت پیکربندی CMP
e. استاندارد طرح تصدیق و صحه گذاری V&V
f. استاندارد طرح آزمون نرم افزار
g. استاندارد طرح انتقال و تحویل نرم افزار
h. استاندارد طرح ضمانت نرم افزار
i. استاندارد طرح نظارت

۳) معماری سیستم:

a. Blackboard
b. Big Ball of Mud Antipattern (Coupling everythings)
c. Centralized, Decentralized (Distributed)
d. Client-server (2-tier, 3-tier, n-tier, cloud computing exhibits this style)
e. Component-based

technology employed for developing real-time web applications. WebSocket provides a full-duplex and bidirectional communication protocol over a single TCP connection.

[36] - **Asynchronous, Event-based, Non-blocking Architecture**

[37] – IT Enterprise Architecture.

[38] – مستندات کاملتر در سایت **شرکت مهندسی نرم افزار گلستان** http://golsoft.com/pages/fa/fa-namatan.html **نماتن** .
ایضاً http://old.irannsr.org/web_directory/ کتابخانه اسناد **سازمان نظام صنفی رایانه ای کشور**

f. Controller Responder Pattern (Master/Slave)
g. Data-centric
h. Event-driven (or implicit invocation)
i. Layered (or multilayered) architecture
j. Microkernel (Core) + Plug-ins architecture
k. Microservices architecture[39]
l. Monolithic application (Highly coupled)
m. Model-view-controller (MVC)
n. Multi-Tier Pattern (Application, API, Product Services, Common Services, Data)
o. Orchestration-Driven Service-Oriented architecture
p. Peer-to-peer (P2P)
q. Pipelines and filters
r. Ports and Adapters Pattern
s. Publish-Subscribe[40] messaging pattern
t. Reactive architecture
u. Representational state transfer (REST)
v. Rule-based
w. Serverless[41] Architectures (BaaS (Backend as a Service) and FaaS (Function as a Service))
x. Shared-Data pattern
y. Shared nothing architecture
z. Space-based architecture

Pattern-analysis summary

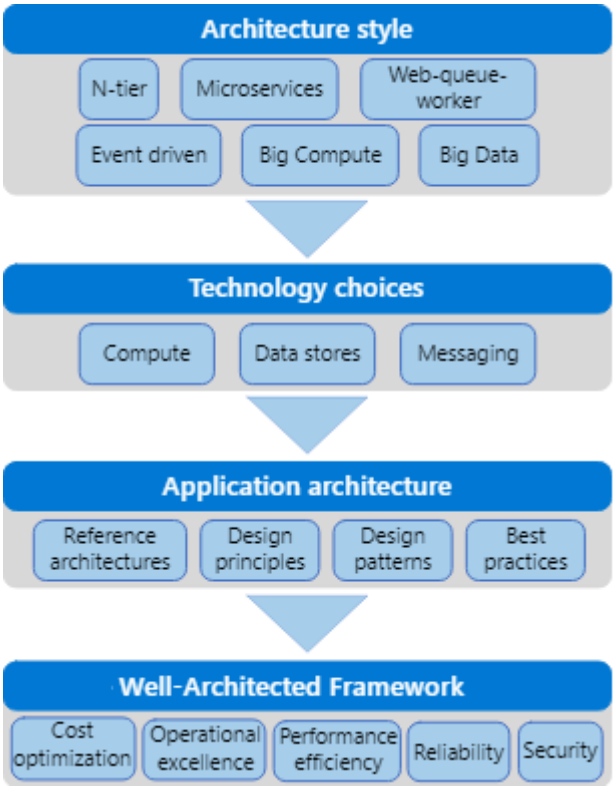
	Layered	Event-driven	Microkernel	Microservices	Space-based
Overall Agility	↓	↑	↑	↑	↑
Deployment	↓	↑	↑	↑	↑
Testability	↑	↓	↑	↑	↓
Performance	↓	↑	↑	↓	↑
Scalability	↓	↑	↓	↑	↑
Development	↑	↓	↓	↑	↓

۴) انتخاب پلاتفرم، تکنولوژیها و امکانات مرتبط:

a. Cloud Native[42]
b. Traditional Web (browser-based applications)
c. Desktop
d. Mobile

[39] - https://microservices.io , https://en.wikipedia.org/wiki/Microservices
[40] - https://en.wikipedia.org/wiki/Publish-subscribe_pattern
https://learn.microsoft.com/en-us/azure/architecture/patterns/publisher-subscriber
[41] - Applications that significantly or primarily depend on third-party applications and/or services in "the cloud" are called **BaaS (Backend as a Service)**.
[42] - Cloud native refers to a set of practices and technologies that let you build and run scalable applications in the cloud. https://www.cncf.io/ Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

Traditional on-premises	Modern cloud[43]
▪ Monolithic ▪ Designed for predictable scalability ▪ Relational database ▪ Synchronized processing ▪ Design to avoid failures (MTBF) ▪ Occasional large updates ▪ Manual management ▪ Snowflake servers	▪ Decomposed ▪ Designed for elastic scale ▪ Polyglot persistence (mix of storage technologies) ▪ Asynchronous processing ▪ Design for failure (MTTR) ▪ Frequent small updates ▪ Automated self-management ▪ Immutable infrastructure



۵) طراحی بر پایه

a. Web Application
   i. database first
   ii. code first
   iii. schema first
   iv. model first
b. Container-first and cloud-native[44]
   i. PaaS (platform as a service)[45]
   ii. IaaS (infrastructure as a service)
   iii. SaaS (software as a service)

[43] - https://learn.microsoft.com/en-us/azure/architecture/guide
https://cloud.google.com/discover

[44] - https://cloud.google.com/learn/what-is-cloud-native

[45] - https://cloud.google.com/learn/paas-vs-iaas-vs-saas

iv. CaaS (containers as a service)

۶) معیارهای انتخاب تکنولوژی ساخت

1) Ability to have separated modules to split logic
2) Interoperability, modules could extend one another
3) Inversion of Control / Dependency Injection
4) Type-safety everywhere
5) Asynchronous & Type-safe Event Management
6) Easy integration with DB Technology (incl. data validation)
7) Isomorphism to have the same concepts in Node, Deno, Browser, ...
8) **Real-time data solutions (vs content-centric projects)**
    - Apps that require high-frequency updates from the server:
        - Gaming
        - Social networks
        - Voting
        - Auctions
        - GPS apps
    - Dashboards and monitoring apps:
        - Company dashboards
        - Live maps
        - Instant sales updates
        - Travel alerts
        - Continuous integration/continuous delivery (CI/CD) pipeline pages
    - Collaborative and multi-user interactive apps:
        - Whiteboard apps
        - Team meeting apps
        - Document sharing apps
        - Visual Studio Live Share
    - Apps that require instant notifications:
        - Email apps
        - Chat apps
        - Turn-based games
        - Time series reporting
        - GitHub Actions, issue and pull request systems

۷) انتخاب پارادایم برنامه سازی[۴۶]

(رجوع شود به [47] **"Peter Van Roy (2009-05-12). "Programming Paradigms: What Every Programmer Should Know**)

۸) ساخت بر پایه[۴۸]:

a. Pure Programming Languages

- Compilers
    1. C/C++
    2. **Rust**
    3. Go

[46] - https://en.wikipedia.org/wiki/Programming_paradigm
[47] - https://www.info.ucl.ac.be/~pvr/VanRoyChapter.pdf
[48] - Before starting a new project, there is a difficult decision on whether it will be based on a framework or on a CMS. When choosing to use a framework, you need to spend much time creating CMS features for the project. On the other hand, when choosing to use a CMS, it's more difficult to build custom application functionality. It is impossible or at least very hard to customize the core parts of the CMS.
https://statisticsanddata.org/data/most-popular-backend-frameworks-2012-2023/
https://w3techs.com/technologies/overview/programming_language

- Scripting
    1. **TypeScript**, Dart
    2. Python
    3. PHP

b. Runtime Environment

- Node[49] (ExpressJS, Angular, …)
- **Deno**[50]
- Bun[51]

c. CMS

- Wordpress, Joomla, Drupal, DotNetNuke, …

d. Frameworks

- **NestJS**, Fastify
- Laravel, Symfony
- Django, Flask
- Rails

e. Platforms (Cloud, Web, Desktop, Mobile, Game, IoT, Machine Learning, API, Microservice, …)

- Java (**Spring**, Micronaut, Quarkus, Dropwizard, Vertx, …)
- .Net (C#, ASP.net Core)

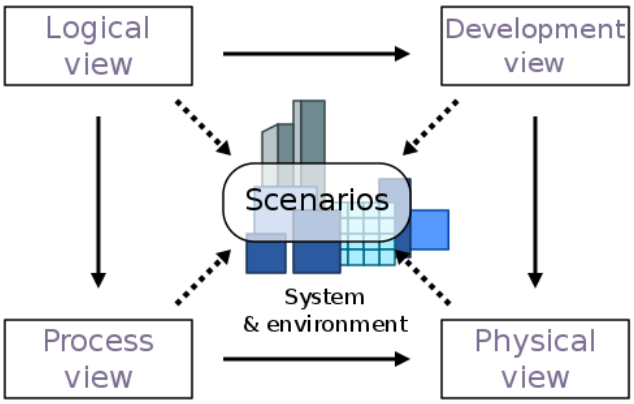
۹) Prototyping

a. Rapid Throwaway Prototyping
b. Evolutionary Prototyping
c. Incremental Prototyping
d. Extreme Prototyping

۱۰) نوع توسعه (مونولیتیک، ماجولار، هسته+پلاگین)

۱۱) سند معماری[52] ۴+۱

مدل ۴+۱ توسط فیلیپ کروتچن برای "توصیف معماری سیستم‌های نرم‌افزاری" معرفی شد. این مدل مبتنی بر استفاده از چند view است. viewها برای توصیف سیستم از دید مصرف کنندگان مختلف و سرمایه‌گذاران نرم‌افزار است مانند کاربران نهایی، برنامه نویسان نرم‌افزار و مدیران پروژه. ۴ view در این مدل شامل **مدل منطقی، توسعه، فرایند و فیزیکی** می‌شود. هر یک از دیدگاه های این معماری با یکی یا چند نمودار uml در ارتباط است[53].



[49] - https://nodejs.org
[50] - https://deno.com
[51] - https://bun.sh
[52] - https://en.wikipedia.org/wiki/4+1_architectural_view_model
[53] - https://www.omg.org/spec/category/software-engineering/ https://www.omg.org/spec/category/modeling/ , https://www.omg.org/spec/SysML/1.6/PDF https://www.omg.org/spec/category/business-modeling/

a. **مدل منطقی** Logical view : این دیدگاه نشان میدهد که عملکرد سیستم چگونه توسط طراحی داخلی فراهم میشود این دیدگاه ساختار ایستا و پویای داخل سیستم را مشخص میکند از جمله نمودار هایی با این دیدگاه در ارتباط اند می توان به **Class diagrams & Object diagrams** اشاره کرد.

b. **مدل توسعه** Development(implemention) View : دید توسعه برای تشریح سیستم از دید یک برنامه‌نویس است و درگیر مدیریت نرم‌افزار است. به این View همچنین Implementation View هم می گویند. در UML از نمودار **Component diagrams** برای توصیف اجزای سیستم استفاده می‌کند.

c. **مدل فرآیند** Process View : دید فرایند، ، درگیر وجهه پویای سیستم است. این دیدگاه المان های سیستم و ارتباط آن ها با هم و ترتیب انجام کارها بر اساس زمان را بررسی میکند از جمله نمودار های با این بخش در گیرند میتوان به **Sequence diagrams, Activity diagrams, State machine diagram** اشاره کرد.

d. **مدل فیزیکی** Physical(deployment) View : دید فیزیکی سیستم را از دید یک مهندس سیستم نمایش می‌دهد. این دید درگیر توپولوژی کامپونت‌های نرم‌افزاری در لایه فیزیکی است، به‌علاوه ارتباطات فیزیکی بین این اجزا. در UML از نمودارهای **Deployment diagrams** برای نمایش لایه فیزیکی استفاده می‌شود.

e. **سناریوها** Use Case : این دیدگاه دید کابران خارجی نسبت به نرم افزار را مورد بررسی قرار میدهد نمودار که در uml این دیدگاه را پوشش میدهد **Use case diagram** است.

۱۲) سند طراحی تفضیلی

a. Function Oriented[54] Design
b. Object-Oriented[55] Design

۱۳) Data Validation

۱۴) هماهنگی/تجمیع/انسجام[56] با سایر سیستمهای سازمان

۱۵) Business Analysis

۱۶) System Analysis

۱۷) استراتژی Top-Down و یا Bottom-up

۱۸) انتخاب نوع پایگاه داده SQL/NoSQL

a. Relational
b. Object Relational
c. Multi-Dimensional
d. NoSQL
   i. Key–value store
   ii. Document store
   iii. Graph
   iv. Object database
   v. Tabular
   vi. Tuple store
   vii. Triple/quad store (RDF) database
   viii. Hosted
   ix. Multivalue databases
   x. Multimodel database

[54] - Function Oriented design is a method to software design where the model is decomposed into a set of interacting units or modules where each unit or module has a clearly defined function. Thus, the system is designed from a functional viewpoint.
[55] - In the object-oriented design method, the system is viewed as a collection of objects (i.e., entities). The state is distributed among the objects, and each object handles its state data.

[56] – Integration

xi. Wide Column Store
xii. Native multi-model database

۱۹) انتخاب DB Engine
۲۰) مرزها و محدودیتها
۲۱) API
۲۲) مکانیزم دریافت/ارسال داده از/به سایر سیستم ها OLTP/ETL/OLAP
۲۳) لایه های نرم افزار:

a. Application layer
b. Execution layer
c. Semantic layer
d. Propagation layer
e. Consensus layer

محدودیتها

- **Technical Limitations**
  - Limited scalability
  - High costs
  - Lack of flexibility
  - Lack of privacy
  - The security model
  - Critical size
  - Hidden centrality
- **Nontechnical Limitations**
  - Lack of legal acceptance
  - Lack of user acceptance

فاز طراحی پایگاه داده

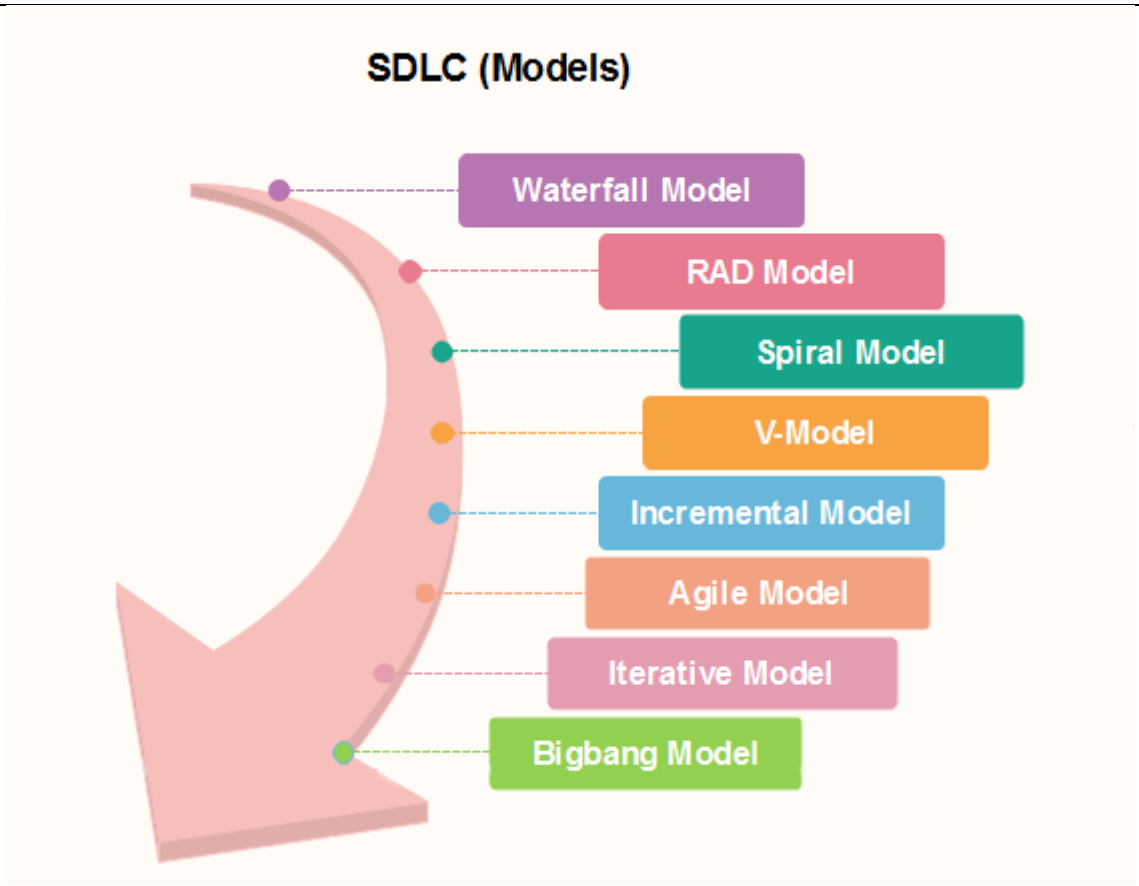
۱) ابزار طراحی مدل ER
۲) طراحی 3NF
۳) لیست جداول
۴) لیست ستونها/فیلدها
۵) کاربرد هر یک از جداول و ستونها
۶) ایندکسها (و انواع آن) و PK
a. روش به روزآوری ایندکسها
b. فواصل زمانی به روز آوری ایندکسها
c. ارائه ایندیکتوری که تریگر فعال سازی به روز آوری ایندکسها را فعال کند.
۷) جامعیت ارجاعی و FK
۸) لیست Viewها و نحوه عملکرد آن
۹) لیست Procedureها و نحوه عملکرد آن
۱۰) لیست Triggerها و نحوه عملکرد آن
۱۱) لیست Functionها و نحوه عملکرد آن
۱۲) لیست کاربران

۱۳) لیست مجوزها و نحوه تخصیص آنها
۱۴) Character set & Collation
۱۵) ORM
a. Data mapper pattern
b. Active record pattern
c. Object–relational mapping

متدولوژی انتخابی

۱) سند توصیف استانداردهای متدولوژی[۵۷]
a. مدل آبشاری Waterfall Model
b. مدل تدریجی Incremental Model
c. مدل حلزونی/مارپیچ Spiral Model
d. RAD
e. مدل چابک Agile[58]
i. Extreme Programming
ii. Kanban
iii. Scrum
f. DevOps
g. CI/CD[59]
i. continuous development
ii. continuous testing
iii. **continuous integration**
iv. **continuous delivery**
v. **continuous deployment**
vi. continuous monitoring
h. SAFe[60]
i. LeSS[61]
j. Big bang model : بر منابع تاکید داشته و برنامه ریزی چندانی نیاز نیست. نیازمندیها به محض وصول، درک شده و اجرا می شوند. مناسب پروژه های کوچک و آکادمیک.

[57] - https://en.wikipedia.org/wiki/Software_development_process
[58] - https://www.agilealliance.org , https://en.wikipedia.org/wiki/Agile_software_development
[59] - https://en.wikipedia.org/wiki/CI/CD
[60] - Scaled Agile Framework https://en.wikipedia.org/wiki/Scaled_agile_framework
[61] - Large-Scale Scrum



Pool of Ideas	Feature Preparation (3 - 10) In Progress	Feature Preparation (3 - 10) Ready	Feature Selected (2 - 5)	User Story Identified (30)	User Story Preparation (15) In Progress	User Story Preparation (15) Ready	User Story Development (15) In Progress	User Story Development (15) Ready (Done)	Feature Acceptance (8) In Progress	Feature Acceptance (8) Ready	Deploy-ment (5)	Delivered
Epic 431	Epic 444	Epic 662	Epic 602			Story 602-02, Story 602-03	Story 602-06, Story 602-04	Story 602-05, Story 602-01	Epic 401	Epic 609	Epic 694	Epic 294
Epic 478	Epic 589		Epic 302	Story 302-03, Story 302-01, Story 302-02, Story 302-06	Story 302-07, Story 302-08	Story 302-09	Story 303-05	Story 302-04	Epic 468	Epic 577	Epic 276	Epic 386
Epic 562	Epic 651		Epic 335	Story 335-09, Story 335-10, Story 335-04, Story 335-08, Story 335-01, Story 335-03	Story 335-05, Story 335-02	Story 335-06, Story 335-07			Epic 362		Epic 339	Epic 419
Epic 439			Epic 512	Story 512-04, Story 512-07, Story 512-02, Story 512-05, Story 512-06, Story 512-03	Story 512-01						Epic 521	Epic 388
Epic 329											Epic 582	Epic 287
Epic 287												Epic 274
Epic 606	Discarded											
	Epic 511	Epic 213										
	Epic 221											

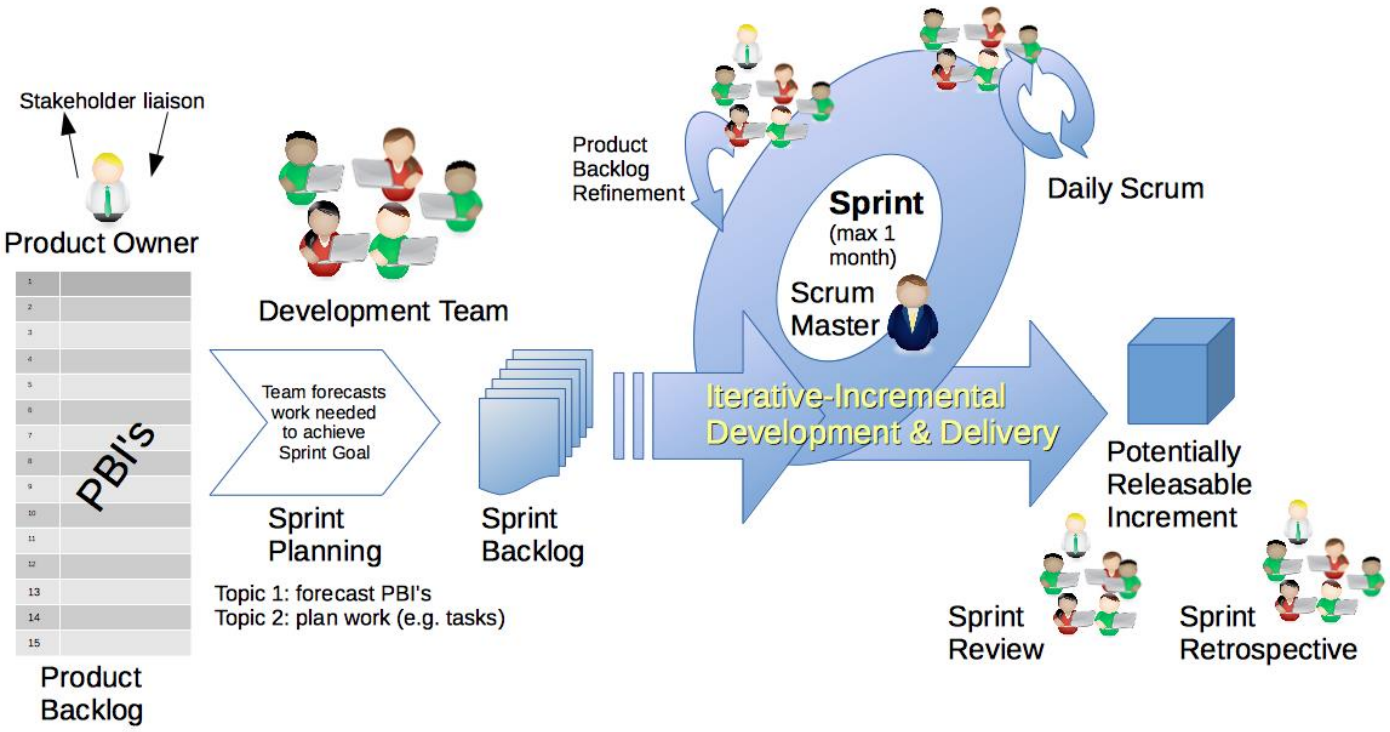
Policy — Business case showing value, cost of delay, size estimate and design outline.

Policy — Selection at Replenishment meeting chaired by Product Director.

Policy — Small, well-understood, testable, agreed with PD & Team

Policy — As per "Definition of Done" (see...)

Policy — Risk assessed per Continuous Deploy-ment policy (see...



چرخه توسعه نرم افزار

۱) محل پیاده سازی منطق کسب وکار Business Logic:

a. MVC(Model)

b. Database (Stored Procedure)

۲) مستندات فریم ورکها، میکروفریم ورک، دلیل انتخاب و نحوه مدیریت و به روز آوری آن

a. Backend

i. Synchronous I/O (PHP, Python, …)

ii. Asynchronous I/O (NodeJS, reactPHP, …)

b. Frontend

i. Responsive واکنشگرا

ii. Progressive پیش رونده

iii. Server side & Client-side Template engine

iv. CSS Framework (Bootstrap, …)

v. JavaScript Framework (React, Angular, VUE, …)

c. Fullstack

۳) مستندات اپلیکیشن، نحوه مدیریت و به روز آوری آن

۴) مستندات تطبیق فرآیندها و نیازمندیها با ماجولهای طراحی شده

۵) ساختار فایلها و دایرکتوری های ثابت و نقش هریک

a. *.php, *.html, *.css, *.js

b. *.jpg, *.png, *. webp

c. …

۶) ساختار فایلها و دایرکتوری هایی که رشد می یابند و نقش هریک

a. *. uploads
b. *. emails
c. *. logs
d. …

۷) ساختار فایلها و دایرکتوری های موقت و نقش هریک

۸) مستندات Best Practice[62] و استانداردهای مورد استفاده

a. File Encoding
  i. Unicode/UTF-8
  ii. ASCII
b. Autoload Standards[63](PSR-0, **PSR-4**)
c. Programming Style
  i. Simplicity
  ii. Control Constructs
  iii. Nesting
  iv. Information Hiding
  v. User-Defined Types
  vi. Side Effects
  vii. Module size
  viii. Module Interface
  ix. Name Space
d. Coding Standards[64]
  i. Code Well documented
  ii. Inline comments
  iii. Line Length
  iv. Spacing
  v. Indentation
  vi. Use of global
  vii. Naming conventions
  viii. Error Messages & Excention handling
  ix. Structured Programming[65] Rules
  x. Pseudo Codes
  xi. Don’t use goto
e. HTML Forms Implementation

۹) تطبیق استانداردهای سورس کد با آنچه که پیاده سازی شده است

۱۰) یکسان سازی کدها Code review

[62] - http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines

[63] - https://www.php-fig.org/psr/ و https://github.com/php-fig/fig-standards/tree/master/accepted

[64] - PSR-1, PSR-2, PSR-12

[65] - https://www.javatpoint.com/software-engineering-structured-programming

۱۱) نحوه ارتباط با سایر نرم افزارها و مدیریت آن
۱۲) Vendor Packages/Plugins/Library
۱۳) Version Control
۱۴) Unit tests code
۱۵) Patterns
۱۶) Micro Services
۱۷) اجرای نسخه Prototype
۱۸) UX/UI
۱۹) SEF[66]/SEO
   a. Server-Side Render
   b. Client-Side Render
۲۰) MVC
۲۱) Template Engine
۲۲) License
۲۳) محیطهای استاندارد
   a. Development
   b. Pre-Production/Staging/Testing
   c. Production/Live
۲۴) تکنولوژی ها (رجوع به پیوست ۲)

کنترل کیفی و نظارت (اندازه گیری/ارزشیابی متریک ها)

۱) انطباق با استانداردهای کنترل کیفیت
   a. ISO 9001
      i. Functionality
      ii. Reliability
      iii. Usability
      iv. Efficiency
      v. Maintainability
      vi. Portability
   b. ISO/IEC 9126 → ISO/IEC 25010:2011 → **ISO/IEC 25019**
   c. ISO/IEC JTC 1/SC 7
   d. SEICMM
   e. PCMM
   f. Six Sigma
      i. DMAIC
      ii. DMADV
۲) قابلیت ردیابی Trace ability
۳) درستی Correctness
۴) اعتبار Validity

[66] - Search Engine Friendly & Search Engine Optimization

۵) کفایت Sufficiency
۶) سازگاری Consistency
۷) یکنواختی Uniformity
۸) امکان پذیری Feasibility
۹) نگهداشت پذیری Maintainability
۱۰) مقیاس پذیری Scalability
۱۱) مدلها
   a. Capability Maturity Model (CMM)(5 Levels)
   b. McCall's Quality Model
   c. Boehm's Software Quality Model
   d. COCOMO (Cost Constructive Model) Model
      i. Basic Model
      ii. Intermediate Model
      iii. Detailed Model

چرخه تست نرم افزار[۶۷]

نکته: تست فقط وجود خطا را نشان می‌دهد و نه عدم وجود آن را. پیدا نشدن خطا در تست به معنای بدون خطا بودن برنامه نیست.

۱) تست سند گزارشات بیزینسی
۲) تست آماره های اپلیکیشن/سایت
۳) سطوح تست :

a. **سند تست واحد (Unit testing[68]) :** تست واحد یا micro level پایین‌ترین سطح تست است. هر کد تست واحد، یک قطعه کد یا یک تابع (متد) خاص را تست می‌کند. این تست نیاز به دانش در مورد طراحی و نحوه کارکرد داخلی تابع یا قطعه کد دارد[۶۹] و توسط برنامه‌نویس **(و نه تست‌کننده)** انجام می‌شود. **در این تست صرفاً ورودی های مشخصی به توابع داده شده و صحت خروجی های مورد انتظار تست می گردد.** هنگامی که توسعه توسط روش TDD[۷۰] انجام میگیرد می توان از این نوع تست سود برد.

b. **سند تست یکپارچگی افزایشی :** تست یکپارچه‌سازی افزایشی با افزوده شدن قابلیت جدید به نرم‌افزار، مجدداً نرم‌افزار تست می‌شود. هدف این تست، بررسی درستی نرم‌افزار پس از افزوده شدن امکان جدید است.

---

67 - https://en.wikipedia.org/wiki/Software_testing

68 - Test Methods of a class. Unit tests involve testing a part of an app in isolation from its infrastructure and dependencies. When unit testing controller logic, only the contents of a single action are tested, not the behavior of its dependencies or of the framework itself.

69 - Unit test included but not limited to below list:
1) **Overflows and underflows:** Make sure that your code doesn't allow numbers to become larger than the largest valid value or smaller than the smallest valid value. Either situation will cause an error.
2) **Valid return values:** Ensure that each function returns values that are valid for the caller. In some cases, the return value is calculated. Your tests should ensure that any calculated return values are always valid.
3) **Boundary conditions:** Always test that your code handles data that meets or exceeds expected limits.
4) **Iteration limits:** Test each looping structure to ensure that it doesn't iterate more times than you intend and burn up all your gas.
5) **Input and output data formats:** Test your code to make sure that it handles data provided or returned in unexpected formats.
6) **Input and output data validation:** Ensure that your code either sanitizes or rejects invalid characters or sequences of characters.

70 - Test driven development (TDD)
Behavior driven development (BDD)

c. **سند تست یکپارچگی (Integration Testing[71])** : همانطور که از اسم آن نیز پیداست از این روش برای تست صحت کارکرد درست بین اجزای/بخشهای مختلف سیستم استفاده می شود. این تست بر روی بررسی ارتباط داده‌ها(Data Communication) در میان ماژول‌های مختلف متمرکز است. تست یکپارچه‌سازی برای تایید ماژول‌های نرم‌افزاری برای کار در یک پیکر واحد ضروریست[72]. برخلاف تست واحد که اجزا به صورت مجزا و ایزوله تست میشوند در اینجا هماهنگی بین کل اجزا سیستم شرط اساسی است. بدلیل پیچیدگی پروسه، انجام تست های این بخش کندتر از تست واحد صورت می گیرد. تعداد تست های این بخش کمتر از تعداد تستهای واحد می باشد. در برخی موارد انجام تست های این بخش توسط همان ابزار تست واحد نیز میتواند انجام گیرد.

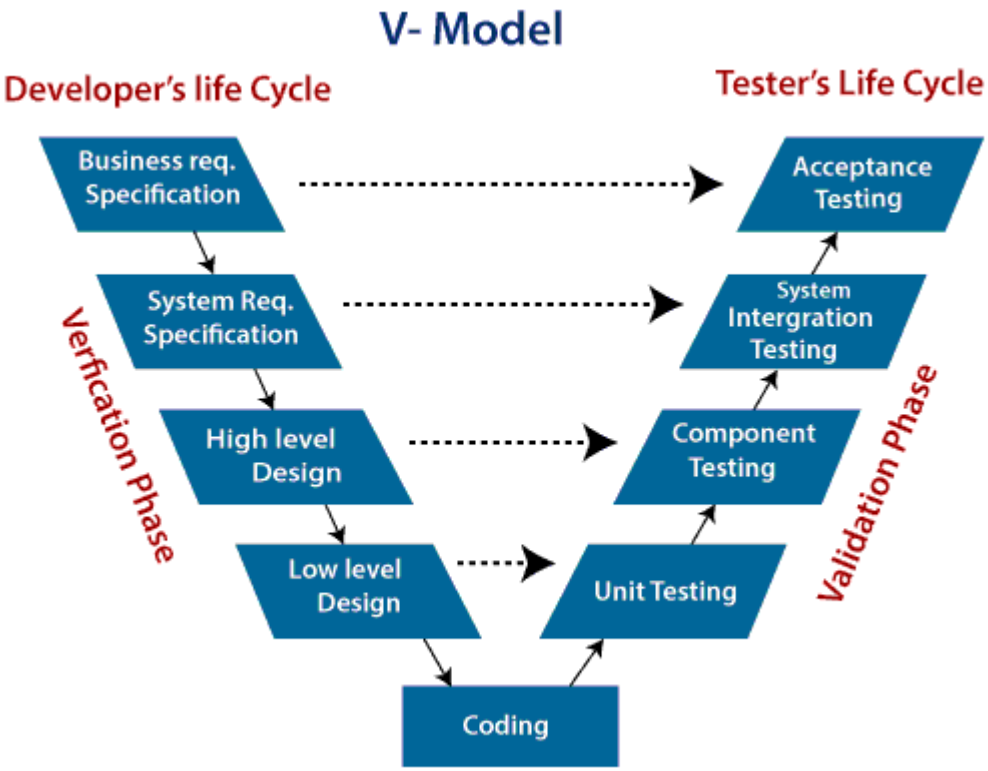
d. **سند تست سیستم (System Testing[73])** : به منظور بررسی عملکرد نرم‌افزار بر روی پلتفرم‌های مختلف انجام می‌شود.

1) Page testing
2) Cross-page testing
3) Logic testing
4) Linting (isn't about finding errors, but potential errors)
5) Link checking (making sure there are no broken links on your site)

e. **سند تست پذیرش (Acceptance Testing[74])** : هدف از این آزمون اطمینان از این نکته است که سیستم در شرایط عملیاتی معمولی و با اطلاعات واقعی قادر به برآورده کردن نیازهای کاربران می باشد.

۶) ابزارهای تست
۷) آزمون استقلال سیستم از پلتفرم (Linux, Windows, Mac,…)
۸) آزمون استقلال سیستم از مرورگر (Chrome, Firefox, Edge, IE, Safari,…)
۹) آزمون اندازه‌های مختلف صفحه نمایش.

71 - Checks if Class A works with Class B and Ensure different parts of system work together.
72 - Integration test included but not limited to below list:
1) Wrong function name
2) Wrong number or format of input parameters
3) Out of range or bad input data
4) Input data containing boundary values
5) Wrong expected output parameters
6) Attempt to call a function that isn't visible
7) *Smart contract* function properly completed with return codes
8) Set a timeout for a function call that is too short
9) Reverse a transaction
73 - System testing ensures the whole system works as user expected before sending it to acceptance testing.
74 - When tests above are for developers at development stage. Acceptance tests are actually done by the users of the software. Users do not care about the internal details of the software. They only care how the software works.



۱۰) آزمون کارکردی (مشخصات پیش بینی شده در اهداف ورودی و خروجی کارکردی)

۱۱) آزمون عملکرد (کارکردی با هزینه(مصرف زمان و منابع) قابل قبول. آستانه پذیرش کارایی سیستم در هر کارکرد باید با توافق کاربر نهایی تعیین گردد.)

۱۲) آزمون همسازی داده ها Data integrity (در صورت کار دائم سیستم، هیچ یک از Constraintها نقض نشود.)

۱۳) آزمون چرخه کسب و کار Business cycle

۱۴) آزمون واسط کاربر GUI (E2E Test, Browser/ Functional testing)

۱۵) آزمون امنیت

۱۶) تست نفوذ Penetration test

۱۷) آزمون تحمل خرابی Fault-tolerance (Planned/Unplanned crash recovery)

۱۸) آزمون پیکربندی

۱۹) آزمون بازگشتی Regression (پس از هربار ارائه یک نسخه جدید از سیستم)

a. تصحیحات انجام شده، منجر به رفع اشکالات قبلی یا بهبود کارایی سیستم شده است.

b. تصحیحات انجام شده، منجر به بروز اشکالات جدید در دامنه پوشش آزمونهای قبلی نشده است.

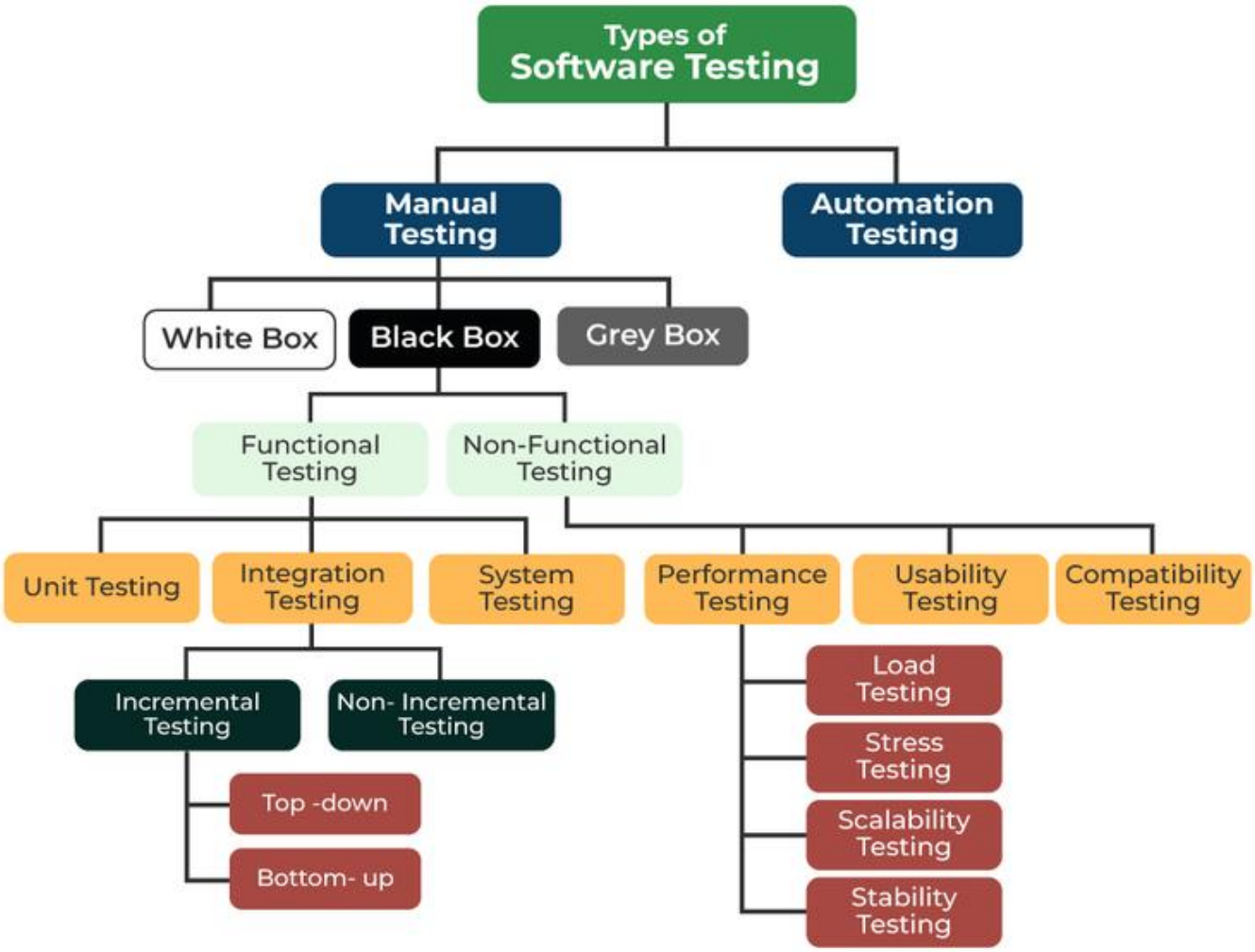
۲۰) آزمون تحمل بار Load (پایداری سیستم در ماکزیمم پیک کار به مدت ۷۲ ساعت)

۲۱) آزمون تنش Stress

۲۲) Alpha/Beta Testing

۲۳) تست API طراحی شده برای نرم افزار.

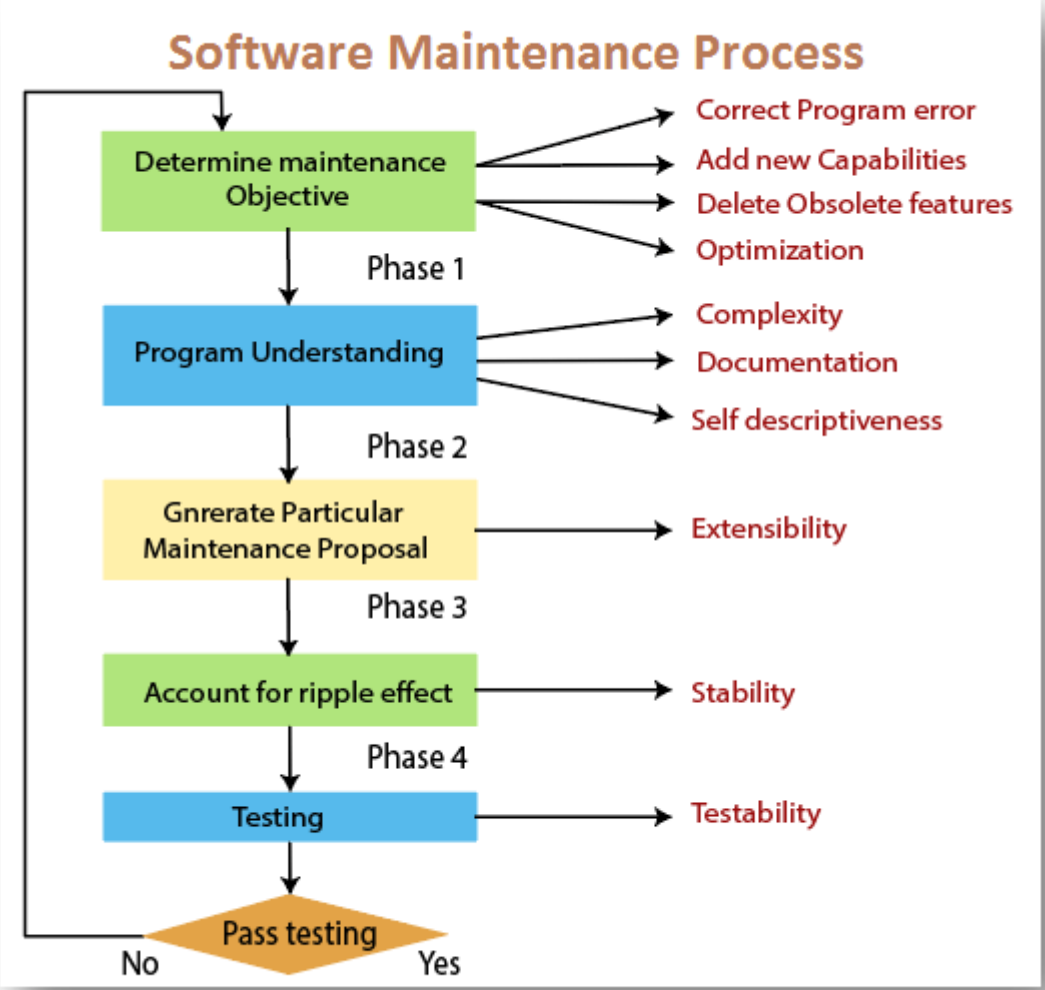
۲۴) Pre-flight check : اطمینان از اینکه سیستم در محیط عملیاتی نیز همانند محیط تست، بدون مشکل اجرا خواهد شد. تست این بخش معمولا شبیه Smoke testing می باشد.



نگهداری

لیست زیر بخش از مواردی است که نیاز به نگهداری را لازم می سازد:

۱) اصلاح مشکلات گزارش شده
۲) تغییر در نیازمندیهای کابران
۳) تغییر در نیازمندیهای سخت افزاری و نرم افزاری
۴) بهبود کارائی
۵) اصلاح عناصر برنامه
۶) کاهش عوارض جانبی ناخواسته



انواع سیاستهای نگهداری :

۱) Corrective Maintenance
۲) Adaptive Maintenance
۳) Preventive Maintenance
۴) Perfective Maintenance

ابزارها و نحوه پیکربندی آنها[۷۵]

۷) کانتینرها (Docker,...)
۸) پایگاه داده MySQL, MariaDb, ...
۹) مفسر PHP, NodeJS, ...
۱۰) وب سرور Apache, Nginx, ...
۱۱) پلاتفرم
۱۲) فریم ورکها
۱۳) Template engines

[75] - https://en.wikipedia.org/wiki/Outline_of_software_engineering

۱۴) ابزار مدیریت وابستگی ها (Composer ,NPM ,...)
   a. Package/Plugins های مورد نیاز
   b. 3rd Party Library
۱۵) ابزارها و سیاستهای بکاپ گیری

راه کارهای جلوگیری از کاهش کارائی Performance سیستم

۱) **Scalability**
   a. Horizontal Elasticiy Replica
   b. Load Balancer
   c. Clustering
۲) **Performance**
   a. Minimize blocking calls (Asynchronous Architecture)
   b. Optimize disk I/O management
   c. Cache at all tiers
   d. Reduce round trips
   e. Effective use of partitioning at the data tier
   f. Concurrent/Multithreading/Parallel Programming
   g. JIT
   h. انجام Best Practice ها و بهینه سازی کد
   i. بهینه سازی دیتا بیس
   j. بهینه سازی وب سرور
   k. بهینه سازی مفسر/کامپایلر
   l. بهینه سازی فریم ورک
   m. بهینه سازی معماری و طراحی کل سیستم

امنیت

۱) Authentication/Authorization
۲) Encryption/Hash Algorithms
۳) Penetration Test
۴) SQL Injection
۵) Hardening
   a. Keep update
   b. Hard password to guess
   c. Backup
   d. ....
۶) Over Flow
۷) PHP Sodium Extension + Hasher/Encryption Algorithms

فاز استقرار

۱) سند انتقال و واگذاری نرم‌افزار
۲) سند نگهداری و به روز آوری (Update/Upgrade/Patch/BugFix/Refactoring[76]) سیستم
۳) تفاوتهای پیکربندی نسخه Development با نسخه Production

[76] - Refactoring is the process of improve and reorganizing code without affecting its original functionality.

۴) تفاوتهای پیکربندی Host, Localhost
۵) ابزارهای Upload/Download/Backup
۶) پیکربندی DNS
۷) پیکربندی eMail
۸) مانیتورینگ، NOC
۹) Logs و Log circulation

کیفیت ارائه سرویس SLA (Service-level agreement)

۱) Planned down time
۲) Unplanned down time
۳) Crash/Recovery Policy
۴) Restore & Resolution time
۵) Retention time
۶) Ticketing
۷) پروتکل ارتباط با مشتریان، نحوه ارجاع مشکلات به واحد فنی(L2,L3) و پروسه رفع آن

تحویل دادنی ها[۷۷]

۱) نسخه نهائی اجرائی نرم افزار(به صورت Optimum[78])
   a. **تحویل آزمایشی[۷۹] (به صورت هفتگی)** (محیط آزمایشی،تست کارکردی و عملکردی با داده های تستی و واقعی)
   b. **تحویل موقت(فاز انتهایی پروژه)** (محیط عملیاتی، عملیاتی شدن سیستم و تست پایداری سیستم و بررسی باگهای احتمالی(سه الی شش ماه))
   c. **تحویل دائم(خاتمه پروژه)** (گذراندن کلیه تستهای آزمایشی و عملیاتی و پایدار شدن سیستم)
۲) پایگاه اطلاعاتی سیستم(اسکریپت ایجاد، Dump)
۳) راهنمای نصب و استقرار
۴) راهنمای کاربران
۵) آموزش کاربران
۶) راهنمای عملیاتی سیستم
۷) طرح آزمون پذیرش
۸) کلیه اسناد توسعه نرم افزار[۸۰]
   a. اسناد الزامات کسب و کار
   b. اسناد الزامات عملکردی
   c. اسناد الزامات بازار
   d. اسناد الزامات استانداردها(محیطی، داخلی)
   e. اسناد الزامات محصول
   f. اسناد الزامات رابط کاربری
   g. اسناد الزامات فنی
   h. اسناد مشخصات الزامات نرم افزار
   i. اسناد الزامات کیفیت

---

[77] - https://en.wikipedia.org/wiki/Outline_of_software_engineering ***See Deliverables***

[78] – نسخه ای کامل است که نتوان چیزی از آن کم کرد.

[79] – Demo به صورت هفتگی

[80] – اگر نرم افزار جهت فروش/صادرات باشد نیازی نیست، ولی چنانچه "خریدار/مشتری/کاربرنهائی/کارفرما/بهره بردار" خود بایستی ادامه توسعه و نگهداری را عهده دار باشد، ارائه کلیه مستندات توسعه نرم افزار توسط "پیمانکار/مجری"، ضروری خواهد بود.

j. اسناد الزامات تست نرم افزار
k. اسناد الزامات امنیتی نرم افزار
l. اسناد الزامات مشتری

پرداختها

پروسه تحویل گیری پروژه های بزرگ معمولا طولانی و بتدریج می باشد، که پیشنهاد میشود شامل ۱۵ زیر بخش لیست زیر باشد (تا از ریسکها و مشکلات سایر پروژه های مشابه دوری جست):

۱) پیش پرداخت اولیه
۲) پرداخت تحویل سورس کد و مستندات توسعه دهندگان نرم افزار
- مستندات طراحی نیازمندیها Requirements.
- مستندات طراحی تفضیلی و معماری.
- مستندات و سورس کد طراحی دیداری Mockups.
- مستندات و سورس کد نمونه اولیه.
- مستندات و سورس کد طراحی و تست امکانات کارکردی اولیه/اجباری .
- مستندات و سورس کد طراحی و تست امکانات کارکردی ثانویه و گزارشات.
- مستندات و سورس کد طراحی API.
- مستندات و سورس کد پلاگینها.
- مستندات و سورس کد طراحی و تست امکانات عملکردی(از نظر Performance, ...) .
- مستندات و سورس کد طراحی تست کلی نرم افزار و تست نفوذ.

i. Unit test
ii. Integration test
iii. System test
iv. Acceptance test

۳) پرداخت تحویل مستندات و سورس طراحی نحوه استقرار، نگهداری و پشتیبانی سیستم روزانه/هفتگی/ماهانه/سالانه و سطح کیفیت سرویس SLA.
۴) پرداخت مستندات آموزشی کاربران.
۵) پرداخت تحویل موقت.
۶) پرداخت پس از دوره گارانتی و تحویل دائمی.

دلایل شکست پروژه ها

سازمان پروژه معمولا به ۳ لایه استراتژی، تاکتیکی و تکنیکی تقسیم میشود که لایه های بالاتر ماموریت و متریکهای لایه پائین تر را تعریف و بر حسن اجرای آن نظارت میکند.

در لایه استراتژیک بایستی به دنبال موارد زیر بود:

- هدف چیست؟
- فاکتورها/عوامل اساسی و بنیادین چیست؟
- شناخته ها و ناشناخته ها چیست؟
- امکانات، منابع، محدودیتها و الزامات چیست؟
- کدام راه حل ها، انعطاف پذیر هستند؟
- چگونه این راه حل ها ارزیابی و اعتبار سنجی می شوند؟

- کدام راه حل بهینه است؟

علل استراتژیک/راهبردی شکست پروژه

۱) ضعف در هدف گذاری (مشخصات محصول) و بازار هدف(نوع مشتریان)
۲) فقدان ضمانت اجرایی
۳) کمیته راهبری ضعیف
۴) فقدان چشم انداز روشن
۵) ارتباطات ضعیف
۶) ضعف در تعریف فرآیندهای سازمان
۷) عدم اختصاص منابع (مالی،انسانی،فنی و ...)
۸) ضعف مدیریت در تعهد طولانی مدت

علل تاکتیکی شکست پروژه

مشکلات محیطی

۱) منابع ناکافی/محدود
۲) رقابت شدید
۳) عدم نیاز بازار

مشکلات داخلی

۱) تخمین های مدیریت پروژه[۸۱]
۲) عدم آمادگی و ضعف در آماده سازی
۳) بینش یا هدف ناکافی برای پروژه و یا عدم دستیابی به اهداف پروژه
۴) ایجاد انتظارات و توقعات بیش از حد
۵) عدم تصویر یکسان کارفرما و مشتری در خصوص محصول نهائی
۶) ضعف در تحلیل ریسکهای موجود، عدم شناخت کافی نسبت به موانع و عدم جلوگیری از واگرائی ها
   a. اشتباه در زمانبندی پروژه
   b. افزایش پیچیدگی
   c. افزاش چالشها
   d. افزایش تقاضای مشتریان و عدم برآورده سازی نیازهای کاربران
   e. کیفیت پائین محصول نهائی
   f. برآورد نادرست هزینه‌های آشکار و غفلت از هزینه های پنهان
      i. هزینه های فنی
         ۱. Module Independence
         ۲. Programming Language
         ۳. Programming Style
         ۴. Program Validation and Testing
         ۵. Documentation
         ۶. Configuration Management Techniques
      ii. هزینه های غیر فنی
         ۱. Application Domain
         ۲. Staff Stability
         ۳. Program Lifetime
         ۴. Dependence on External Environment

[81] - https://en.wikipedia.org/wiki/Estimation_(project_management)
https://en.wikipedia.org/wiki/Software_development_effort_estimation

۵. Hardware Stability

۷) ضعف در تعریف فرآیندهای پروژه ها

۸) وابستگی به منابع (مالی، انسانی، ...)

۹) وابستگی وظایف

۱۰) نیازمندی های گمراه کننده

۱۱) عدم برنامه ریزی منابع

۱۲) قربانی کردن کیفیت محصول(NonFuntional)

a. عدم مقیاس پذیری

b. عدم یکپارچگی

c. عدم قابلیت نگهداری

۱۳) ضعف استراتژی و مدیریت بازاریابی محصول

۱۴) فقدان دیدگاه مشتری مداری

۱۵) مدیریت تغییر غیر اثربخش

۱۶) تغییر الویت بندی ها

۱۷) فقدان و ضعف در سطح مشاوران

۱۸) **عدم وجود جهتی واحد :** بیش از هر چیز، یک تیم موثر به یک هدف مشترک نیاز دارد. پروژه ها معمولاً از ریل اصلی خود خارج می شوند زیرا هدف اصلی خود را گم کرده و یا اصلا این هدف مشخص نیست.

۱۹) نداشتن تجربه و ارتباط مورد نیاز برای جذب سرمایه گذار و نبود مدیریت صحیح در مسائل مالی شرکت و پروژه

۲۰) **توهم تکنولوژی برتر :** برخی افراد تصور می کنند که اگر در یک پروژه از تکنولوژی های به اصطلاح جدیدتری استفاده کنند پروژه شکست نخواهد خورد این افراد به بیماری High-Tech illusion دچار هستند .

۲۱) انگیزش ضعیف تیم پروژه

۲۲) کار هوشمندانه در مقابل کار سخت

۲۳) برنامه ضعیف آموزش

۲۴) تغییرات سریع تکنولوژی

۲۵) نبود مستندات

۲۶) ناتوانی روش های تولید نرم افزار در پاسخگویی به افزایش تقاضا

علل فنی/تکنیکی شکست پروژه

۱) بی تجربگی و عدم تعهد کاری مدیر پروژه

۲) عدم رعایت استانداردهای مهندسی نرم افزار

۳) **عدم درک درست از نیازمندیهای کاربران :** به سرعت قابل تشخیص و رفع شدن است.

۴) **معماری فنی نامناسب :** شناسائی این ریسک سخت بوده و معمولا تا به یک بن بست پیاده سازی نرسید متوجه چنین اشتباهی نخواهید شد.

۵) **اشتباه در طراحی :** از خطرناکترین اشتباهات بوده و تشخیص آن بسیار سخت است. چرا که معمولا تمام نیازهای عملیاتی کابران پوشش داده می شود ولی کاربران همچنان ناراضی است و پیاده سازی فعالیتها یکی پس از دیگری سخت تر شده و تغییرات هر بار با هزینه بیشتری همراه است.

۶) ضعف در فرآیند مهندسی معکوس و مهندسی مجدد

۷) ایده آل گرائی افراطی

۸) ابزارگرائی

۹) برنامه ریزی نامناسب

۱۰) کمبود نیروی فنی متخصص و با تجربه

۱۱) عدم ثبات منابع انسانی در تیم پروژه
۱۲) عدم شناخت صحیح تکنولوژی مورد استفاده
۱۳) فقدان مستندات
۱۴) پشتیبانی اجرایی ضعیف
۱۵) عدم تحویل به موقع
۱۶) عدم تامین نیازمندی های کاربر
۱۷) کیفیت پایین و پایدار نبودن نرم افزار
۱۸) **سندرم عجله** : وقت برای فکر کردن نیست ، فقط انجام بده **(نتیجه : بدهی فنی)**.

ریفرنسها

https://en.wikipedia.org/wiki/Agile_software_development
https://en.wikipedia.org/wiki/Anti-pattern
https://en.wikipedia.org/wiki/Code_audit
https://en.wikipedia.org/wiki/Cost_estimation_in_software_engineering
https://en.wikipedia.org/wiki/Cyclomatic_complexity
https://en.wikipedia.org/wiki/DevOps
https://en.wikipedia.org/wiki/Estimation_(project_management)
https://en.wikipedia.org/wiki/Extreme_programming
https://en.wikipedia.org/wiki/Halstead_complexity_measures
https://en.wikipedia.org/wiki/International_Software_Testing_Qualifications_Board
https://en.wikipedia.org/wiki/List_of_software_development_philosophies
https://en.wikipedia.org/wiki/Programming_complexity
https://en.wikipedia.org/wiki/Programming_paradigm
https://en.wikipedia.org/wiki/Scrum_(software_development)
https://en.wikipedia.org/wiki/Software_development_effort_estimation
https://en.wikipedia.org/wiki/Software_metric
https://en.wikipedia.org/wiki/Software_quality
https://en.wikipedia.org/wiki/Software_quality_assurance
https://en.wikipedia.org/wiki/Software_quality_control
https://en.wikipedia.org/wiki/Static_program_analysis
https://en.wikipedia.org/wiki/ISO/IEC_9126
https://www.educba.com/software-development/software-development-tutorials/software-engineering-tutorial/
https://www.geeksforgeeks.org/software-engineering/
https://www.javatpoint.com/advantages-and-disadvantages-of-software-engineering
https://www.javatpoint.com/software-engineering
https://www.scaler.com/topics/software-engineering/

پیوست ۱ : Best Practices[82]

Information security (CIA triad)

1) Confidentiality
2) Integrity (یکپارچگی (در ساخت) استحکام، انسجام)
3) Availability

CAP & PACELC theorem (Distributed Sytems)

1) **Consistency:** Every read receives the most recent write or an error. (ثبات یکپارچگی، انسجام)(هماهنگی، سازگاری)
2) **Availability:** Every request receives a (non-error) response, without the guarantee that it contains the most recent write or data.
3) **Partition tolerance:** The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes. Partition tolerance refers to the tolerance of a storage system to failure of a network partition. Even if some of the messages are dropped or delayed the system continues to operate.

Onion Architecture

1) Core Domain: (Should be isolatd from the outside world)
    a. Entity
    b. Commands (Actions)/Events (Result of actions)/Messages (Commands & Queries)
    c. Domain Services
2) Non-Core Domain:
    a. Database/Repositories
    b. File systems
    c. 3rd parties' systems
3) Non-Domain:
    a. Application Services
    b. UI

Mindsets

1) Reactive Manifesto https://www.reactivemanifesto.org/
2) Frameworkless Manifesto https://www.frameworklessmovement.org/ https://github.com/frameworkless-movement/manifesto
3) spotify squad/guild model https://www.atlassian.com/agile/agile-at-scale/spotify
4) Manifesto for Agile Software Development https://agilemanifesto.org/
5) Twelve Principles of Agile Software https://agilemanifesto.org/principles.html

Design Principles

Goal of Design Principles
Code must be KISS, DRY and YAGNI!

1) Design principles[1] should lead to good object-oriented design.
2) Code and architecture smells are based on the violation of accepted design principles.
3) Provide important information on how to fix code and architecture smells.
4) If the violated design principle can be identified, it provides a first indication of what a better structure for the system might look like.
5) Have been published and propagated by Robert C. M

SOLID[83] (object-oriented design)

1) **Single responsibility principle (SRP):** Every class should have only one responsibility.
2) **Open-closed principle (OCP):** This rule is about an Inheritance and Abstraction. Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification.
3) **Liskov substitution principle (LSP):** Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it.
4) **Interface segregation principle (ISP):** Clients should not be forced to depend upon interfaces that they do not use.
5) **Dependency inversion principle (DIP):** Depend upon abstractions, [not] concretions.

[82] - https://en.wikipedia.org/wiki/List_of_software_development_philosophies

[83] - https://en.wikipedia.org/wiki/SOLID , https://www.tutorialsteacher.com/csharp/solid-principles

KISS (Keep it simple, stupid)

- Think of several options and then try the simplest option first.
- Always ask: Is there an easier way to do this?
- Think about the future maintainers, assume that they will be you, and work on that basis.
- When you look at existing systems, ask: Do I need all this stuff or Do we need it at all?

- Means that always the simplest solution of a problem should be chosen.
- Avoid code that is too complex and therefore too complicated.
- Finding simple solutions is a rule that helps to avoid various errors.
- A healthy rejection of non-simple solutions is a sign of searching for good code.
- The more difficult code is to explain, the more likely it is that it is more complicated than necessary and is not the most elegant solution.
- Make things as simple as possible, but not simpler!

YAGNI (You Aren't Gonna Need It)

- The architecture should be intended to support current and future requirements agreed with business stakeholders.
- Where no requirement, also no implementation!
- A program should only implement functionality when this functionality is needed.
- Contrary to this approach, in practice it is often attempted to prepare programs for possible future change requests and features through additional or more general code. "…this is what THEY will demand soon anyway…"
- Such code can be very annoying to read, understand, change existing functionality and costs time and money!

POLA (Principle of least astonishment)

- Keep things consistent so that people are not surprised when they find that a similar task is being done in a different place in a different way.
- If a difference is needed, document why.
- Consistency guides understanding, if you name things wrong or call the same thing by different names, you increase complexity.

LoD (The Law of Demeter) (a less well-known principle) (principle of least knowledge)

1) A unit should have only limited knowledge about other units: only units "closely" related to the current unit.
2) A unit should only talk to its immediate friends
3) A unit should not talk to strangers

SCP (Speaking Code Principle) Writing code that speaks for itself and that does not need a comment.

- Code should communicate its purpose, even without comment and documentation.
- Comments are no substitute for bad code.
- Clear and expressive code with few comments is much better than messy and complex code with numerous comments.
- Instead of spending time writing comments that explain the chaos, spend time on cleaning up the chaos.
- Comment must be adjusted when the code is changed.

DRY (Don't repeat yourself)

- Refers to the avoidance of duplicated code, i.e. code fragments that are implemented in the same or very similar way in several places.
- Redundant existing source code is difficult to maintain, as consistency between the individual duplicates must be ensured.
- In systems that remain loyal to the DRY principle, however, changes need only be made in one place.

SoC (Separation of concerns[84]) (single responsibility architecture)

- Keep related things together, unrelated things apart.
- A component should have exactly one task.
- Avoid mixing of several responsibilities in e.g. one class.
- Components become simpler, easier to understand, maintain and have better reusability.
- The complex of tasks of a unit should be self-contained (high cohesion).
- The unit should depend as little as possible on other units (low coupling).
    - N-tier architecture

OOD

Four Pillars of OOPs (object-oriented programming) in Your Projects

[84] - https://en.wikipedia.org/wiki/Separation_of_concerns

1) Data abstraction is the process of hiding unnecessary details of an object's internal structure. By abstracting an object's data, its structure and behavior can be kept separate and more easily understood.
2) Encapsulation is the process of wrapping data and related functions into a single unit (object). Encapsulation limits access to object data and methods, preventing their misuse and ensuring their proper functioning.
3) Inheritance is the ability to create a new class (child class) from an existing one (parent class). The child class typically inherits the attributes (members and methods) of the parent class, although it can also redefine them.
4) Polymorphism is the ability of an object to take on multiple forms. This allows objects of different classes to be used interchangeably, as long as they implement a certain interface (have methods of the same name).

- Encapsulation is a way to keep code organized and separate from other codes in the system. It is done by using classes that contain all the code related to a specific feature or subject.
- Abstraction is the process of hiding internal details or processes from the user. This is done by creating classes that represent real-world objects and their attributes.
- Inheritance is when a class inherits the methods and properties of another class. This allows developers to reuse code and easily add new features.
- Polymorphism allows developers to create multiple methods with the same name, but with different implementations. This makes it easier to create flexible systems that are easier to understand and maintain.

Finally, the fifth principle is modularity, which means that a program should be broken down into smaller, more manageable pieces. This makes it easier to update and maintain the code and reduces the risk of introducing bugs.

Architectures

1) Microservices architecture
    a. Saga Orchestration vs Choreography vs Hybrid approach
2) Stateless services architecture
3) Event-driven architecture
4) Data storage and access architecture
5) Resiliency architecture
6) Evolution and operations architecture

Domain-Driven Design 4-tier architecture

1) Presentation Layer
2) Domain Layer (Business Part)(No Dependency)
3) Infrastructure Layer (Depend on Domain Layer)
4) Application Layer (Technical Part)(Depend on Domain Layer & Infrastructure Layer)

Anaemic Domain Model vs. Rich Domain Model

Facade pattern[85]

a software-design pattern commonly used in object-oriented programming

Other Robustness[86] principles

1) CQRS[87] and Event Sourcing "Command and Query Responsibility Segregation"
    a. SQL/Relational: High normal forms are good for commands(Create, Update, Delete)
    b. NoSQL : Low normal forms are good for queries(Read/Select)
    c. -- Note : CAP & PACELC theorem[88]
2) GRASP (object-oriented design) "General Responsibility Assignment Software Patterns (or Principles)"
3) DTSTTCPW "do the simplest thing that could possibly work"
4) GraphQL[89]

SaaS (twelve-factor[90] app)

In the modern era, software is commonly delivered as a service: called web apps, or software-as-a-service. The twelve-factor app is a methodology for building software-as-a-service apps that:

[85] - https://en.wikipedia.org/wiki/Facade_pattern
[86] - https://en.wikipedia.org/wiki/Robustness_principle
[87] - https://cqrs.files.wordpress.com/2010/11/cqrs_documents.pdf
[88] - https://en.wikipedia.org/wiki/PACELC_theorem
[89] - graphql.org
[90]- https://12factor.net

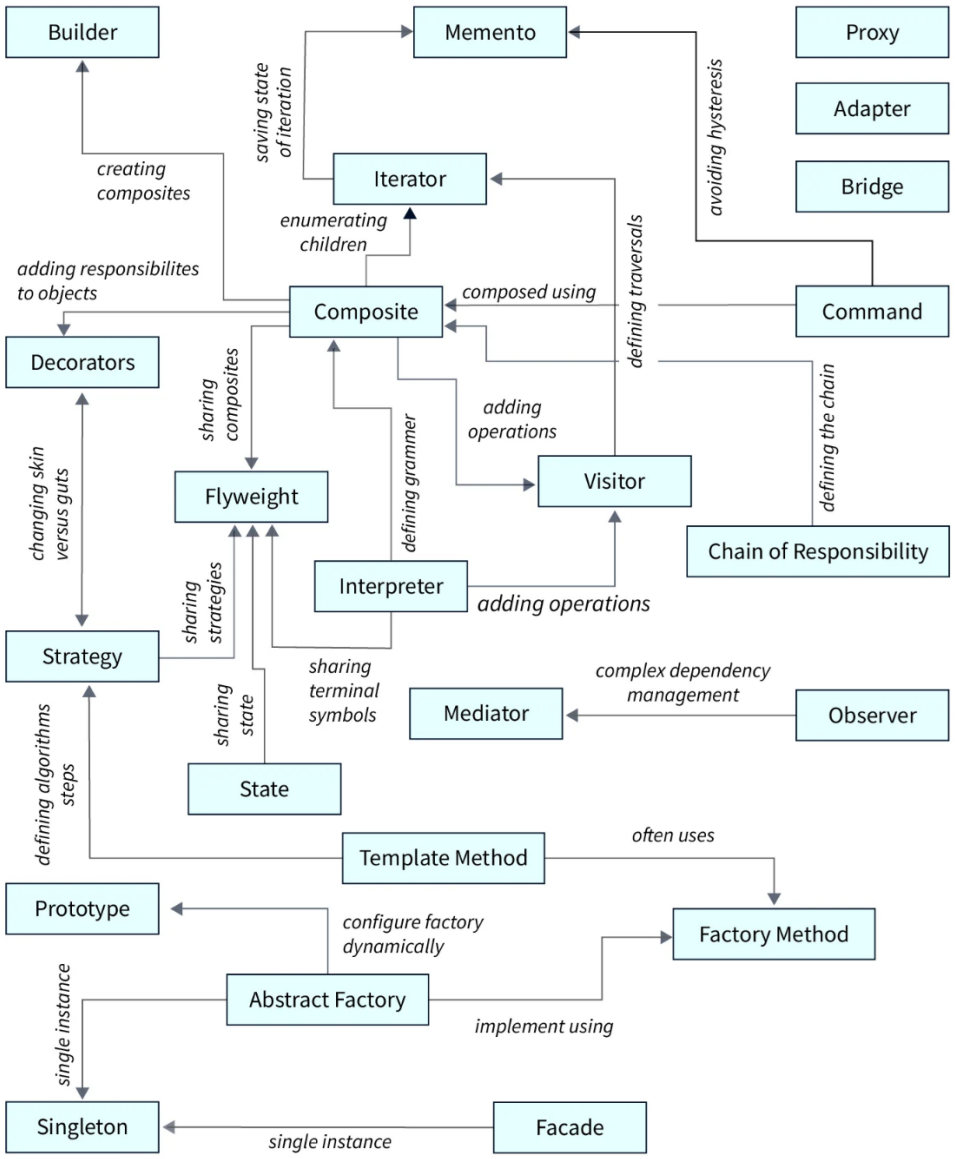
1) Use declarative formats for setup automation, to minimize time and cost for new developers joining the project;
2) Have a clean contract with the underlying operating system, offering maximum portability between execution environments;
3) Are suitable for deployment on modern cloud platforms, obviating the need for servers and systems administration;
4) Minimize divergence between development and production, enabling continuous deployment for maximum agility;
5) And can scale up without significant changes to tooling, architecture, or development practices.

The Twelve Factors

```
1)  Codebase            : One codebase tracked in revision control, many deploys
2)  Dependencies        : Explicitly declare and isolate dependencies
3)  Config              : Store config in the environment
4)  Backing services    : Treat backing services as attached resources
5)  Build, release, run : Strictly separate build and run stages
6)  Processes           : Execute the app as one or more stateless processes
7)  Port binding        : Export services via port binding
8)  Concurrency         : Scale out via the process model
9)  Disposability       : Maximize robustness with fast startup and graceful shutdown
10) Dev/prod parity     : Keep development, staging, and production as similar as possible
11) Logs                : Treat logs as event streams
12) Admin processes     : Run admin/management tasks as one-off processes
```

Design patterns

1) Creative: Helpful in creating objects
2) Structural: Helpful in dealing with the composition of objects
3) Behavioral: Helpful in defining the interactions between objects and distributing responsibility



Anti patterns[91]

1) Spaghetti Code
2) Golden Hammer
3) Boat Anchor
4) Dead Code
5) God Object and God Class
6) Copy and Paste Programming

Clean Code Fundamentals

1) understandability
2) readability
3) changeability
4) extensibility
5) maintainability

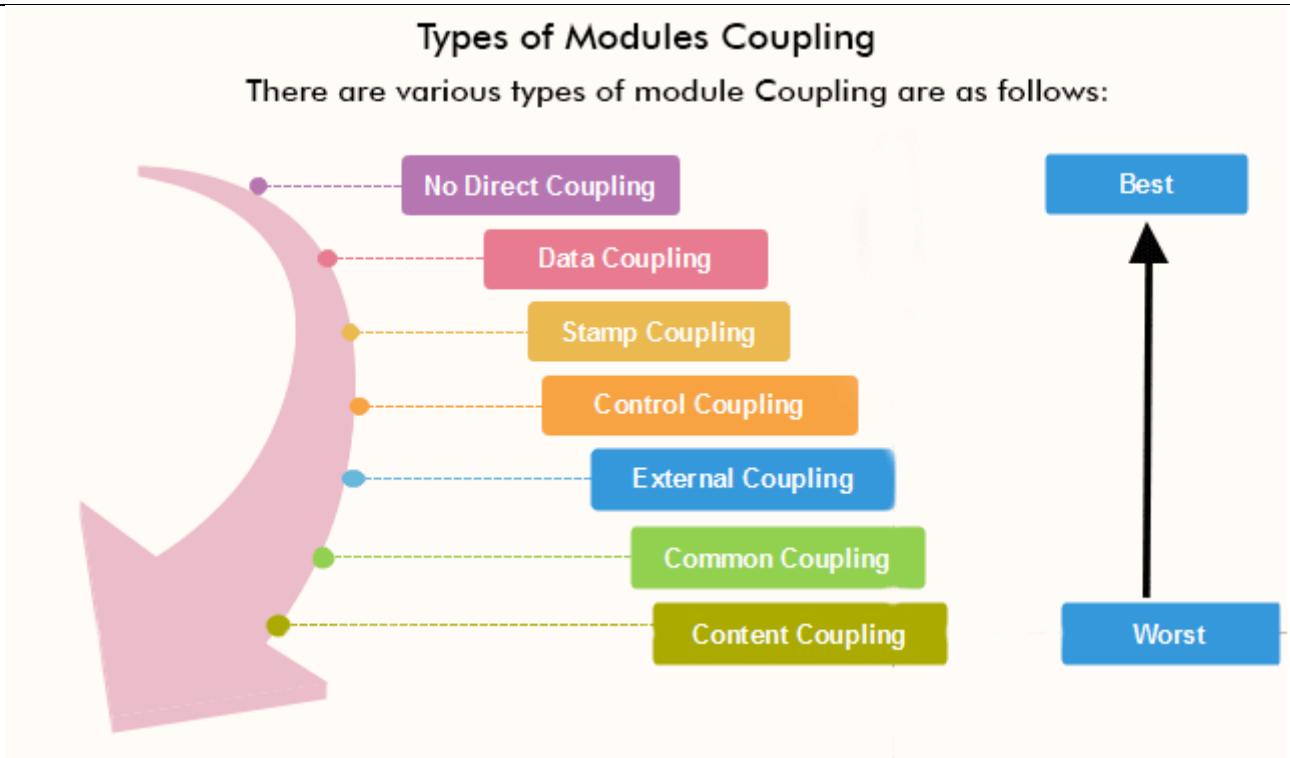
Design is good when

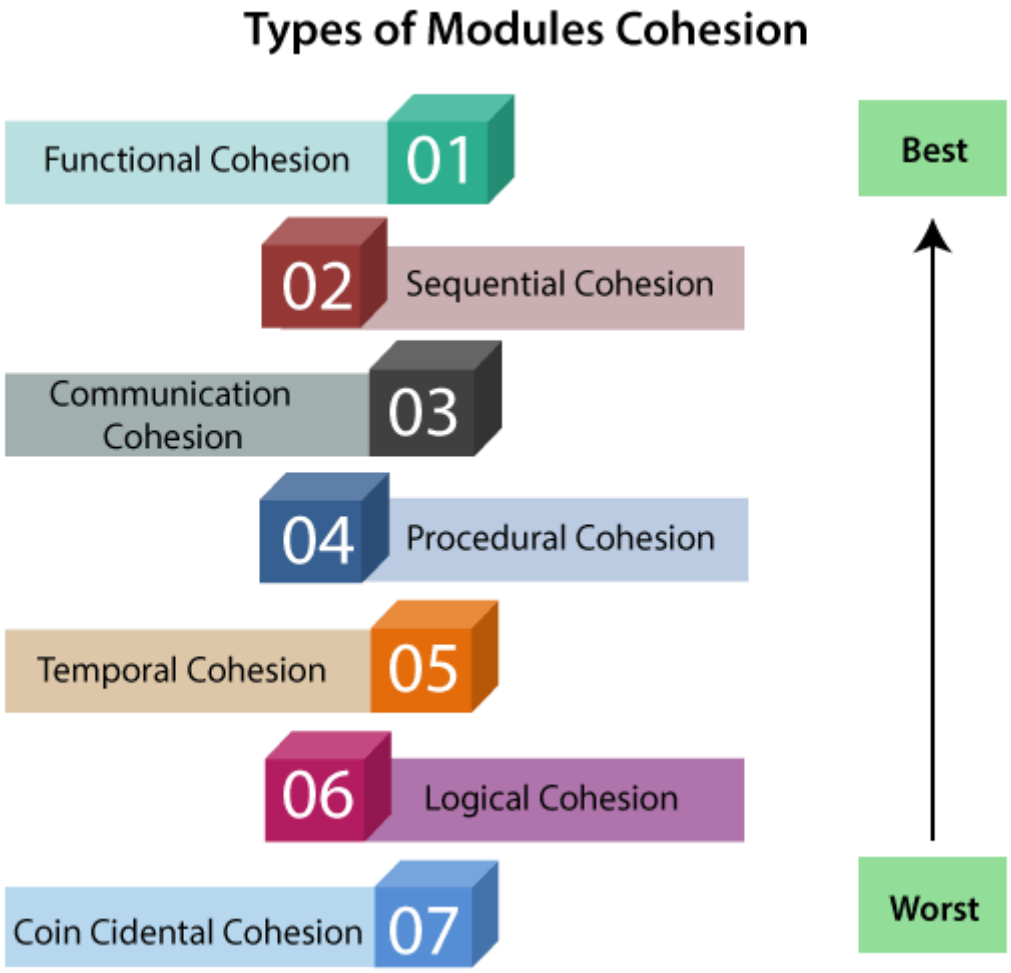
- it breaks down the complexity of the software into manageable and simple problems.
- small interfaces were defined.
- the components are decoupled.
- the components have clearly defined responsibilities.
- the software is maintainable.
- the software can be easily changed and extended.
- the software is stable.
- bugs can be fixed quickly.
- the software is reusable in other software projects.
- the code is understandable.

Criteria for good design

1) Correctness
    a. Fulfilment of requirements
    b. Playback of all functions of the system model
    c. Ensuring the non-functional requirements
2) Comprehensibility
    a. Self-explanatory code and design
    b. Good documentation
3) **High cohesion**[92] (Intra-Module Binding: relationship within the module.)
4) **Low coupling (**Inter-Module Binding: relationships between modules)
5) Reusability
6) Customizability
7) No cyclic dependencies

91 - https://www.scaler.com/topics/software-engineering/anti-patterns/
92 - https://www.javatpoint.com/software-engineering-coupling-and-cohesion


Types of Modules Coupling
There are various types of module Coupling are as follows:
No Direct Coupling
Data Coupling
Stamp Coupling
Control Coupling
External Coupling
Common Coupling
Content Coupling
Best
Worst


Types of Modules Cohesion
Functional Cohesion 01
02 Sequential Cohesion
Communication Cohesion 03
04 Procedural Cohesion
Temporal Cohesion 05
06 Logical Cohesion
Coin Cidental Cohesion 07
Best
Worst

Symptoms of bad design

1) Never-touch-running-code Syndrome
    a. Developers are afraid to change code.
    b. Many workarounds, code is developed around it.
    c. Changes have unknown and undetected side effects.
2) Small change in requirements leads to big changes in code
3) Reuse through code duplication (Copy-Paste)
    a. Developers chase after the places that need to be changed.
    b. The more code the more difficult it becomes to keep track of duplicates.
    c. Errors have to be patched several times at different places.
4) Cyclic relations between artifacts
    a. Artifacts that are cyclically coupled cannot be tested individually.
    b. Artifacts that are used in different cycles often play several roles, which makes them difficult to understand.
    c. Artifacts that are used in different cycles cannot be exchanged easily.

Concurrency vs. Parallelism

Concurrency:

The recognition that we can divide up a computation (an algorithm) into separate pieces where the order of execution of the pieces doesn't matter. Concurrency is a function of the algorithm you use to solve a problem. This also means that even if you create a concurrent program, if you run it on a single processor it will still run serially.

Parallelism:

The mechanism used to execute a program on a particular machine or machine architecture in order to improve the performance of the program. Parallelism allows a concurrent program to execute on many different processors and thus potentially improve the overall performance of the program.

Monolithic architecture challenges

1) Large code base : This is a scenario where the code lines outnumber the comments by a great margin. As components are interconnected, we will have to bear with a repetitive code base.
2) Too many business modules : This is in regard to modules within the same system.
3) Codebase complexity : This results in a higher chance of code-breaking due to the fix required in other modules or services.
4) Complex code deployment : You may come across minor changes that would require whole system deployment.
5) One module failure affecting the whole system: This is in regard to modules that depend on each other.
6) Scalability : This is required for the entire system and not just the modules in it.
7) Intermodule dependency : This is due to tight coupling.
8) Spiraling development time : This is due to code complexity and interdependency.
9) Inability to easily adapt to a new technology: In this case, the entire system would need to be upgraded.

Recommendations

The app needs multiple instances to handle load.
The app needs SSL/TLS termination (for Network Load Balancers).
The app needs to redirect HTTP requests to HTTPS.
The app needs to serve multiple domains.
The app needs to serve static resources, e.g. jpeg files.

1) Use A Reverse Proxy
    a. HAProxy
    b. Nginx
2) Kubernetes (Auto Scaling)
3) Capacity Planning for Production
4) Running Multiple Instances

Programming paradigms[93]

1) Action
2) Array-oriented[94]
3) Automata-based
4) Concurrent computing
    a. Actor-based
    b. Choreographic programming
    c. Multitier programming
    d. Relativistic programming
    e. Structured concurrency
5) Data-driven[95]
6) Data-oriented[96]
7) **Declarative (contrast: Imperative)**
    a. **Functional**
        i. **Functional logic**
        ii. **Purely functional**
    b. Logic
        i. Abductive logic
        ii. Answer set
        iii. Concurrent logic
        iv. Functional logic
        v. Inductive logic
    c. Constraint
        i. Constraint logic
            1. Concurrent constraint logic
    d. Dataflow
        i. Flow-based
        ii. Reactive
            1. Functional reactive
    e. Ontology
    f. Query language
8) Differentiable
9) Dynamic/scripting
10) Event-driven
11) Function-level (contrast: Value-level)
    a. Point-free style
        i. Concatenative
12) Generic
13) **Imperative (contrast: Declarative)**
    a. **Procedural**
    b. **Object-oriented**
14) Intentional
15) Language-oriented[97]
    a. Domain-specific
16) Literate
17) Macroprogramming
18) Metaprogramming
    a. Automatic
        i. Inductive programming
        ii. Program synthesis
    b. Reflective
        i. Attribute-oriented[98]
    c. Macro

93 - https://en.wikipedia.org/wiki/Programming_paradigm
94 - https://en.wikipedia.org/wiki/Array_programming
95 - https://en.wikipedia.org/wiki/Data-driven_programming
96 - https://en.wikipedia.org/wiki/Data-oriented_design
97 - https://en.wikipedia.org/wiki/Language-oriented_programming
98 - https://en.wikipedia.org/wiki/Attribute-oriented_programming

    d. Template
19) Natural-language programming
20) Non-structured (contrast: Structured)
    a. Array
21) Nondeterministic
22) Parallel computing
    a. Process-oriented[99]
23) Probabilistic
24) Quantum
25) Set-theoretic
26) Stack-based
27) Structured (contrast: Non-structured)
    a. Block-structured
    b. Object-oriented[100]
        i. **Agent-oriented[101]**
        ii. Class-based
        iii. Concurrent
        iv. Prototype-based
        v. By separation of concerns:
            1. **Aspect-oriented[102]**
            2. **Role-oriented[103]**
            3. **Subject-oriented[104]**
    c. Recursive
28) Symbolic
29) Value-level (contrast: Function-level)

[99] - https://en.wikipedia.org/wiki/Process-oriented_programming
[100] - https://en.wikipedia.org/wiki/Object-oriented_programming
[101] - https://en.wikipedia.org/wiki/Agent-oriented_programming
[102] - https://en.wikipedia.org/wiki/Aspect-oriented_programming
[103] - https://en.wikipedia.org/wiki/Role-oriented_programming
[104] - https://en.wikipedia.org/wiki/Subject-oriented_programming

پیوست ۲ : فناوریها/تکنولوژیها

FUNDAMENTALS

1) Custom providers
2) Asynchronous providers
3) Dynamic modules
4) Injection scopes
5) Circular dependency
6) Module reference
7) Lazy-loading modules
8) Execution context
9) Lifecycle events
10) Platform agnosticism
11) Testing

TECHNIQUES

1) Configuration
2) Database
3) Mongo
4) Validation
5) Caching
6) Serialization
7) Versioning
8) Task scheduling
9) Queues
10) Logging
11) Cookies
12) Events
13) Compression
14) File upload
15) Streaming files
16) HTTP module
17) Session
18) Model-View-Controller
19) Performance (Fastify)
20) Server-Sent Events
21) Sharding

SECURITY

1) Authentication
2) Authorization
3) Encryption and Hashing
4) Helmet
5) CORS
6) CSRF Protection
7) Rate limiting

GRAPHQL

1) Resolvers
2) Mutations
3) Subscriptions
4) Scalars
5) Directives
6) Interfaces
7) Unions and Enums
8) Field middleware
9) Mapped types
10) Plugins
11) Complexity
12) Extensions
13) CLI Plugin

14) Generating SDL
15) Sharing models
16) Federation
17) Migration guide

WEBSOCKETS

1) Gateways
2) Exception filters
3) Pipes
4) Guards
5) Interceptors
6) Adapters

OPENAPI

1) Types and Parameters
2) Operations
3) Security
4) Mapped Types
5) Decorators
6) CLI Plugin

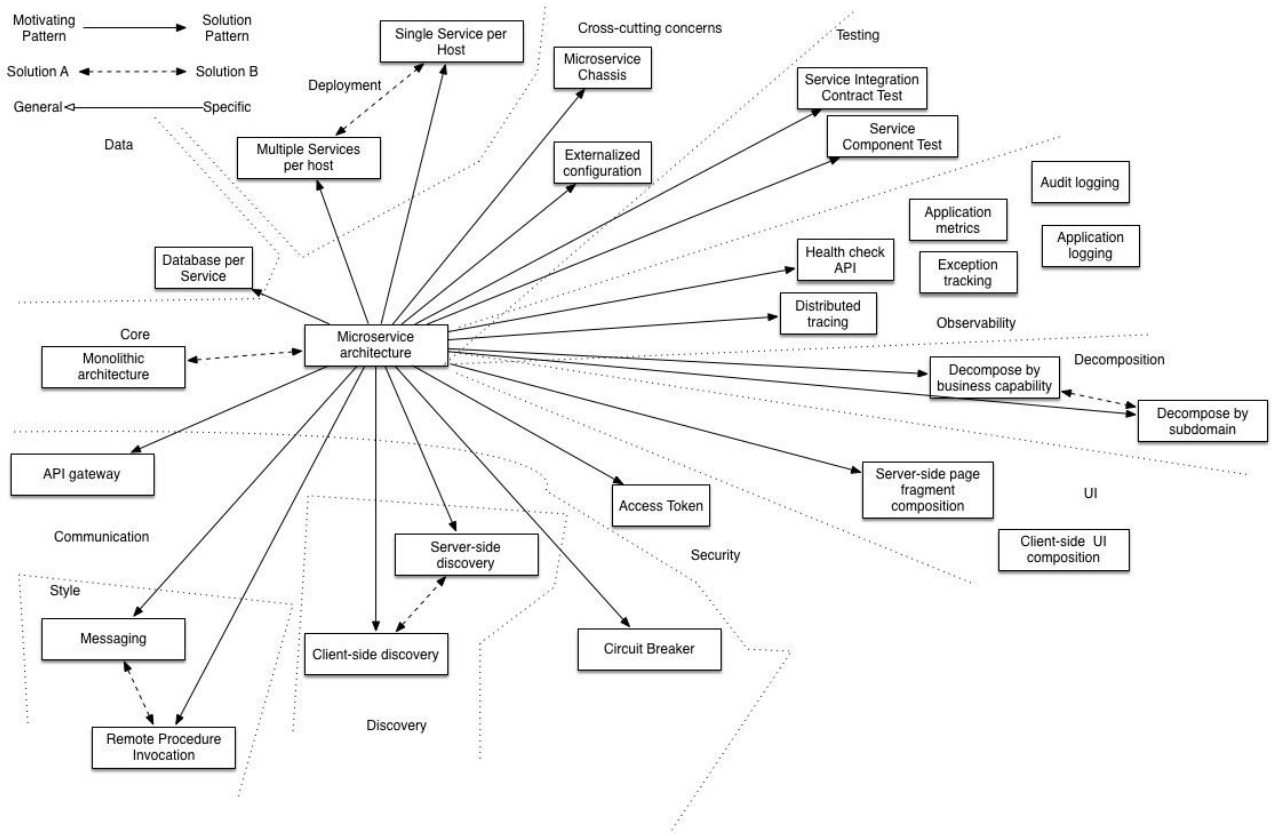
Other RECIPES

1) REPL
2) CRUD generator
3) SWC (fast compiler)
4) Passport (auth)
5) Hot reload
6) MikroORM
7) TypeORM
8) Mongoose
9) Sequelize
10) Router module
11) Swagger
12) Health checks
13) CQRS[105] (Command and Query Responsibility Segregation)
14) Compodoc
15) Prisma
16) Serve static
17) Commander
18) Async local storage
19) Automock
20) Auto-scaling: The ability of an application to automatically scale its resource usage in response to changes in demand.
21) Load balancers
22) Service Mesh: Implementing network communication between microservices using a service mesh, such as Istio or Linkerd, to provide features such as traffic management, security, and observability. A configurable infrastructure layer for microservices applications that makes communication between service instances flexible, fast, and reliable.
23) CI/CD pipelines: A series of automated processes for continuously building, testing, and deploying code.
24) Serverless computing: Computing model in which the cloud provider manages the infrastructure and automatically allocates resources capacity needed to run applications.
25) Circuit Breaker: Implementing a circuit breaker pattern to prevent cascading failures and improve the resilience of microservices.
26) API Gateway: Implementing an API gateway to manage and secure access to microservices and provide a single-entry point for external consumers.
27) Blue-Green Deployment: Deploying new application versions to a parallel environment and switching traffic between the old and new versions to minimize downtime.
28) Canary Release: Gradually releasing new application versions to a subset of users to test and validate before releasing to the entire user base.
29) Strangler Pattern: Gradually replacing parts of a monolithic application with microservices over time to minimize disruption and risk.

[105] - https://learn.microsoft.com/en-us/azure/architecture/patterns/cqrs
https://martinfowler.com/bliki/CQRS.html

MICROSERVICES[106]

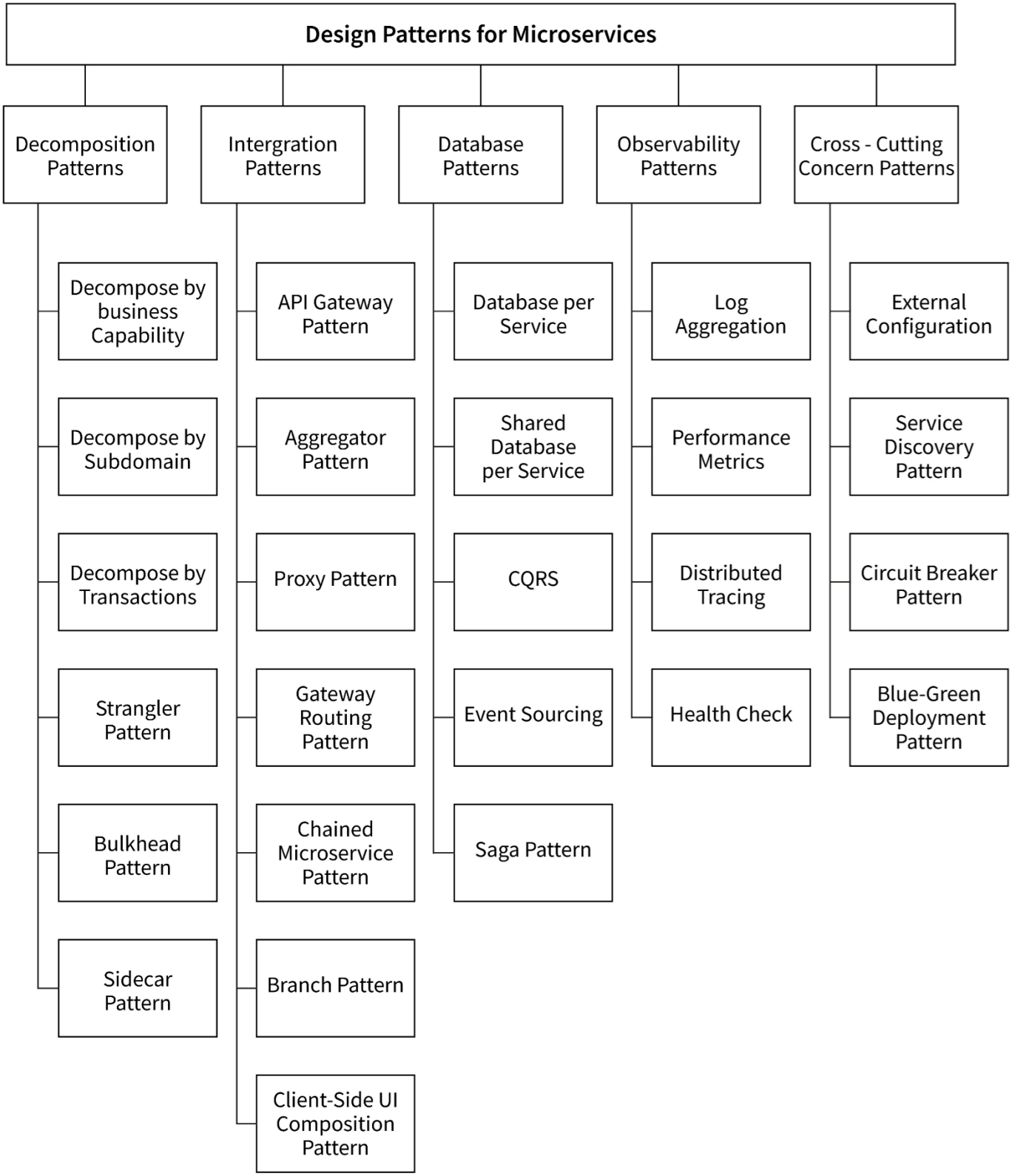
1) Patterns[107]
2) SAGA[108] distributed transactions pattern
    a. Choreography
    b. Orchestration
3) API Gateway
4) Redis
5) MQTT
6) NATS
7) RabbitMQ
8) Kafka
9) gRPC
10) Custom transporters
11) Exception filters
12) Pipes
13) Guards
14) Interceptors



[106] - https://microservices.io/patterns/microservices.html , https://eventuate.io/exampleapps.html
https://cloud.google.com/learn/what-is-microservices-architecture

[107] - https://microservices.io/patterns

[108] - https://microservices.io/patterns/data/saga.html
https://learn.microsoft.com/en-us/dotnet/architecture/microservices
https://learn.microsoft.com/en-us/azure/architecture/reference-architectures/saga/saga
https://dotnet.microsoft.com/en-us/download/e-book/microservices-architecture/pdf


Design Patterns for Microservices
Decomposition Patterns
Intergration Patterns
Database Patterns
Observability Patterns
Cross - Cutting Concern Patterns
Decompose by business Capability
Decompose by Subdomain
Decompose by Transactions
Strangler Pattern
Bulkhead Pattern
Sidecar Pattern
API Gateway Pattern
Aggregator Pattern
Proxy Pattern
Gateway Routing Pattern
Chained Microservice Pattern
Branch Pattern
Client-Side UI Composition Pattern
Database per Service
Shared Database per Service
CQRS
Event Sourcing
Saga Pattern
Log Aggregation
Performance Metrics
Distributed Tracing
Health Check
External Configuration
Service Discovery Pattern
Circuit Breaker Pattern
Blue-Green Deployment Pattern