

# Engenharia de Software

- É a aplicação de uma forma sistemática, disciplinada e quantificável ao desenvolvimento, operação e manutenção de software
- É uma disciplina de engenharia preocupada com todos os aspectos da produção de software
- É uma forma sistemática de desenhar, construir e manter software como "utilitário, levaria" (mais forte, mais rápido, mais barato e melhor)

1. Requisitos do Software
2. Desenvolvimento do Software
3. Construção do Software
4. Teste do Software
5. Manutenção do Software
6. Gestão da Configuração do Software
7. Gestão da Engenharia do Software
8. Processo da Engenharia do Software
9. Métodos e Modelos da Engenharia de Software
10. Qualidade do Software
11. Prática Profissional da Engenharia de Software

## Manifesto para o Desenvolvimento Ágil de Software:

1. Indivíduos e interação > processos e ferramentas
2. Software funcional > documentação abrangente
3. Colaboração com o cliente > negociação contínua
4. Responder à mudança > seguir um plano

## Processos de Software e Modelos de Processo:

1. Waterfall
2. Iterativo
3. Espiral
4. Processo Tracional Unificado
5. Scrum
6. Programação Extrema (XP)

# Processos de Software

O que é Engenharia de Software?

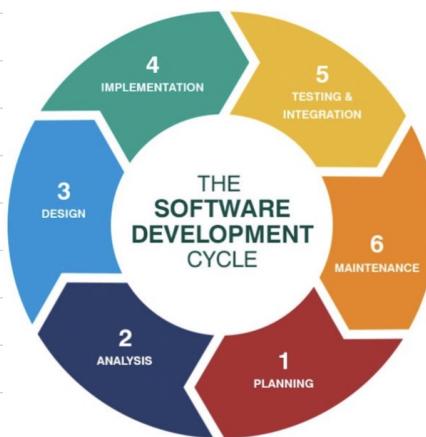
- (1) A aplicação de forma sistemática, disciplinada e quantificável ao desenvolvimento, operação e manutenção de software; isto é, a aplicação da engenharia ao software
- (2) O estudo das abordagens como em (1)

- A engenharia de software preocupa-se com o desenvolvimento de forma eficaz em custo e tempo de software de alta qualidade de forma previsível, particularmente para sistemas de larga escala

Processo de Software: conjunto estruturado de atividades necessárias para desenvolver um sistema de software

Atividades do Processo:

1. Especificação: define o que o sistema deve fazer
2. Design e Implementação: define a organização do sistema e implementá-lo
3. Validação: verifica se o sistema faz o que o cliente pretende
4. Evolução: muda o sistema em resposta a mudanças nas necessidades do cliente

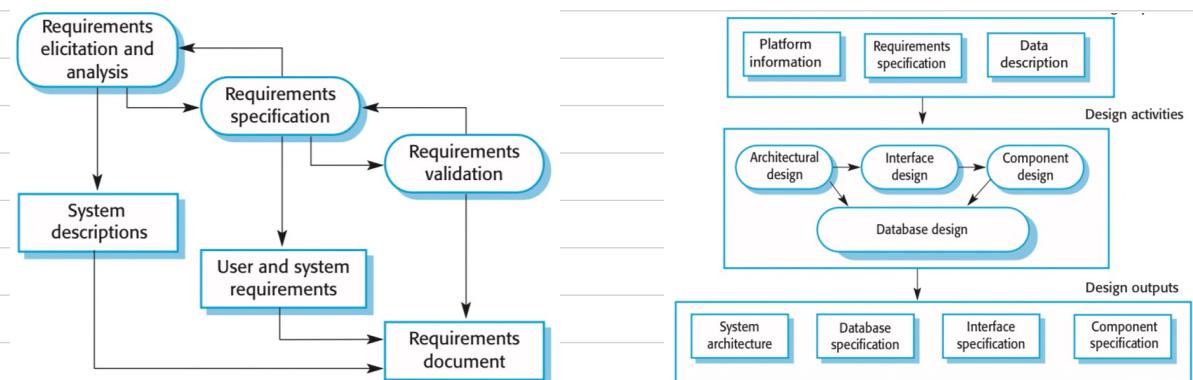


Porque é que não necessários processos definidos?

1. Eficiência
2. Consistência
3. Base para Melhorar

Processos "Plan-Driven": processos em que todas as atividades do processo são planeadas antecipadamente e o progresso é medido contra o planeado

Processo Ágil: processos em que o planeamento é incremental e é fácil mudar o processo para refletir mudanças nos requisitos do cliente

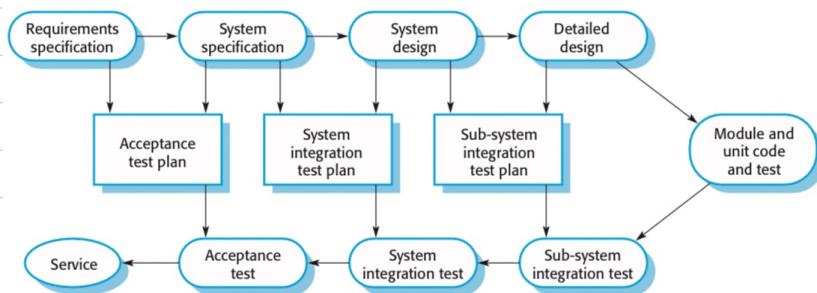


O que fazer?

Como fazer?

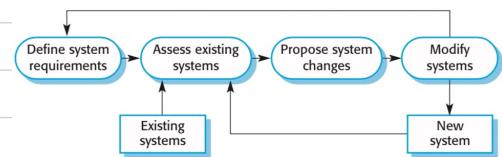
Verificação: mostra que o sistema está conforme com a especificação

Validação: mostra que o sistema cumpre os requisitos e necessidades do cliente



1. Teste Unitário: componentes individuais são normalmente testados pelos seus desenvolvedores
2. Teste de Integração: realizado quando os componentes estão integrados, para detectar problemas de integração
3. Teste do Sistema: o sistema como um todo é testado, normalmente por uma equipe independente, com o foco em propriedades emergentes (desempenho, usabilidade, ...)
4. Teste de Alergias: o sistema é testado (sob a responsabilidade do consumidor) com os dados do consumidor para verificar se as suas necessidades são satisfeitas

## Mantenção/Evolução de Software:

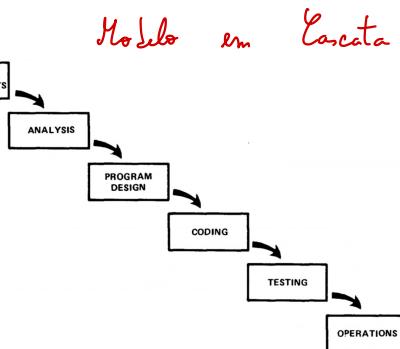


1. Correção: resolução de bugs
2. Adaptativa: adaptação a novas plataformas
3. Perfetta: novas funcionalidades

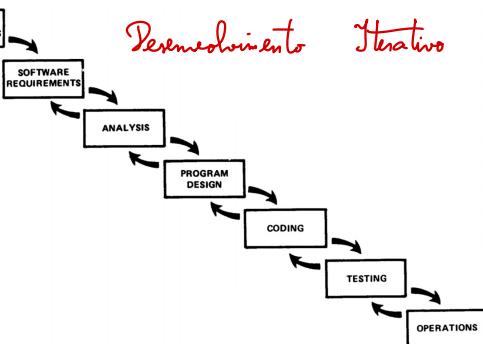
## Modelos de Processo de Software

1. Modelo em Lixeira: modelo "plan-driven"; fases de especificação e desenvolvimento separadas e distintas
2. Desenvolvimento Incremental: especificação, desenvolvimento e validação estão intercaladas; pode ser "plan-driven" ou ágil
3. Integração e Configuração: o sistema é construído a partir de componentes configuráveis; pode ser "plan-driven" ou ágil
4. Prototipagem de Software: não um modelo, mas uma forma de lidar com incerteza

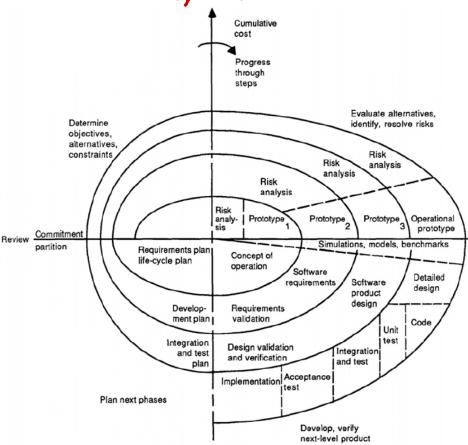
## Modelo em Cascata



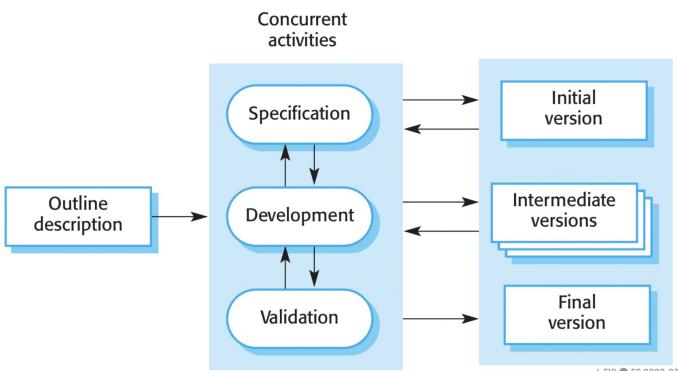
## Desenvolvimento Iterativo



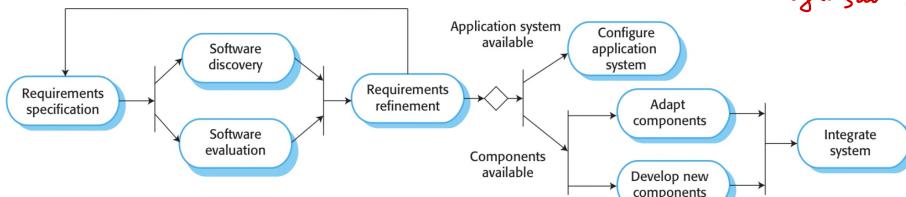
## Espiral



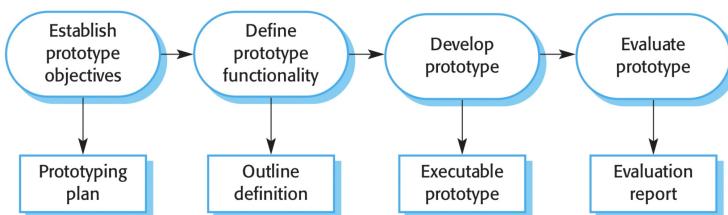
## Desenvolvimento Incremental



## Integração e Configuração

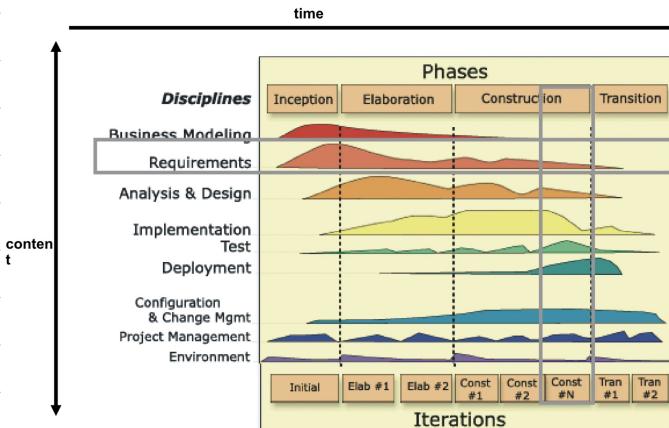


## Prototipagem de Software



## Rational Unified Process

- Processo iterativo e incremental
- UML
- Guiado por casos de utilização
- Centrado em arquitetura



### Fases:

1. Iniciação: estabelece o âmbito/límite do projeto, os critérios e casos de uso críticos, bem como exibi/ demonstra uma arquitetura candidata, estimar custo, tempo e riscos e preparar o ambiente de suporte ao projeto
2. Elaboração: define e valida a arquitetura, refina a missão, os casos de desenvolvimento, o ambiente de suporte e a arquitetura, bem como criar planos de iterações detalhados para a fase de construção
3. Construção: gestão e controle de recursos, otimização do processo, desenvolvimento e teste completos contra os critérios e avaliação do produto contra os critérios
4. Transição: executar planos de lançamento/produção, finalizar material de ação ao utilizador final, testar o produto final em ambiente de desenvolvimento, lança o produto, obter feedback dos utilizadores, aprimorar o produto com base no feedback e torná-lo disponível para o utilizador final

# Extreme Programming XP

- "Extreme Programming vira o processo convencional de software ao contrário. Em vez de planejar, analisar e burlhar para um futuro longínquo, os programadores em XP fazem todas essas atividades, num fôoco de cada vez, durante o desenvolvimento"
- Método Ágil

## Valores:

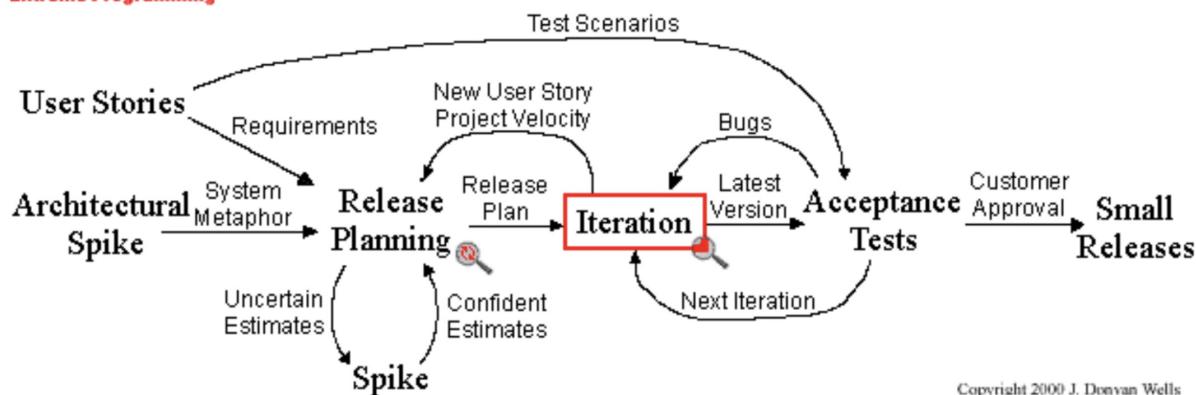
1. Comunicação: cliente no local, "user stories", programação em pares, código coletivo,...
2. Simplicidade: "fazer a coisa mais simples que pode possivelmente funcionar"
3. Feedback: feedback constante do cliente, testes unitários, iterações frequentes
4. Coragem: comunicar e aceitar feedback, deixar fora código (protótipo), refactoring

## Práticas:

1. Ágio de Planeamento: "user stories", estimativas, decisões do cliente e do desenvolvedor
2. Pequenos Lançamentos: lançar cedo e frequentemente, com novas funcionalidades de cada vez
3. Metáfora do Sistema: história que todos podem contar sobre o funcionamento do sistema
4. Design Simples: usar o design mais simples que faz o trabalho
5. Test-Driven Development: testar antes de adicionar uma funcionalidade visual
6. Refactoring: melhora a estrutura do código sem alterar o seu comportamento extensamente
7. Programação em Pares: um elemento escreve código, o outro critica
8. Código Coletivo: ninguém detém um módulo significativo, todos podem trabalhar nele
9. Integração Contínua: todas as mudanças são integradas no código pelo menos diariamente
10. Software Testável: "fresco e animado de malha, cansado e satisfeito à noite"
11. Cliente no Local: acena contínuo a um cliente real ao vivo
12. Standards de Código: todos programam para os mesmos standards



# Extreme Programming Project

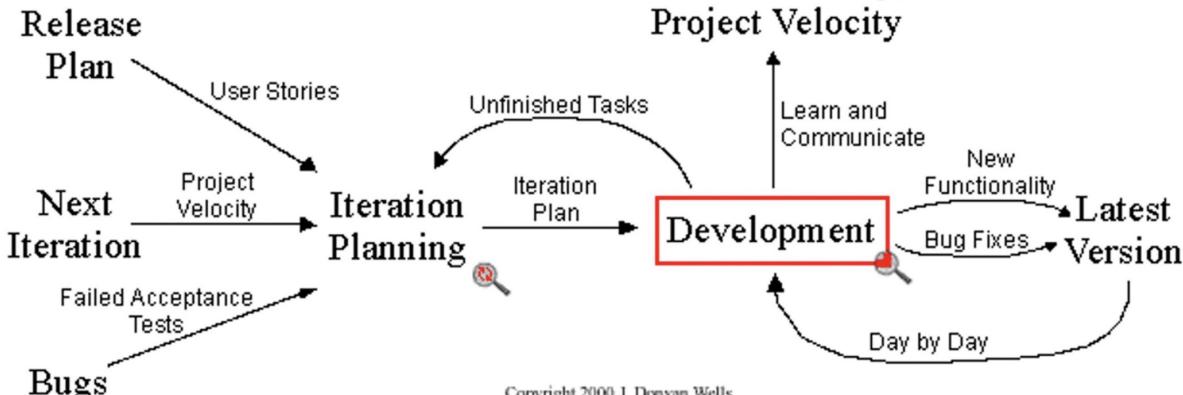


Copyright 2000 J. Donvan Wells



## Iteration

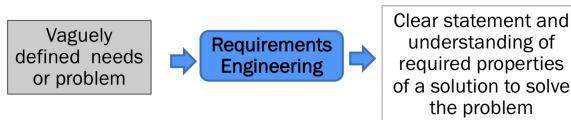
Zoom Out



Copyright 2000 J. Donvan Wells

Release planning: Release content & date

# Engenharia de Requisitos



Engenharia de Requisitos: o processo de estudar as necessidades do utilizador e do consumidor para chegar a uma definição dos requisitos do sistema, hardware or software

Requisito de Software: propriedade que deve ter um software desenvolvido ou adaptado para resolver um problema particular

Desafios:

1. Estabelecer comunicação e compreensão dos requisitos
2. Gerir requisitos em evolução (alterações/crescimento)
3. Desenfogar os requisitos

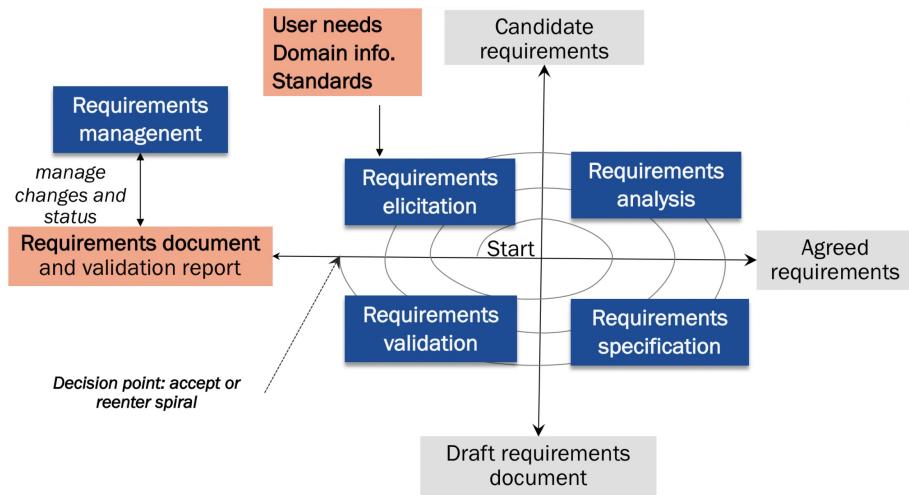
Requisitos Funcionais: descrevem as funções que o software tem de executar (capacidades)

Requisitos Não-Funcionais: requisitos que atuam para restringir a solução



Stakeholders: principais fontes de requisitos; pessoas que vão ser afetadas pelo sistema e que têm influência direta ou indireta na elaboração de requisitos

Ex: cliente; utilizadores; gestores; entidades reguladoras; mantenedores; organizações; ...



### Aatividade de Engenharia de Requisitos:

Elicitação: interação com stakeholders e outras fontes para descobrir requisitos

Análise: organização e avaliação da informação recolhida (completnude, consistência, clareza, ...) para chegar a uma lista de requisitos acordados

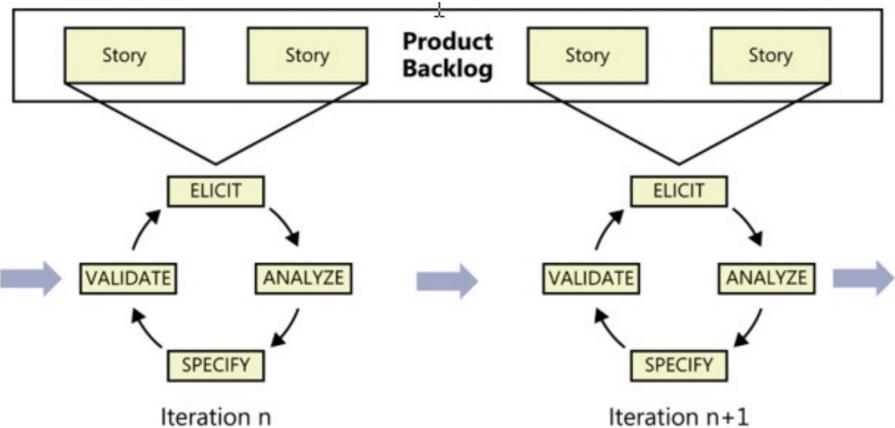
Especificação: produção da documentação dos requisitos, com um nível de detalhe adequado ao contexto

Validação: assegurar que os requisitos documentados permitem alcançar os objetivos negociais do projeto

## Artifícios de Engenharia de Requisitos:

1. Lista de requisitos
2. Modelos do sistema
3. Protótipos/Mockups da interface do utilizador
4. Testes de aceitação

## Engenharia de Requisitos em Métodos Ágeis



"Uma Story": forma leve de registar uma necessidade do software, com informação apenas suficiente (para triunfar, planejar e iniciar conversas)

Quem? O que? Porquê?

Independente Negociável Valioso Estimável Small Testável

Teste de Aceitação: casos de teste definidos para cliente decidirem se a implementação de um sistema ou de uma funcionalidade pode ser aceite (satisfazer requisitos/expectativas)

"Behavior-Driven Development": Dado [contexto inicial ou pré-condições]

Quando [evento ocorre]

Então [alguma resultado ou pós-condições]

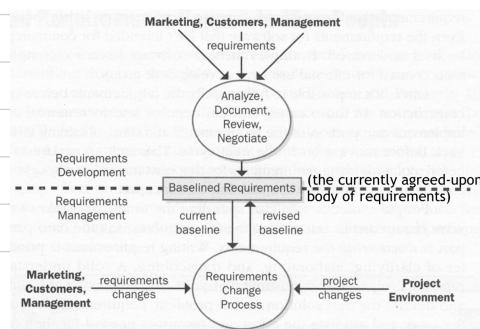
**Protótipo da Interface do Usuário:** versão inicial/primitiva do sistema, mais barata, mais fácil e mais rápida de desenvolver, mas com funcionalidade limitada, que dá uma antevista de como o sistema final vai apresentar e funcionar, usada em engenharia de requisitos para (1) abordar áreas de maior incerteza e risco de mau entendimento e (2) valida requisitos previamente identificados e identificam novos

**Protótipo "Show-and-Tell":** arregua foco nos requisitos em vez de limitações da implementação

**Protótipo Evolucionário:** apropriado para desenvolvimento rápido e iterativo de aplicações com forte envolvimento da utilizador final

### Características de Qualidade:

1. Adequação à Funcionalidade
2. Eficiência de Desenvolvimento
3. Fiabilidade
4. Usabilidade
5. Compatibilidade
6. Manutenção
7. Portabilidade
8. Agilidade

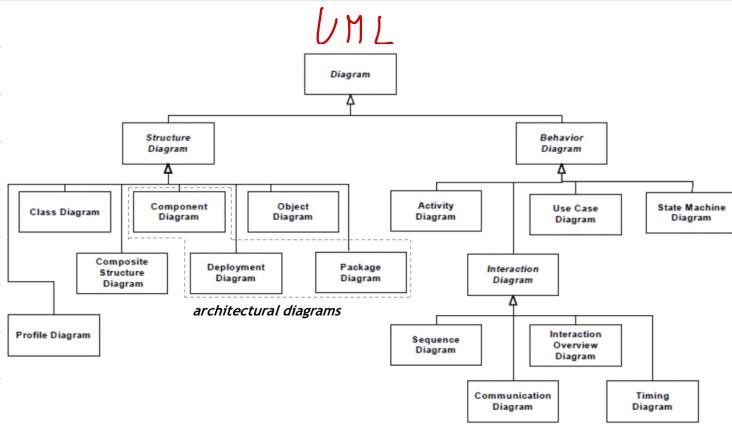


# Casos de Utilização e Modelação de Domínios com UML

Modelo de Sistema: representação simplificada de um sistema a partir de uma certa perspectiva que ajuda a reduzir a ambiguidade da linguagem natural

Modelo de Caso de Utilização: para definir fronteiras e alcance do sistema e organizar requisitos funcionais

Modelo de Domínio: para organizar o vocabulário e capturar requisitos de informação no domínio do problema

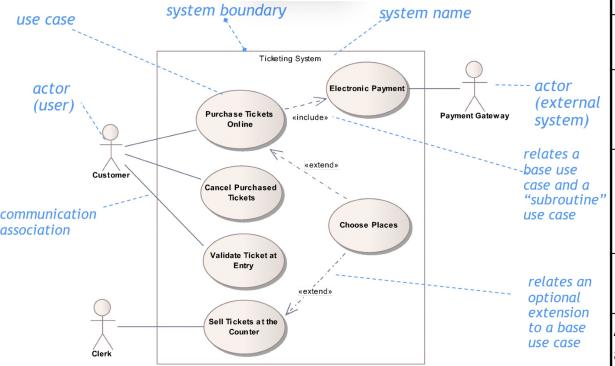


Modelo de Caso de Utilização: diagrama(s) de casos de utilização + documentações

Atores: profissionais utilizadores ou sistemas externos

Caso de Utilização: funcionalidades ou reuniões do sistema como percecionadas pelos utilizadores; tipos de interações entre os atores e o sistema

Relações: participação, generalização, inclusão, extensão



Name	Use case name, showing the actor's intent.
Actor	Name of the actor that initiates the use case.
Description	Brief description highlighting the utility (possibly including results and responsibilities).
Preconditions	Initial conditions (usually about the initial state of the system and actors and input parameters) to be able to execute the use case successfully.
Postconditions	Effects of use case execution, usually in the form of outputs produced and changes in the state of the system and actors (particularly those not visible in the flow of events).
Normal flow	Textual description (numbered list) of the of normal events (actions of the system and actors).
Alternative flows and exceptions	Alternative or exceptional flows of events, described in relation to the normal flow.

RUP:

1. Encontra atores
2. Encontra casos de utilização
3. Estrutura o modelo de casos de utilização
4. Detalha os casos de utilização

Modelo de Domínio: Diagramas de classes + documentação

Notation	Description	Notation	Description
ClassName	Class	—	Association
«interface» InterfaceName	Interface	→	Navigable association (object reference)
InterfaceName		▷	Generalization (extends in Java)
ClassName Param	Parameterized class (generic)	→▷	Realization (implements in Java) Alternative notation:
PackageName	Package	.....>	Dependency (particularly, between packages)
instanceName: ClassName	Object (instance of a class)	◇—	Aggregation ("parts" may exist independently of a "whole")
		◆—	Composition ("parts" only exist within a "whole")
		○—	Nesting (inner classes)

Category	Notation	Description
Modifiers	<i>italic</i>	Abstract classes and operations
	<u>underlined</u>	Static attributes and operations
	/name	Derived (calculated) attribute
Visibility	+ - # ~	public, private, protected, package
Multiplicity	* , 0..*, 1, 0..1, 1..*	0 or more, 0 or more, 1, 0 or 1, 1 or more
Constraints	{constraint}	Restricts the valid instances; is usually defined in the context of a class.

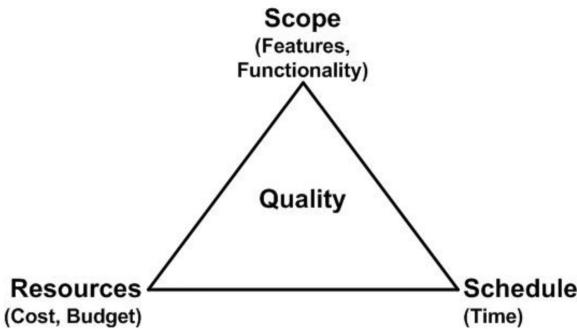
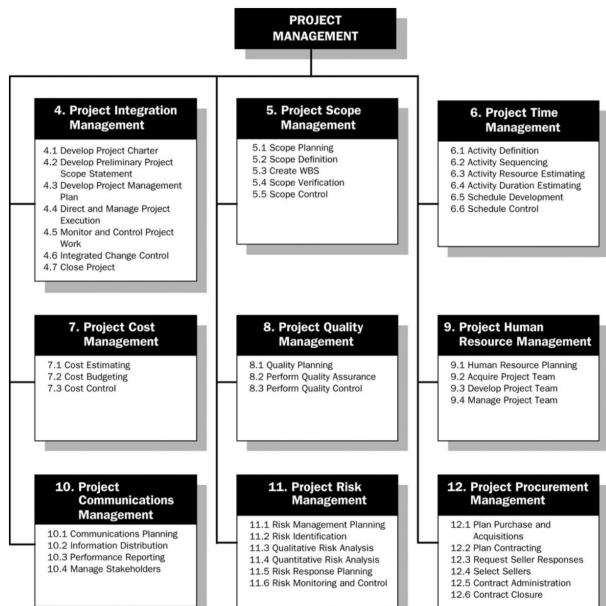
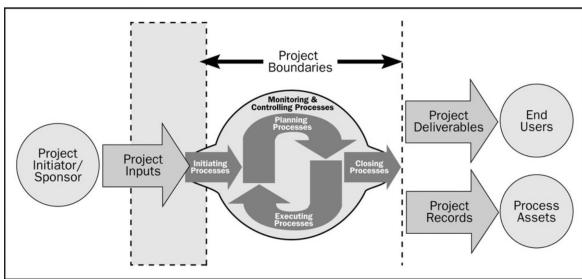
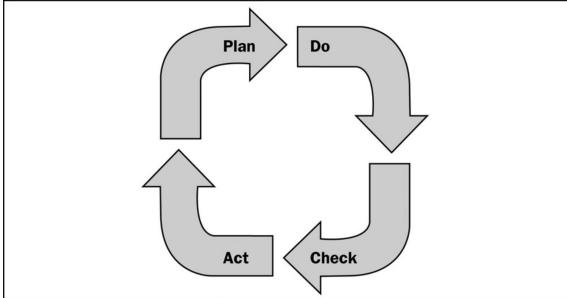
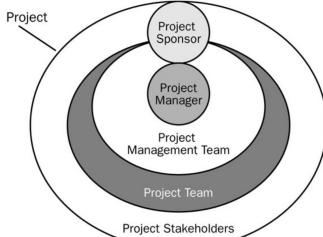
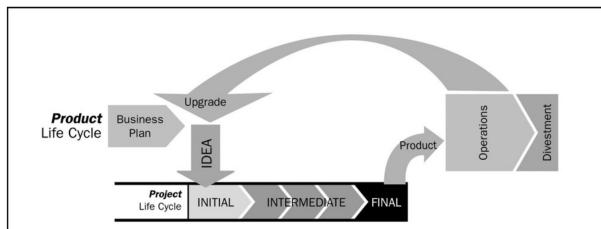
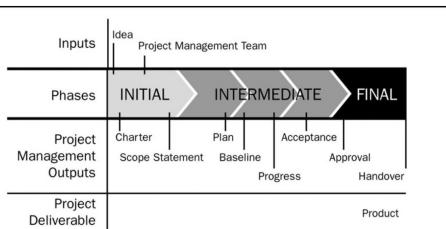
## Relações:

1. Generalização: entre um conceito mais geral e um conceito mais especializado
  - deve ser possível dizer "é um (caso especial de)";
  - atores especializados herdam casos de utilização do genérico
  - casos de utilização especializados herdam o comportamento, significado e atores do genérico e podem adicionar comportamento
2. Extensão: indicam comportamentos adicionados condicionalmente
  - permitem salientar funcionalidades opcionais e distinguir-las do que é essencial/obrigatório
3. Inclusão: quando vários casos de utilização partilham algum comportamento comum, esse comportamento comum pode ser reforçado e descrito num novo caso de utilização que é incluído nos primeiros

## Restrições de Integridade:

1. Restrições de domínio (no conjunto de valores de atributos)
2. Restrições de unicidade (chaves)
3. Restrições relativas a ciclos em associações
4. Restrições relativas a tempo
5. Restrições que definem elementos derivados
6. Restrições de existência (de objetos ou valores)
7. Restrições genéricas de negócio

# Gestão de Projetos de Software



## Variáveis:

1. Recursos: variável menos eficaz para ajustar (custo, pessoal, orçamento)
2. Tempo: variável mais dolorosa para ajustar
3. Âmbito: variável mais eficaz para ajustar (funcionalidades)
  - a. Largura: o que está incluído
  - b. Profundidade: refinar
4. Qualidade

Iterativo: execução repetida de ciclos de processos aninhados/embalados, em que cada iteração fornece ponto de aprimoramento e de feedback

Incremental: o sistema é construído em estágios progressivos, em que cada iteração adiciona funcionalidades e melhorias e os incrementos são sistemas a funcionar

Tarefas: agrupa atividades de tipos similares com preferência para completa tarefas em série - modelo da cascata

Concorrente e Paralelo: as atividades ocorrem oportunisticamente, ocorrendo atividades de todos os tipos ao mesmo tempo e sendo o completamento parcial a norma

Planeamento Predefinido: criação de planos comprensivos baseados em atividades, execução das atividades definidas para seguir o plano e gestão/controlé conforme o plano

Planeamento Ágil: criação de um conjunto priorizado de "delivrables", execução oportunística de atividades para criar "delivrables" e gestão via feedback e adaptações

Equilíbrio do Projeto num Processo Ágil: gestão sustentável de recursos (equipes estavam, ritmo constante), fixo do tempo e adaptativa do âmbito

Desenvolvimento Heróico: enfase nos indivíduos - as atividades são atribuídas a indivíduos e os resultados do projeto dependem fortemente do desempenho individual

Desenvolvimento Colaborativo: enfase nas equipes - as equipes auto-organizam as atividades para cumprir objetivos, mandam capacidades diversas e mitigam riscos "Kephali"

"Command and Control Strategy": as decisões são feitas por autoridades centrais, as atividades delegadas e controladas pelo gestor

"Facilitation and Empowerment Strategy": as decisões não são feitas pelos mais informados, as atividades aceitas e a equipe auto-gere-se e adapta-se num ambiente de apoio

Iteração 0: trabalho útil para a equipe do projeto, não para o consumidor - trabalho de arquitetura, treino tecnológico, documento de requisitos, ...

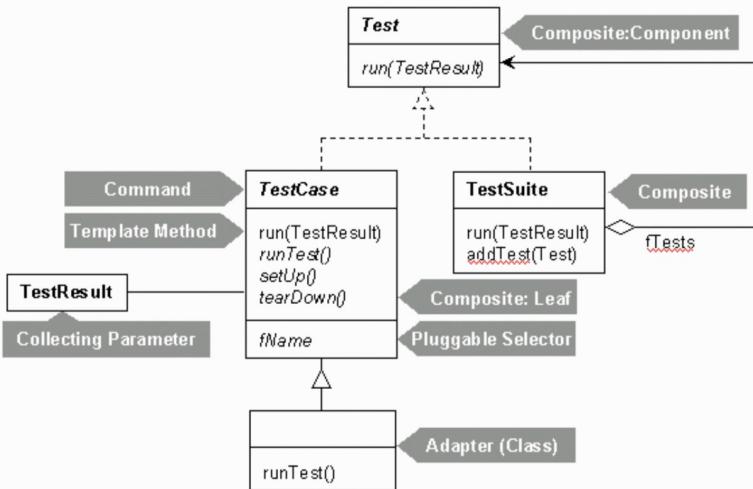
Iterações 1-N: planeadas atribuindo funcionalidades a cada iteração para a duração do projeto, ajudando na sensação de fluxo de projeto, determinação de data, etc.

Planos de Iterações {  
1. Completo: plano completo com funcionalidades atribuídas a cada iteração  
2. 2 iterações: plano da próxima iteração e depois tudo a seguir  
3. 1 iteração: plano iteração por iteração

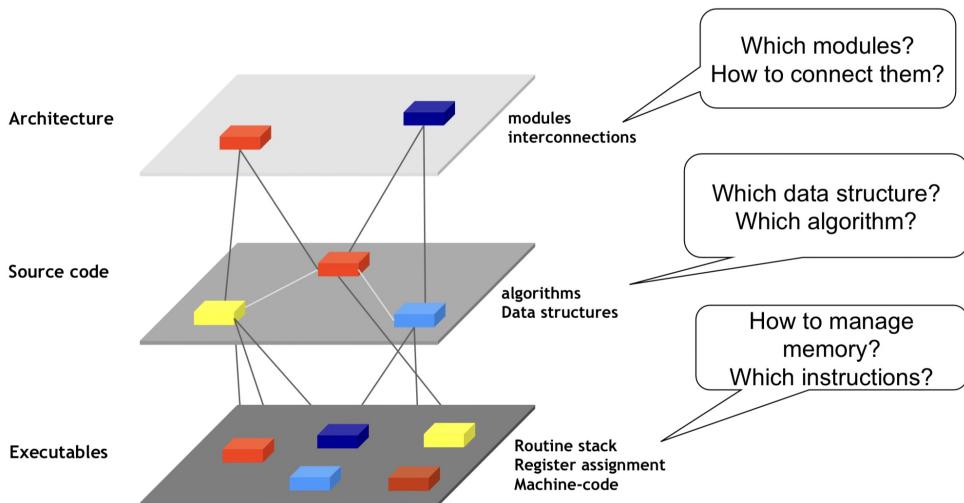
Plano do Próxima Iteração:

1. Atribuições: constui a lista de atividades para implementar cada funcionalidade
2. Re-estimativa: re-estimativa do esforço de trabalho baseada na avaliação <sup>ajustes</sup> e ✓
3. Atribuição: membros da equipe discutem-se para funcionalidades/atividades com base nas suas capacidades e/ou desejos

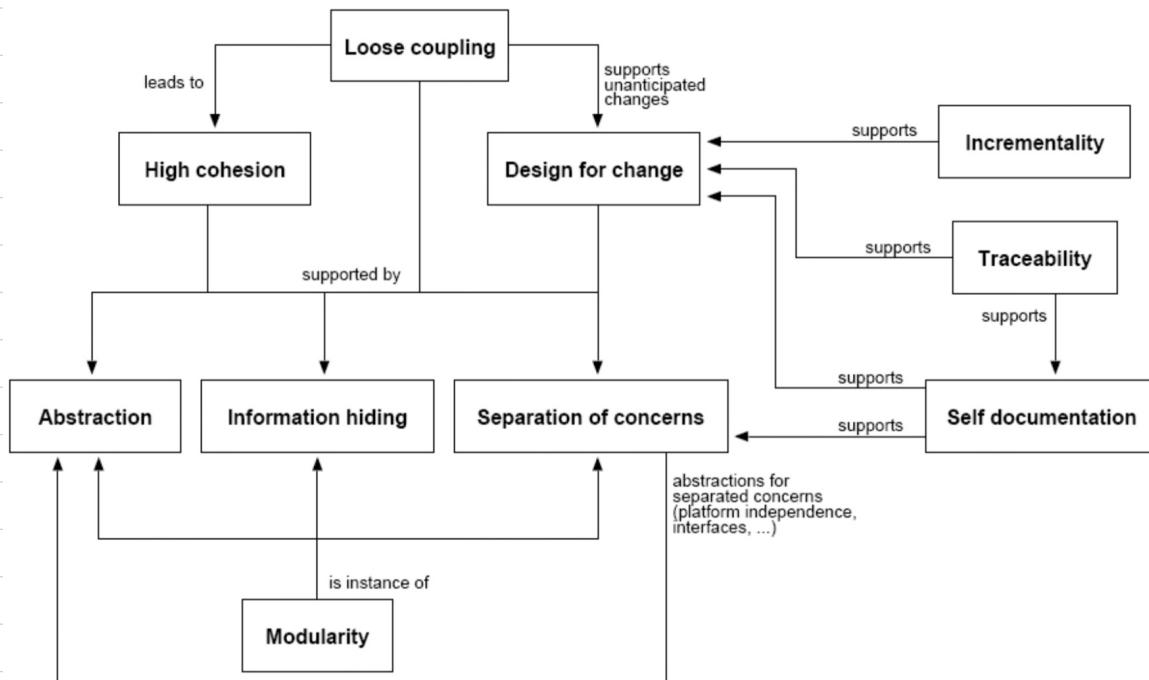
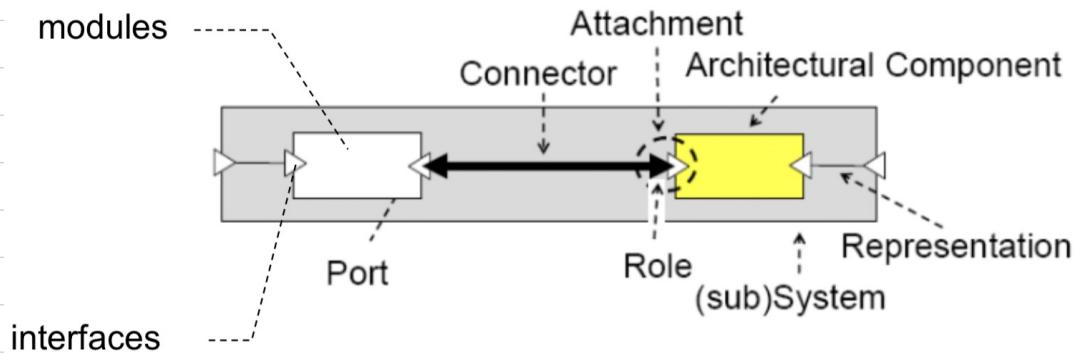
# Arquitetura de Software



*Desafio:* arquitetar e desenhar software de grande escala e resistente ao tempo



**Arquitetura de Software:** organização fundamental de um sistema, incorporando os seus componentes, as suas relações uns aos outros e ao ambiente, bem como os princípios que governam o seu design e evolução



## Padões de Arquitetura:

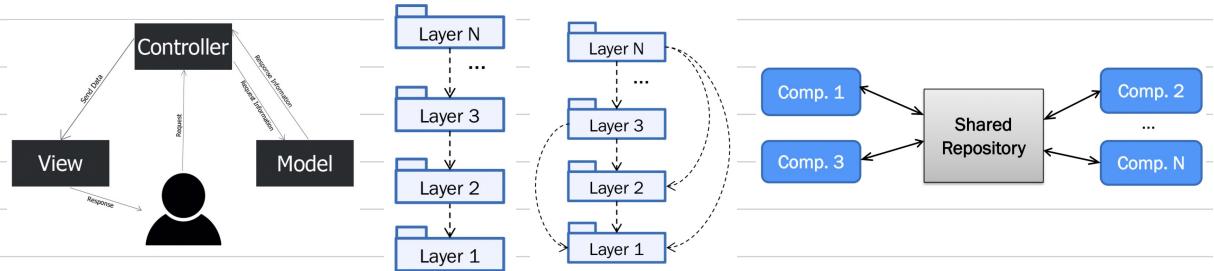
1. Model - View - Controller (MVC): separa apresentação (V) e interação (C) dos dados/estado (M) da aplicação ao estruturar o sistema em três partes lógicas que interagem entre elas
2. Pipes e Filtros (Fluxo de Dados): organiza o sistema como um conjunto de componentes que processam dados (filtros), ligados de modo que os dados fluem entre componentes para processamento ("pipes")
3. Arquitetura em Camadas: organiza o sistema num conjunto de camadas, cada uma delas agrupa funcionalidades relacionadas e fornece serviços para a camada acima

**Estática:** cada camada só pode interagir com a camada diretamente abaixo  
**Relaxada:** cada camada pode interagir com qualquer camada inferior

4. Repositórios (centrados em dados): toda a informação no sistema é gerida num repositório central que está acessível a todos os componentes do sistema ou sub-sistemas; componentes/subsistemas não interagem diretamente, só através do repositório

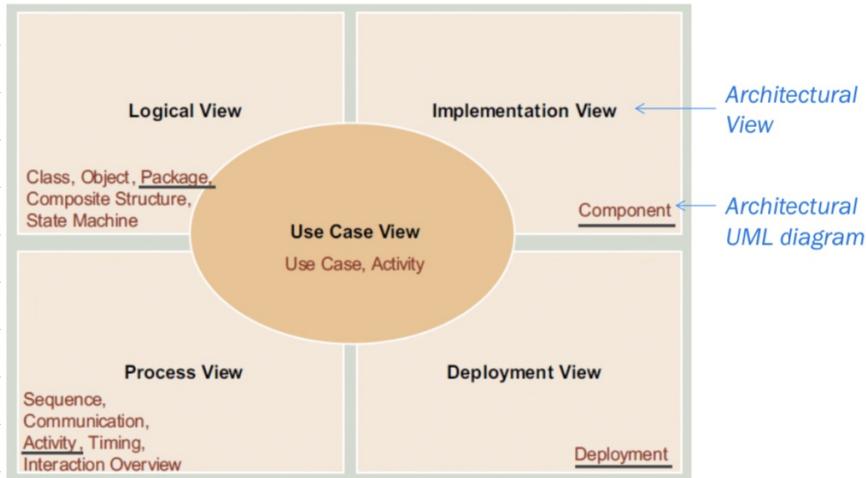
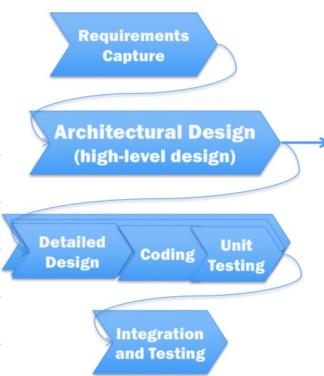
**Passivo:** armazena regras dos componentes de forma passiva

**Ativo:** alterações no repositório ativam a execução de componentes



## Outputs de Design Arquitetural:

1. Documento de Decisões de Arquitetura
2. Modelos de Arquitetura
3. Protótipos de Arquitetura
4. Plano de Teste de Integração

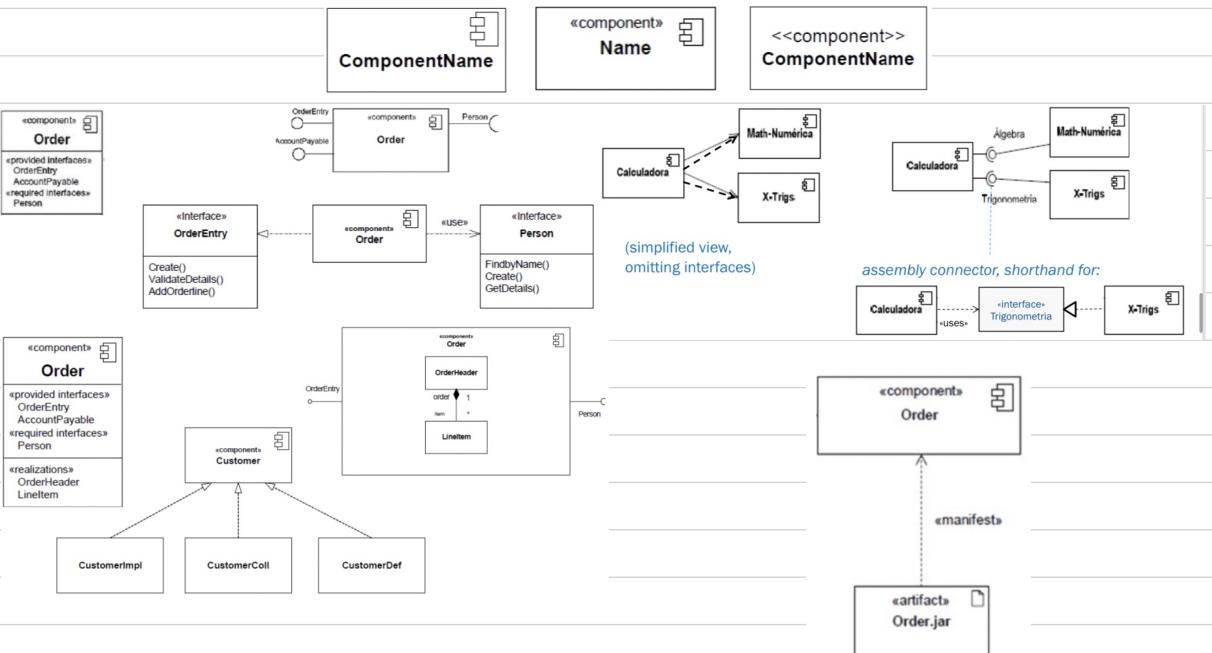


<b>Logical View:</b> Package diagrams (and others)	<b>Implementation View:</b> Component diagrams
Shows logical packages* and their relationships	Shows software components and dependencies among them
*division of responsibilities, independently of allocation to sw components or hw nodes	
<b>Process View:</b> Activity diagrams (and others)	<b>Deployment View:</b> Deployment diagrams
Shows processing steps, data/object stores, data/object-flows, and opportunities for parallelization	Shows hardware nodes, communication relationships and software artifacts deployed on them

# Diagramas de Componentes

**Componente:** representa uma parte modular do sistema que encapsula os seus conteúdos e cuja manifestação é substituível no seu ambiente.

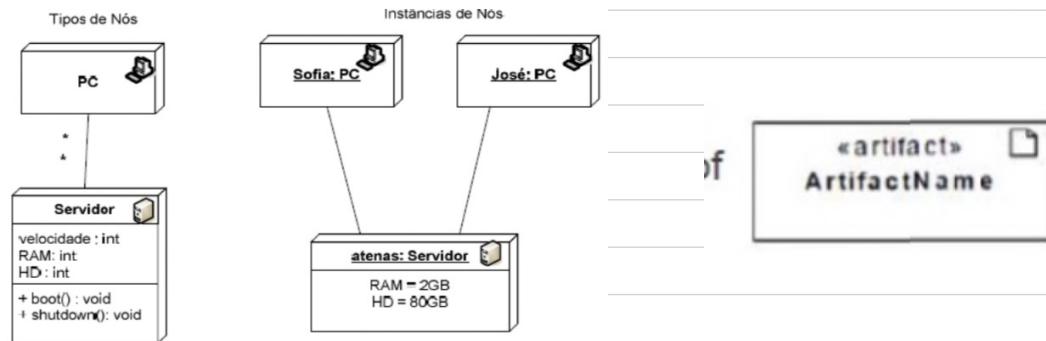
- Um componente define o seu comportamento em termos de interfaces fornecidas (realizadas) e interfaces necessárias (usadas)
- Os componentes não devem depender diretamente de outros componentes, mas sim de interfaces implementadas por outros componentes
- O comportamento de um componente é realizado (implementado) pelas suas classe internas.
- Os componentes manifestam-se fisicamente como artefactos



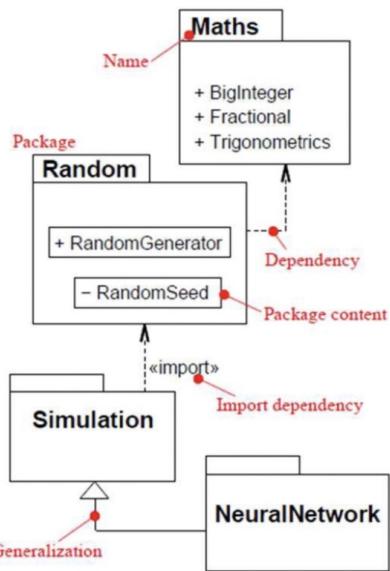
## Diagramas de Utilização

Nós: recursos computacionais onde artefactos podem ser utilizados

Artifícios: elementos de informação física usados ou produzidos por um processo de desenvolvimento de software ou pela instalação/operação de um sistema



## Diagramas de Paços



## Desenvolvimento Iterativo

1. Cada funcionalidade desejada pelo cliente é escrita como "user story"
2. Os desenvolvedores estimam o "tamanho" de cada user story (esforço para implementar)

**Velocidade do Projeto:** tamanho total feito por iteração

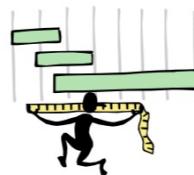
3. Dadas as estimativas e a velocidade do projeto, o cliente ordena que stories implementar

**Clientes decidem...**

- Ambito (user stories)
- Prioridades
- Contéudo das Releases
- Datas de Entrega

**Desenvolvedores decidem...**

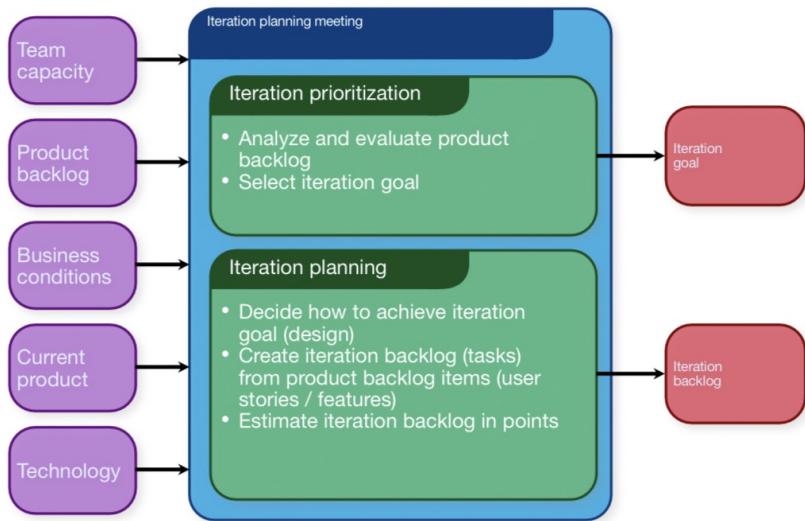
- Estimativas de Esforço
- Consequências
- Processo
- Horários Detalhados de Tarefas



"As práticas apoiam-se mutuamente. As fraguas de uma não cobetas pelas forças de outra."

**Planeamento de Releases:** clientes e desenvolvedores decidem sobre o conteúdo das releases (user stories) e data das próximas releases

**Planeamento de Tarefas:** os desenvolvedores decidem sobre as tarefas de desenvolvimento para a próxima iteração (iterações posteriores aos releases)



### Fecho de Iteração:

1. Âmbito de incrementos bem definido e identificado nos documentos, código e planejamento
2. Existe "release" no git hub
3. "Release" pode ser facilmente usada pelo utilizador final
4. O incremento claramente adiciona valor para o utilizador final
5. "User stories" adicionadas, estimadas, atribuídas, revisadas e documentadas
6. Retrospetiva da Iteração: fizemos bem? fizemos diferente? puzzles?

Retrospetivas de Projeto: "Uma retrospetiva é uma oportunidade para os participantes aprendem como melhorar. O foco é em aprender e não em encontrar culpas"

NÃO É ... autóptica | ingrat | regred | destrutivo | casa à loura

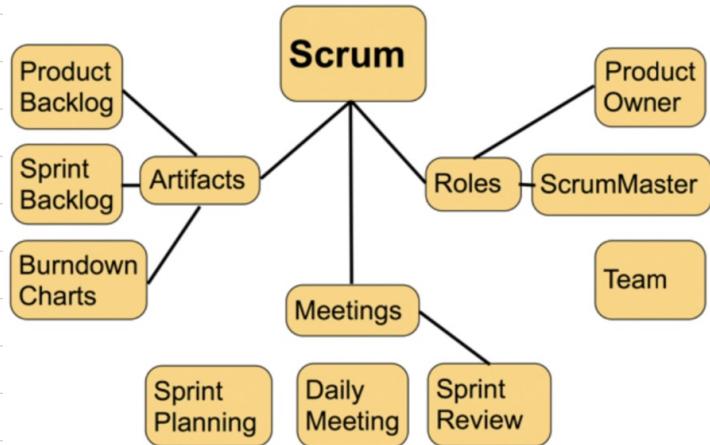
E depois?

1. Relatórios de Retrospetivas
2. Padrões
3. Planos de Ação

# Scrum

Fundações:

- Confiança
- Foco
- Transparência
- Coragem
- Respeito
- Empenho
- Confiança



PO - Product Owner  
 SM - ScrumMaster  
 T - Team  
 C - Customer

Input from End-Users, Customers, Team and Other Stakeholders

① PO  
 Product Owner

Product Backlog (Features)

Sprint Backlog (Stories)

Sprint Planning

24 hrs

**Sprint**  
1-4 Weeks

② PO  
 Product Owner

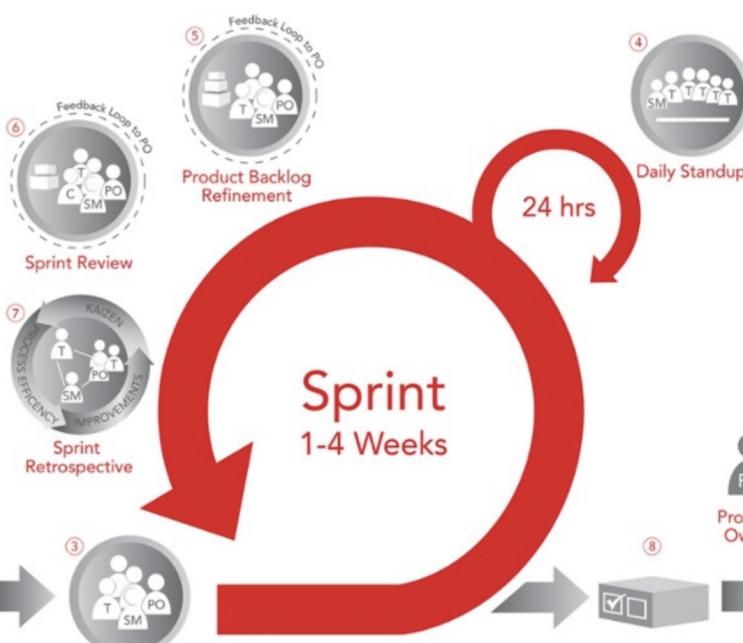
③

④

Customer-Ready Product Increment

⑤

Incremental Product Release



## Equipas:

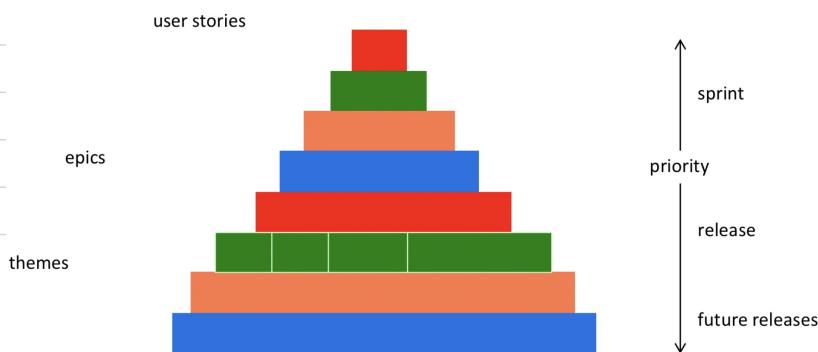
- 5-9 pessoas
- multi-funcional
- os membros devem ser a tempo inteiro
- as equipas organizam-se a elas próprias
- os membros não podem mudar entre sprints

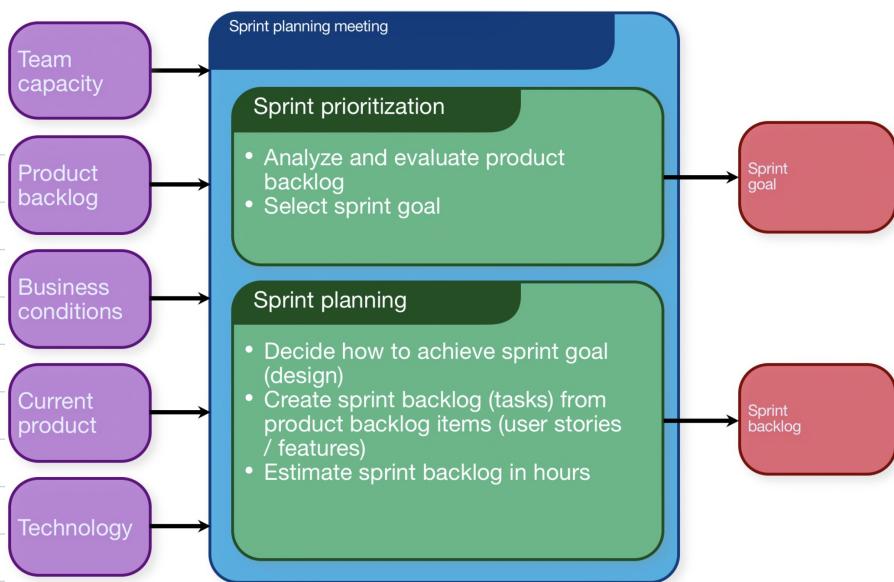
## "Scrum Master":

- representa a gestão para o projeto
- responsável por promover os valores e as práticas do Scrum
- remove impedimentos
- assegura que a equipa é completamente funcional e produtiva
- permite cooperação próxima entre todos os papéis e funções
- protege a equipa de interferências exteriores

## "Product Owner":

- define as funcionalidades do produto
- decide as datas de lançamento e o conteúdo
- é responsável pela rentabilidade do produto
- prioriza funcionalidades de acordo com o valor de mercado
- ajusta funcionalidades e prioriza todas as iterações conforme necessário
- aceita ou rejeita resultados do trabalho





**"Sprint Review"**: a equipa apresenta o que conseguiram durante o sprint

- Normalmente sob a forma de uma demonstração de funcionalidades ou apresentação
- Informal (2 horas sem slides)
- Toda a equipa participa
- "Convida o mundo"

## Scrum: Práticas chave

1. Organizar o trabalho em ciclos pequenos
2. A gestão não interrompe a equipa durante o ciclo de trabalho
3. A equipa reports to the client, não ao gestor
4. A equipa estima quanto tempo o trabalho vai demorar
5. A equipa decide quanto trabalho conseguem fazer numa iteração
6. A equipa decide como fazer o trabalho na iteração
7. A equipa faz o seu próprio desenpenho
8. Define objetivos de trabalho antes do início de cada ciclo
9. Define objetivos de trabalho através de user stories
10. Sistematicamente remover impedimentos

# Engenharia de Software

Simplicidade: "Fazer a coisa mais simples que pode funcionar" (KISS)

YAGNI: "Não vai precisar disso" ↔ Os requisitos não mudam

Programação a Puro: dois programadores, numa máquina — o "driver" escreve códigos enquanto o "navigator" critica

Vantagem: processo de revisão informal; maior qualidade

Refactoring: melhorar a estrutura do código sem mudar o comportamento visual

Testes Práticos: antes de acrescentar uma funcionalidade, escrever um teste para ela!

Testes Unitários: testar pequenas partes de funcionalidades conforme são escritas

Testes de Aceitação: especificações pelo consumidor para verificar o funcionamento global do sistema

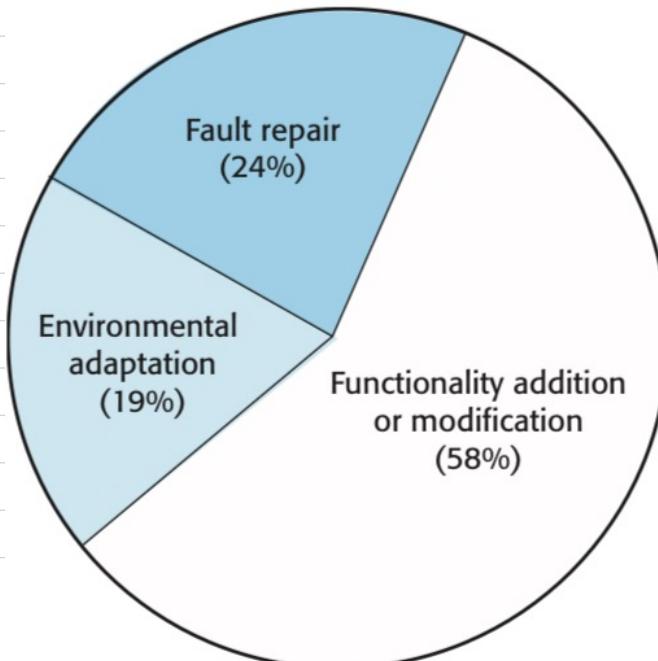
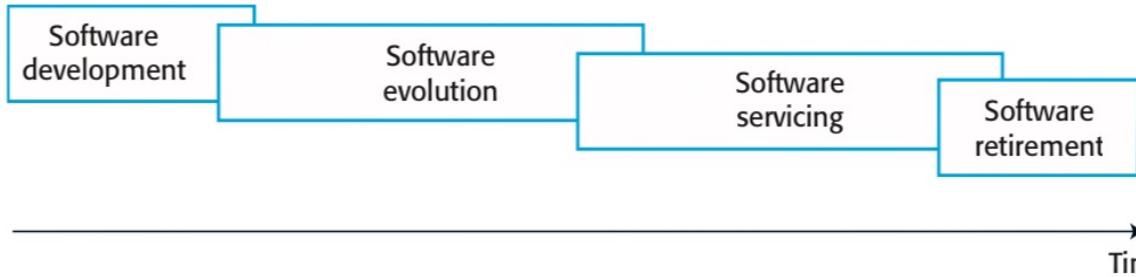
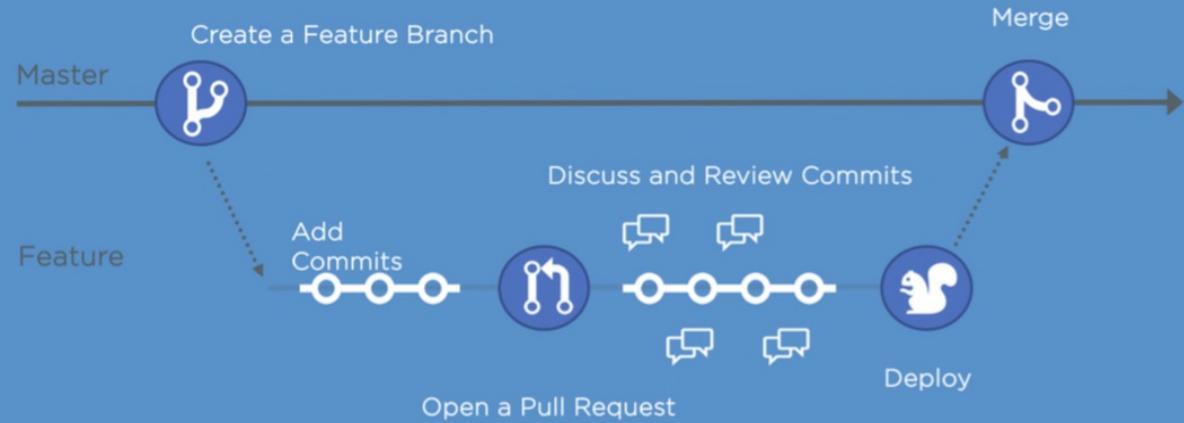
Código Coletivo: ninguém "detém" um módulo e qualquer programador pode mover/trabalhar em qualquer código

Integração Contínua: todas as mudanças são integradas na base de código pelo menos diariamente, em vez de "integração big-bang"

- Os testes têm de cobrir 100% antes e depois da integração
- Permite lançamentos frequentes

! Git Hub Flow!

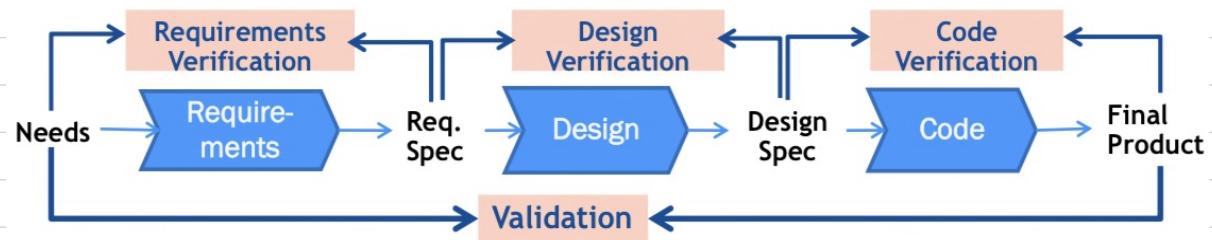
# GitHub Flow



## Verificações e Validação

Verificação: garantir (normalmente através de revisões) que os produtos de trabalho intermédio e o produto final estão "bem construídos", i. e., de acordo com a especificação — "Estamos a construir o produto bem?"

Validação: garantir (normalmente através de testes) que o produto final vai satisfazer o uso previsto no ambiente pretendido — "Estamos a construir o produto certo?"

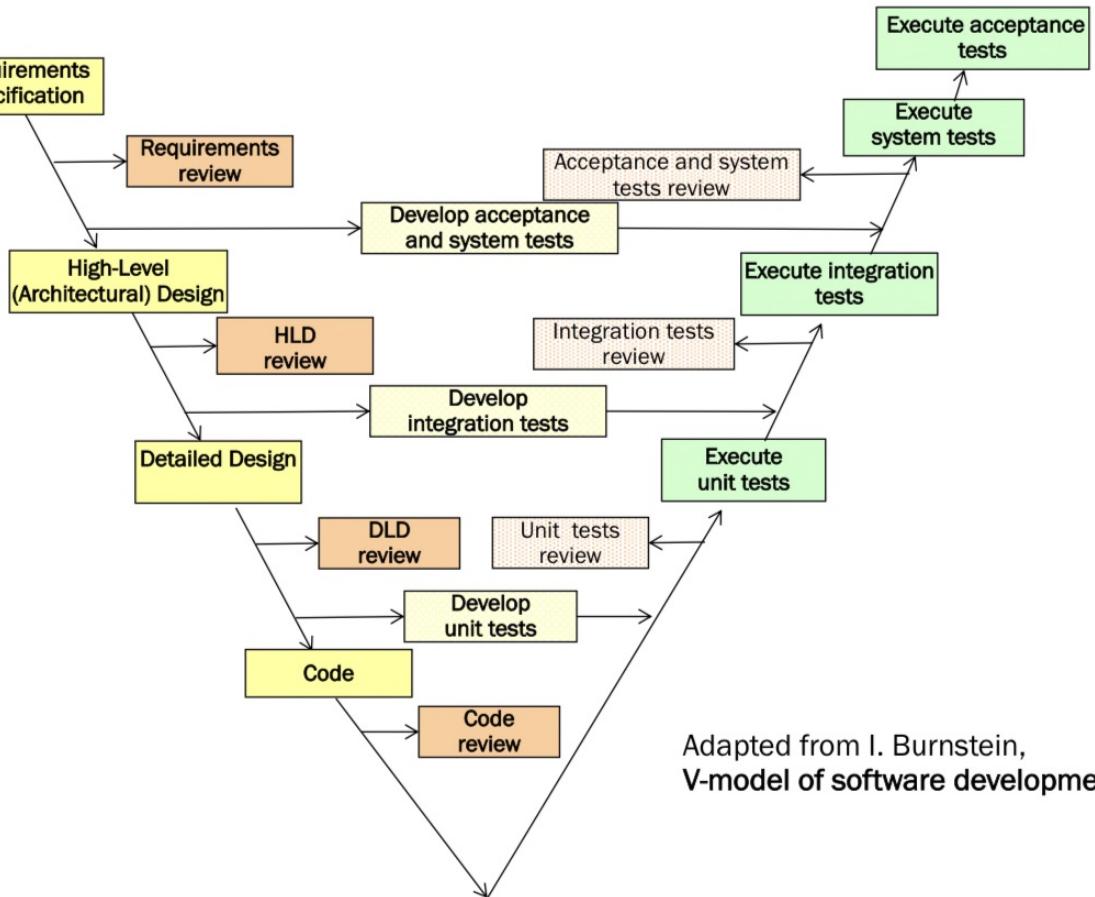


Técnicas Estáticas: envolvem analisar as representações estáticas do sistema para encontrar problemas e avaliar qualidade

Ex: revisões e inspeções; análise estática automatizada; verificação formal

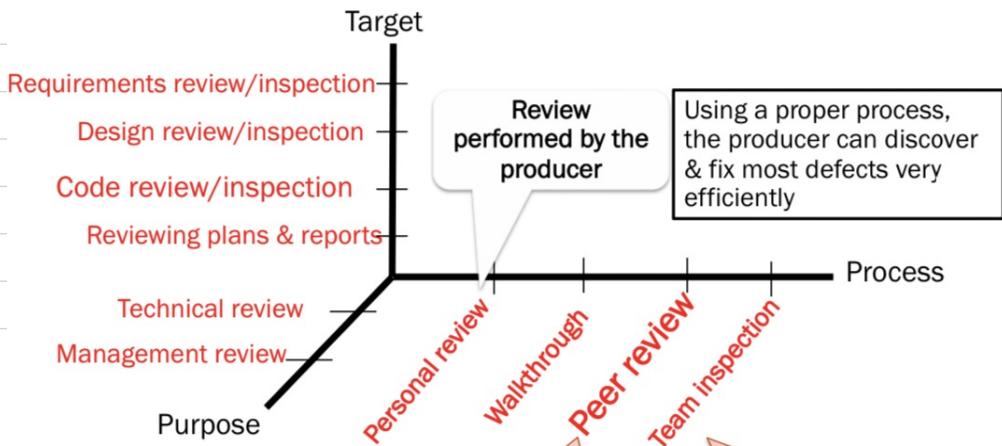
Técnicas Dinâmicas: envolvem executar o sistema (ou numa representação executável) e observar o seu comportamento

Ex: testagem de software; simulação



Adapted from I. Burnstein,  
V-model of software development

Inspeções e Revisões de Software: análise de representações estáticas do sistema para encontrar problemas



Category	What to verify
Naming conventions	<ul style="list-style-type: none"> <li>Classes, enums, typedefs, and extensions name should be in UpperCamelCase.</li> <li>Libraries, packages, directories, and source files name should be in snake_case</li> <li>Variables, constants, parameters, and named params should be in lowerCamelCase.</li> <li>Private variables names preceded with underscores</li> <li>...</li> </ul>
Folder and code organization	<ul style="list-style-type: none"> <li>Segregate code into a proper folder structure namely providers, models, screens, utils.</li> <li>Code is properly formatted with trailing commas used appropriately.</li> <li>Adequate comments added for documentation.</li> <li>...</li> </ul>
Widget structure and usage	<ul style="list-style-type: none"> <li>Use ListView builder when working with infinite lists or very large lists</li> <li>Use const in widgets which will not change when setState is called</li> <li>The build method should be pure, without any side effects.</li> <li>...</li> </ul>
Linting rules	<ul style="list-style-type: none"> <li>Avoid relative imports for files in lib/. Use package imports.</li> <li>Follow linting rules (See <a href="https://dart-lang.github.io/linter/lints/">https://dart-lang.github.io/linter/lints/</a>).</li> <li>...</li> </ul>
State management and separation of logic from UI	<ul style="list-style-type: none"> <li>Use provider as the recommended package for state management.</li> <li>You can also choose to use any other approach for state management like Bloc.</li> <li>Business logic should be separated from the UI</li> <li>...</li> </ul>
...	

## Testagem de Software

• Testagem de software consiste em executar os casos de teste definidos e observar o seu comportamento para descobrir defeitos

• "Testagem pode mostrar a presença de bugs, não a sua ausência"

Testagem de Defeitos: casos de teste unitários com alta capacidade de encontrar defeitos

Defeito / Falha / Bug: algo que tem de ser alterado num programa e pode levar a uma falha se não for corrigido

Testagem Estatística: uma informação de teste representativa e casos de teste para estimar uma métrica de qualidade de software

**Caso de Teste:** um conjunto de inputs de teste, condições de execução e resultados esperados desenvolvidos para um objetivo particular, tais como executar um caminho particular do programa ou para verificar cumprimento com um requisito específico & documentação de inputs específicos, resultados previstos e um conjunto de condições de execução para um item de teste

**Script de Teste:** definição concreta de passo de teste/procedimento

**Atividades de Teste:**

1. Planeamento do Teste
2. Monitorização e Controlo do Teste
3. Análise do Teste
4. Desenvolvimento do Teste
5. Implementação do Teste
6. Execução do Teste
7. Conclusão do Teste

Test level or phase

(regression)

acceptance

system

integration

unit

functional

security (penetration)

performance

usability

reliability

Other types:

- API / GUI testing
- developer / customer testing
- statistical / defect testing
- manual / automated testing

Test design strategy

white-box

black-box

Quality characteristics

## Níveis de Testes:

1. Testes Unitários: testagem de hardware individual ou unidades de software em grupos de unidades relacionadas
2. Testes de Integração: testagem em que componentes de software/hardware são combinados e testados para avaliar interação entre eles
3. Testes do Sistema: testagem conduzida num sistema completo integrado para avaliar a concordância do sistema com requisitos específicos
4. Testes de Aceitação: testagem formal conduzida para determinar se um sistema satisfaz ou não os seus critérios de aceitação e para permitir a um cliente, utilizador ou outra entidade autorizada para determinar se aceitar ou não o sistema
5. Testes de Regressão: retestagem relativa de um sistema ou componente para verificar que modificações não causaram efeitos indesejados e que o sistema ou componente continua a cumprir com os requisitos especificados

Testes Caixa-Preta: derivação de casos de teste baseados numa especificação externa

Partição em Classes Equivalentes: partição do domínio do input em classes de comportamento equivalente, separando classes de inputs válidos e inválidos e selecionar pelo menos um caso de teste de cada classe

Análise de Valores Limite: relação de valores de teste nas fronteiras de cada partição, para além de valores típicos

Testes Caixa-Branca: derivação de casos de teste de acordo com a estrutura do programa

# Modelação de Comportamentos UML

Como funciona um sistema?

1. Diagramas de Sequência
2. Diagramas de Máquinas de Estado
3. Diagramas de Atividade

Diagrama de Sequência: mostra a interação (mensagens trocadas ao longo do tempo) entre um conjunto de participantes num dado contexto

ATORES  
SISTEMAS

MECANISMO  
CENTRADO  
...

Tipos de Mensagens:

- synch Call: chamada de operação síncrona (o "caller" bloqueia)
- asynch Call: chamada de operação assíncrona (o caller não bloqueia)
- asynch Signal: envio e receção assíncronos de uma instância de um sinal
- ↔ reply: retorno de uma operação de chamada
- create Message: mensagem de criação de um objeto
- delete Message: mensagem que causa a destruição de um objeto

Fragmentos Combinados:

- opt: permite a operação de um operando (único), dependendo de uma condição
- alt: execução alternativa de (múltiplos) operandos, dependendo das respectivas condições
- loop: execução repetida de um operando (único), dependendo de uma expressão
- for: execução paralela dos operandos

Diagrama de Máquinas de Estado: úteis para modelar o ciclo de vida de objetos/sistemas com um comportamento discreto dirigido a eventos, mostrando:

- estados possíveis do objeto/sistema (finitos)
- transições entre estados (normalmente instantâneas)
- eventos que originam transições
- (-) ações tomadas pelo objeto/sistema em resposta a um evento
- (-) atividades realizadas pelo objeto/sistema enquanto num estado

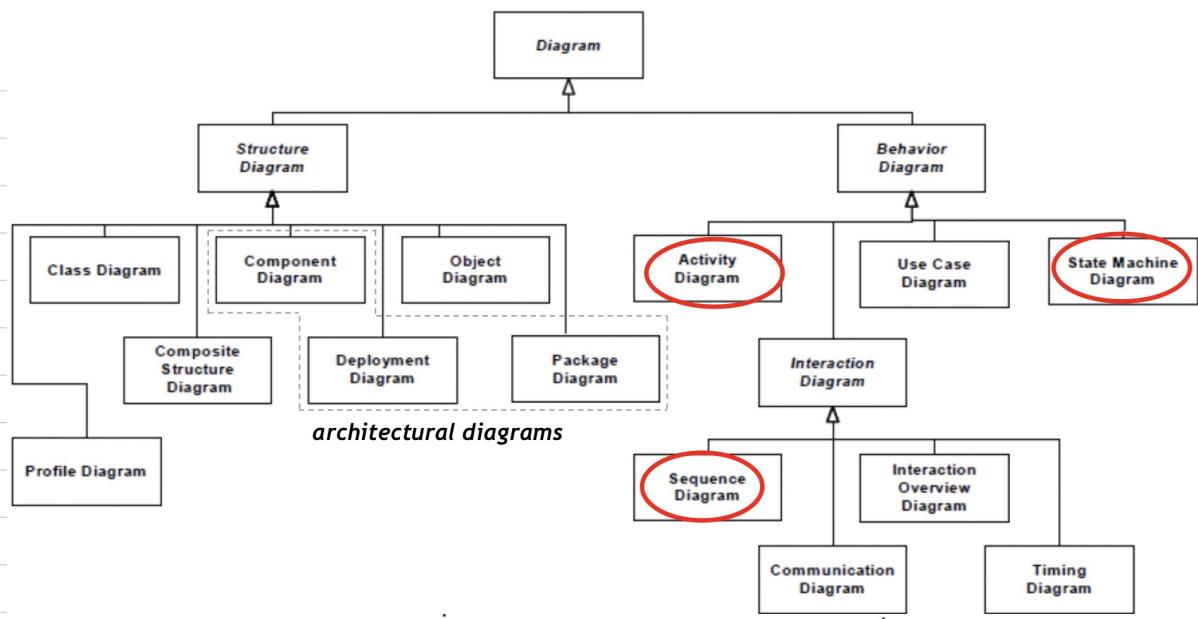
Extensões relativamente a automatos finitos:

1. Variáveis de Estado
2. Estados Compósitos
3. Regras Urtogonais / Concorrentes

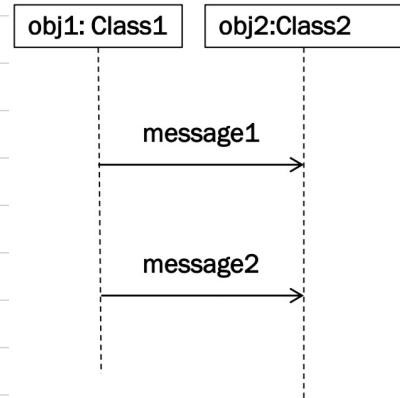
Tipos de Eventos:

1. chamada: operação de chamada, normalmente síncrona
2. envio: evento simbólico, modelado como um objeto que é enviado assincronamente por um objeto e recebido por outro objeto
3. Eventos Temporais:
  - a. after(t): ocorre  $t$  tempo depois de entrar no estado de origem
  - b. when(t): ocorre no instante de tempo  $t$
4. Evento de Alteração / Mudança:
  - a. when(...): ocorre quando a condição no estado interno do objeto/sistema se torna verdadeira

Diagrama de Atividade: mostra a decomposição hierárquica de uma atividade (um comportamento não atómico) num conjunto de unidades subordinadas (ações atómicas) e a sua coordenação usando um controlo múltiplo e modelo de fluxo de dados.

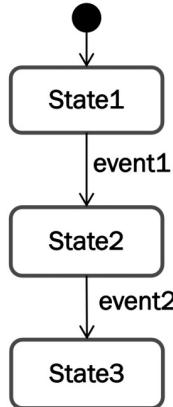


Sequence diagram



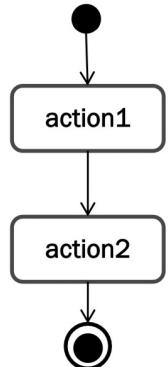
Emphasizes interactions (between objects, systems, actors, components, etc.)

State machine diagram



Emphasizes states & transitions of an object or system

Activity diagram



Emphasizes processing steps

# Trabalho de Equipa

1. O Espírito do Jogo
2. O Alvo
3. Comunidade de Confiança

YoLo Fértil: Empenho; Foco; Abertura; Respeito; Conexão

Ley de Conway: Comunicação & Feedback

Scrum Master:

1. observar e questionar
2. facilitar
3. ensinar
4. facilitar
5. "ativamente não fazer nada"

Planeamento do Sprint: desenho da solução para gran de confiança

Retrospectiva do Sprint: avaliar como correu o trabalho / iteração

Orgulho no Produto: identificação da equipa com o produto

Orgulho na Equipa: espírito de equipa e orgulho nela própria

- Definição de Engenharia de Software
- Áreas de Engenharia de Software
- História de Engenharia de Software
- Processos de Engenharia de Software (Atividades, (Des) Vantagens)
- Processos Ágeis VS. Plan-Driver
- Rational Unified Process
- Extreme Programming
- Engenharia de Requisitos (Atividades)
- Requisitos Funcionais VS. Não-Funcionais
- Características da Qualidade do Produto
- User Stories (INVEST)
- Diagramas UML
- Padrões de Arquitetura
- Testes: Verificação & Validação
- Técnicas Estáticas VS. Dinâmicas
- Tipos de Testes
- Atividades de Teste

## Engenharia de Software:

1. pesquisa, desenho, desenvolvimento e teste de software ao nível dos sistemas, compiladores e redes para diversas áreas
2. aplicação sistemática do conhecimento técnico e científico, métodos e experiência para projetar, implementar, testar e documentar software
3. aplicação de uma abordagem sistemática, disciplinada e quantificável ao desenvolvimento, operação e manutenção de software
4. disciplina de engenharia que se encarrega de todos os aspectos da produção de software
5. o estabelecer e usar de princípios sólidos de engenharia para obter economicamente software que é fiável e funciona eficientemente em máquinas reais
6. uma forma sistemática para projetar, construir e manter software mais rápido, mais barato e melhor

Areas: requisitos; design; construção; teste; manutenção; gestão; qualidade; ...

- 1945: primeiro bug — Grace Hopper
- 1956: primeiro processo de desenvolvimento de software documentado — SAGE
- 1960: Apollo — Margaret Hamilton
- 1968: "Go To Statement Considered Harmful" — Edgar Dijkstra
- 1968: primeira conferência de engenharia de software — NATO
- 1975: "The Mythical Man-Month" — Fred Brooks
- 1986: acidentes Therac-25 — "race condition"
- 1986: "No Silver Bullet" — Fred Brooks
- 1995: UML
- 1996: explosão Ariane 5 — "overflow"
- 2001: Manifesto Ágil

Processo de Software: conjunto estruturado de atividades necessárias para desenvolver um sistema de software ↔ eficiência, consistência e melhoria

Atividades {

- Especificação: definir o que o sistema deve fazer
- Design & Implementação: definir a organização do sistema e implementá-lo
- Validação: verificar que o sistema faz o que o cliente quer
- Evolução: mudar o sistema em resposta a alterações das necessidades

Ciclo:

1. Planeamento
2. Análise
3. Design
4. Implementação
5. Teste & Integração
6. Manutenção

"Plan-Driven": todas as atividades do processo são planejadas antecipadamente e o progresso é medido contra o plano

Ágil: o planejamento é incremental e é mais fácil mudar o processo para refletir alterações nos requisitos do cliente

Atividades {

- Corretiva: resolução de bugs
- Adaptativa: adaptação a novas plataformas e tecnologias
- Preditiva: novas funcionalidades

Modelo em Escada: plan-driven; fases distintas e separadas; requisitos fixos

Desenvolvimento Incremental: o sistema é desenvolvido em incrementos <sup>successivamente</sup> ~~avaliados~~

Integração & Configuração: o sistema é constituído a partir de componentes <sup>configuráveis existentes</sup>

Prototipagem de Software: objetivos → funcionalidade → desenvolver → avaliar

## National Unified Process:

- iterativo e incremental
- a base do UML
- dirigido para casos de utilização
- centrado na arquitetura

### Fases:

1. Iniciação: estabelecer o âmbito e condições limite/fronteira, visão operacional, critérios de aceitação, casos de uso críticos, cenários de operações primárias, arquitetura candidata, bem como estimar o custo, calendário e recursos e preparar um ambiente de apoio ao projeto
2. Elaboração: definir, validar e estabelecer a arquitetura, refinar a visão, criar planos de iteração detalhados para a fase de construção e pôr em prática o ambiente de desenvolvimento
3. Construção: gestão de recursos, controlo e otimização do processo; desenvolvimento completo dos componentes e teste contra os critérios de avaliação definidos e avaliação dos lançamentos contra os critérios de aceitação para a visão
4. Transição: executar planos de produção; finalizar material de apoio ao utilizador final; testar o produto entregável em local de desenvolvimento; iniciar o lançamento do produto; ajustar o produto com base no feedback; tornar o produto disponível para os utilizadores finais

Disciplinas: Modelação do Negócio; Requisitos; Análise e Design; Implementação; Teste; Produção; Configuração e Gestão de Alterações; Gestão do Projeto; Ambiente

**Extreme Programming:** "metodologia leve para equipes pequenas/médias desenvolverem software em face de requisitos vagos ou em rápidas mudanças"

O custo de um programa em mudança pode ser mantida constante ao longo do tempo — eliminar objetos code, redefinindo os custos

"Mudança a Mudança": iterar frequentemente, corrigir e reverenciar, programar e testar frequentemente, manter o cliente envolvido (obter feedback)

### Valores:

1. Comunicação: cliente no local, "user stories", código coletivo, reuniões diárias
2. Simplicidade: "Fazer a coisa mais simples que pode vir a funcionar"
3. Feedback: testes unitários, "user stories", lançamentos e "revisões"
4. Coragem: comunicar e aceitar feedback, testar código e refazê-lo

### Práticas:

1. Jogo de Planeamento: clientes decidem o esforço, longevidade, processos e tarefas
2. Lançamentos Pequenos: lançar cedo e frequentemente, poucas funcionalidades cada vez
3. Metáfora do Sistema: história que todos podem contar sobre como funciona o sistema
4. Design Simples: usar o design mais simples que cumpre o pretendido (YAGNI)
5. Desenvolvimento Dirigido a Testes: ter os primeiros, antes de adicionar a funcionalidade
6. Refactoring: melhorar a estrutura do código sem alterar o comportamento visível
7. Programação em Pares: o "driver" programa enquanto o "navegador" critica
8. Código Coletivo: ninguém tem um módulo individualmente / bagunça
9. Integração Contínua: todas as alterações são integradas pelo menos diariamente
10. Ritmo Sustentável: "fresco e ansioso de manhã, cansado e satisfeito à noite"
11. Cliente no Local: acesso contínuo a um cliente real que vai usar o sistema
12. Padrões de Código: convenções, indentação, organização, comentários, espaços, ...

**Engenharia de Requisitos:** o processo de estudar as necessidades do cliente e do utilizador para chegar à definição de requisitos do sistema/software/hardware

**Requisito de Software:** propriedade que deve ser exibida por software desenvolvido ou adaptado para resolver um problema particular

**Requisitos Funcionais:** descrevem as funções que o software deve executar (capacidades)

**Requisitos Não Funcionais:** atuam para restringir/limitar a solução

- Qualidade do Produto**
- |                  |                             |
|------------------|-----------------------------|
| 1. Portabilidade | 5. Adequação Funcional      |
| 2. Manutenção    | 6. Eficiência de Desempenho |
| 3. Segurança     | 7. Compatibilidade          |
| 4. Fidabilidade  | 8. Usabilidade              |

• Stakeholders são as principais fontes de requisitos: pessoas que não são afetadas pelo sistema e que têm influência direta ou indireta na elaboração de requisitos

1. Elicitação: interação com stakeholders e outras fontes para descobrir as suas necessidades e requisitos - entrevistas, brainstroming, questionários
2. Análise: organização e avaliação da informação recolhida para chegar a uma lista prioritária de requisitos concordados
3. Especificação: produção da documentação dos requisitos com nível de detalhe apropriado, dependente do contexto
4. Validação: averiguar/garantir que os requisitos documentados permitem alcançar os objetivos reais do projeto

**Artifício:** lista de requisitos, modelos do sistema, protótipos da interface do utilizador (mockups), testes de aceitação

User Story: "Promessa para uma conversa" — forma livre de registrar uma necessidade de software, com informação suficiente (para priorizar, planejar e iniciar conversas) — Quem? O que? Porque?

Independent Negotiable Valuable Estimable Small Testable

Teste de Aceitação: caso de teste definido para o cliente decidir se um sistema ou implementação de funcionalidade pode ser aceito, isto é, satisfaz requisitos e expectativas

Desenvolvimento Dirigido pelo Comportamento: cenários de teste que especificam comportamentos esperados do sistema

Given: contexto inicial ou pré-condição

When: evento ocorre

Then: arreganar resultados em pós-condições

Protótipo: versão inicial/primitiva de um sistema, mais barato, mais fácil e mais rápido de desenvolver, mas com funcionalidade limitada

- antevizão de como o sistema final vai parecer e funcionar
- abrange áreas de incerteza e riscos de malentendidos
- valida requisitos previamente identificados e identifica novos

Protótipo "Throw-away": foco nos requisitos em vez de limitações da implementação

Protótipo Evolutivo: para desenvolvimento rápido e iterativo de aplicações com forte envolvimento do utilizador final

Modelo do Sistema: representação simplificada de um sistema a partir de uma certa perspectiva

Modelo de Casos de Utilização: para classificar limites / fronteiras e âmbito do sistema e organizar requisitos funcionais (visão da utilização)

Modelo de Domínio: para organizar o vocabulário e capturar requisitos de informação no domínio do problema — diagrama de classes

Gestão de Projetos de Software: integração, âmbito, tempo, custo, qualidade, RH, comunicação, risco, procedura — PLAN - DO - CHECK - ACT

Variáveis { Recurso: menos eficaz para ajustar  
Tempo: mais dolorosa para ajustar  
Âmbito: mais eficaz (e importante!) para ajustar } QUALIDADE

Iterativo: execução repetida de ciclos de processo embriocados / aninhados

Incremental: o sistema é construído em estágios progressivos

Focusado: agrupa atividades de tipos semelhantes

Paralelo / Concomente: atividades de todos os tipos ocorrem ao mesmo tempo

Preditivo: criação de planos compreensivos baseados em atividades

Ágil: criação de um conjunto priorizado de entregáveis (execução optima)

Hélico: enfase nos individuos — dependência do desempenho individual e riscos

Colaborativo: enfase na equipa — auto-organização, diversidade e mitigação

Estratégia de Comando e Controlo → Estratégia de Empoderamento e Facilitação

**Arquitetura de Software:** "para arquicular e projetar software intemporal em grande escala" — controlar a complexidade (tecnológica e escala), garantir a integridade e qualidade do sistema, melhorar a reutilizabilidade do desenvolvimento, estabelecer "trade-offs" e facilitar a colaboração — é a organização fundamental de um sistema, incorporada nos seus componentes, as suas relações entre eles e o ambiente, bem como os princípios que governam o seu design e evolução

**Model-View-Controller:** separa a apresentação (V) e interação (C) do estado e dos dados (M) da aplicação, os estruturam o sistema em três partes lógicas que interagem entre elas — processamento interativo

**Paios e Filhos (Fluxo de Dados):** organiza o sistema como um conjunto de componentes de processamento de dados (filhos), ligados de modo que a informação flui entre componentes para processamento — "batch"

**Camadas:** organiza o sistema num conjunto de camadas em que cada agrega funcionalidades relacionadas e fornece serviços à camada acima COMPLEXIDADE DE ABSTRAÇÃO

**Vertical:** cada camada só pode interagir com a camada imediatamente inferior

**Horizontal:** cada camada pode interagir com qualquer camada inferior

**Repositório:** toda a informação no sistema é gerida num repositório central que é acessível a todos os componentes do sistema ou subsistemas; componentes ou subsistemas não interagem diretamente, só através do repositório

**Passivo:** aceita passivamente pedidos dos componentes

**Ativo:** alterações ao repositório acionam a execução de componentes

**Vista Lógica:** mostra as abstrações - chaves do sistema como objetos, classes ou pacotes — diagrama de pacotes UML — mostra pacotes lógicos e as suas relações



**Pacote:** mecanismo de agrupamento

**Vista de Implementação:** mostra como é que o software é decomposto (em componentes) para o desenvolvimento — diagrama de componentes UML — mostra os componentes de software e as dependências entre eles



**Componente:** representa uma parte modular de um sistema que encapsula os seus conteúdos e cuja manifestação é substituível no seu ambiente — o seu comportamento é definido em termos de interfaces (fornecidas/requeridas)

**Vista de Produção:** mostra o hardware do sistema e como é que os componentes de software são distribuídos por nós de hardware — diagrama de produção UML — mostra nós de hardware, relações de comunicação e artefactos de software produzidos neles



**Nó:** recurso de hardware onde artefactos podem ser produzidos

**Vista de Processo:** mostra como é que, em tempo de execução, o sistema é composto por processos que interagem — diagrama de atividade UML — mostra passos de processamento, arranqueamento de dados/objetos, fluxo de dados/objetos e oportunidades para paralelismo

**Vista de Caso de Utilização:** relaciona as outras vistas

**SCRUM:** confiança; foco; transparência; coragem; respeito; empenho

**Equipa:** 5-9 pessoas; multi-funcional; a tempo inteiro; auto-organização

**Scrum Master:** representa a gestão para o projeto; é responsável por incentivar os valores e práticas Scrum; remove impedimentos; garante que a equipa está completamente funcional e produtiva; promove cooperação próxima entre todos os papéis e funções; protege a equipa de interferências exteriores

**Product Owner:** define as funcionalidades do produto; decide a data de lançamento e o conteúdo; é responsável pelo lucro do produto; prioriza funcionalidades de acordo com o valor de mercado; ajusta funcionalidades e prioridades a cada iteração, conforme necessário; aceita ou rejeita resultados de trabalho

**Sprint Planning:** priorização e planeamento do Sprint

**Sprint Review:** a equipa apresenta o que consegui realizar no Sprint

- Organizar o trabalho em ciclos pequenos
- A gestão não interrompe a equipa durante um ciclo de trabalho
- A equipa reporta ao cliente, não à gestão
- A equipa estima quanto tempo o trabalho vai demorar
- A equipa decide quanto trabalho consegue fazer numa iteração
- A equipa decide como fazer o trabalho na iteração
- A equipa mede o seu próprio desempenho
- Os objetivos de trabalho não são definidos antes do início de cada ciclo
- Os objetivos de trabalho são definidos através de user stories

Verificação: estamos a construir o produto bem? garantir que produtos de trabalho intermédio e o produto final estão bem construídos, isto é, de acordo com as especificações

Validação: estamos a construir o produto certo? garantir que o produto final vai cumprir o seu propósito no ambiente pretendido

Técnicas Estáticas: analisam representações estáticas do sistema para encontrar problemas e avaliar qualidade - review e inspeções

Técnicas Dinâmicas: executar o sistema e observar o comportamento - testes

Testagem de Software: executar o software com casos de teste definidos e observar o seu comportamento para descobrir defeitos

Teste de Defeitos: usar casos de teste com alta probabilidade de encontrar <sup>defeitos</sup>

Defeito / Erro / Bug: algo que tem de ser removido num programa e pode levar a falhas se não for corrigido

Caso de Teste: conjunto de inputs de teste, condições de execução e resultados esperados desenhados para um objetivo particular, tal como executar um caminho particular do programa ou verificar concordância com um requisito específico **OU** documentação que especifica inputs, resultados permitidos e um conjunto de condições de execução para um item de teste

Script de Teste: definição concreta de etapas/procedimento de teste

1. Planejamento
2. Monitorização e Controlo
3. Análise
4. Design

5. Implementação
6. Execução
7. Completar

**Teste Unitário:** teste de unidades individuais de software/hardware em grupos de unidades relacionadas

**Teste de Integração:** teste em que componentes de hardware/software são combinados e testados para avaliar a interação entre eles

**Teste do Sistema:** teste conduzido num sistema completo e integrado para avaliar a concordância do sistema com requisitos específicos

**Teste de Aceitação:** teste formal conduzido para determinar se um sistema satisfaz ou não os seus critérios de aceitação e para permitir a um cliente ou utilizador determinar se aceita ou não o sistema

**Teste de Regressão:** verificação seletiva de um sistema ou componente para verificar que modificações não causaram efeitos indesejados e que o sistema ou componente ainda cumpre com os requisitos especificados

**Black-Box:** casos de teste baseados na especificação externa - ECP, BVA

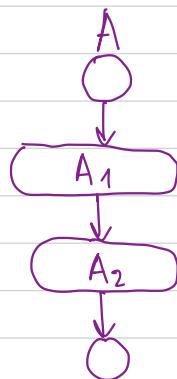
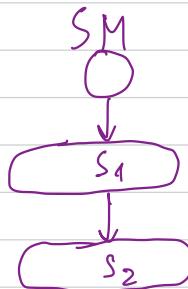
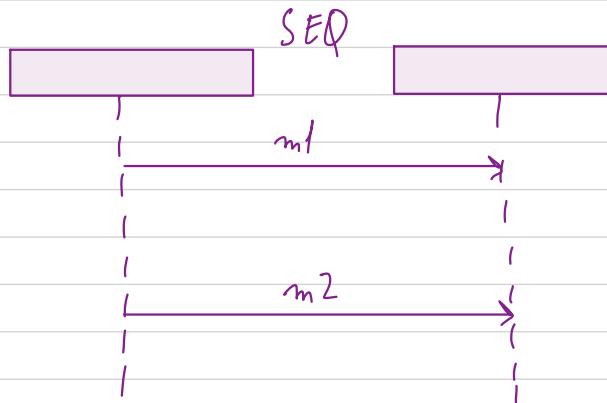
**White-Box:** casos de teste baseados na estrutura do programa

- Statement Coverage
- Decision/Branch Coverage
- ...

Diagrama de Seqüência: interações

Diagrama de Máquina de Estados: estados e transições

Diagrama de Atividade: passos/etapas de processamento





--

## L.EIC - ES EXAM

### June 20th, 2022

Duration : 60 min + 5 min for copying your answer options to the answer sheet.  
No consultation allowed. You can select only one answer option per question. You earn 0.5 points for a correct answer and lose 0.166 points for an incorrect one.  
Questions with multiple answers will get 0 points. Return all the sheets in the end.

**Question 1** What is the programming language used in Flutter?

- A Gherkin
- B Kotlin
- C Java
- D Dart

**Question 2** Which of the following is NOT a purpose of a domain model:

- A describe the dynamic system behavior
- B organize the vocabulary of the problem domain
- C identify relevant entities, attributes and relationships
- D capture information requirements

**Question 3** When should the waterfall model be applicable?

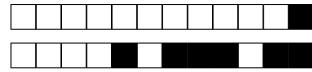
- A In projects with frequent requirements changes
- B Never
- C In large, multi-site, projects, with stable requirements
- D In the development of ultra large management applications

**Question 4** What is the typical structure of a user story?

- A Given ... when ... then ...
- B As a ... I want ... so that ...
- C For ... who ... the ... is a ... that ...
- D Preconditions ... flow of events ... postconditions ....

**Question 5** User interface prototyping is useful to (select the WRONG answer)

- A Validate previously identified requirements
- B Validate internal system architecture
- C Discover new requirements
- D Address areas of uncertainty



**Question 6** According to the IEEE Standard Glossary of Software Engineering Terminology, a software engineering approach should be (select the WRONG answer):

- A quantifiable
- B disciplined
- C complex
- D systematic

**Question 7** Which of the following is NOT a responsibility of the Scrum Master?

- A Shield the team from external interferences
- B Enact Scrum values and practices
- C Ensure that the team is fully functional and productive
- D Represent the Product Owner to the project

**Question 8** Which one is NOT a typical column in a project board (kanban board) on GitHub?

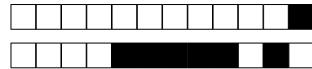
- A Done
- B Deferred
- C To do (or Product backlog, Iteration backlog, etc.)
- D In progress

**Question 9** What statement is FALSE about the structure of apps in Flutter?

- A The main function calls another function, from the Flutter library, passing as argument a widget of any type defining the root of the tree of UI objects
- B Almost all widgets are `StatelessWidget` or `StatefulWidget`
- C Anything related to UI (displayed objects, formatting, interaction gestures, ...) is a widget
- D The application is structured as a tree of UI objects, that derive from the `Widget` class

**Question 10** Which UML diagram shows messages exchanged between a set of participants over time?

- A Activity diagram
- B Use case diagram
- C State machine diagram
- D Sequence diagram



**Question 11** “Find actors and use cases” is an activity of which RUP discipline?

- A Analysis and Design
- B Implementation
- C Test
- D Requirements

**Question 12** Which one is NOT a desired “INVEST” property of a user story?

- A Traceable
- B Independent
- C Negotiable
- D Valuable

**Question 13** “An ordered list of items (epics, stories, work items, etc.) needed to deliver the whole product” is called:

- A Product backlog
- B Sprint backlog
- C Iteration backlog
- D Backlog

**Question 14** UML package diagrams are best suited to describe which view?

- A Deployment View
- B Process View
- C Implementation View
- D Logical View

**Question 15** Which one is a UML diagram suited to show processing steps, using a mixed control and data flow model?

- A Process diagram
- B Data flow diagram
- C Activity diagram
- D Workflow diagram

**Question 16** Which of the following is a non-functional requirement of a system?

- A The system should allow the custom to purchase tickets online.
- B The system should allow the custom to purchase tickets at the counter.
- C The system should allow the custom to purchase multiple tickets at once
- D The system should be developed in a functional programming language.



**Question 17** Which one is NOT a phase in the Rational Unified Process (RUP)?

- A Elaboration
- B Construction
- C Delivery
- D Inception

**Question 18** What is the normal sequence of Github Flow?

- A pull request > branch > commits > review > deploy > merge
- B branch > commits > pull request > review > deploy > merge
- C branch > pull request > commits > review > deploy > merge
- D branch > commits > review > deploy > pull request > merge

**Question 19** “Studying stakeholder needs to arrive at a definition of system, hardware or software requirements” is called:

- A software engineering
- B requirements study
- C requirements engineering
- D requirements specification

**Question 20** Which one is NOT part of the “Definition of Done” (DoD) of user stories and work items?

- A Accepted
- B Tested
- C Estimated
- D Implemented

**Question 21** Which of the following IS NOT a XP practice?

- A Continuous Integration
- B Refactoring
- C Test-driven Development
- D Individual Code Ownership

**Question 22** What is FALSE about how pair programming works?

- A Driver enters code, while navigator critiques it
- B Periodically switch roles and pairs
- C One of the programmers is designated to commit all the code
- D Two programmers work together at one machine



**Question 23** Which of the following is NOT a valid element or relationship in UML deployment diagrams?

- site
- node
- artifact
- association

**Question 24** Which package should be used for automating acceptance tests in Flutter?

- test
- flutter\_gherkin
- flutter\_test
- acceptance\_test

**Question 25** T-shirt sizes are used in agile project management to estimate

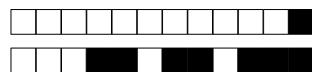
- effort of backlog items
- size of software items
- cost of human resources
- duration of backlog items

**Question 26** The MoSCoW method is used to assign \_\_\_\_\_ to product backlog items

- profitability estimates
- priorities
- effort estimates
- schedule estimates

**Question 27** Which of the following is a key practice of Scrum?

- The management is free to engage and interrupt the team during a work cycle
- The team measures its own performance
- Work goals can be adapted at any time
- The Scrum Master decides how to do the work in the iteration



**Question 28** What is the structure of a test scenario in Gherkin?

- A Preconditions - Flow of events - Postconditions
- B Setup - Body - Teardown
- C Given - When - Then
- D Setup - Assert - Teardown

**Question 29** “A subset of items taken from the product backlog to be implemented in an iteration” is called the:

- A Iteration goal
- B Iteration backlog
- C Iteration whish list
- D Iteration plan

**Question 30** Which UML diagram is used to model the lifecycle of objects or systems with a discrete event-driven behavior?

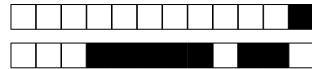
- A State machine diagram
- B Composite structure diagram
- C Activity diagram
- D Object diagram

**Question 31** In agile processes, if the planned work is not completed at the end of an iteration, what should be done?

- A Keep the work in the iteration backlog of this iteration
- B Extend the iteration until the work is completed
- C Assign the work remaining to developers outside the team
- D Move the work remaining to the product backlog

**Question 32** Which of the following IS NOT a software engineering knowledge area according to SWEBOK?

- A Software Design
- B Software Testing
- C Software Requirements
- D Software Business



**Question 33** Which of the following is NOT a test phase?

- A acceptance testing
- B integration testing
- C delivery testing
- D unit testing

**Question 34** Which of the following is NOT a view in the 4+1 view model of software architecture?

- A Implementation View
- B Logical View
- C Dynamic View
- D Deployment View

**Question 35** The purpose of a use case model is to show:

- A internal system structure
- B system workflow
- C the main concepts and their relationships in a domain
- D system functionalities or services as perceived by users

**Question 36** Regarding the advantages of iterative and incremental development, which of the following statements is FALSE?

- A Reduces the need for code refactoring
- B More frequent & early customer feedback reduces failure risk
- C System functionality is available earlier
- D The cost of accommodating changing requirements is reduced

**Question 37** In the use case “Purchase tickets online”, “The customer gets the electronic tickets with a QR code” is a:

- A postcondition
- B invariant
- C precondition
- D constraint



**Question 38** Which statement is FALSE about pull requests in GitHub?

- A Pull requests let you tell others about changes you have pushed to a branch in a repository
- B For the changes proposed in the pull request to be merged, the pull request must be approved
- C A pull request must be associated with a work item in the project board
- D Once you have created a pull request other contributors can review your proposed changes and add review comments

**Question 39** The role of software architecture includes which of the following aims?

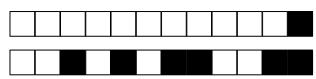
- A To manage different development teams
- B To ensure required quality attributes, such as scalability, interoperability, modularity, etc.
- C To reduce system integrity
- D To avoid decisions that influence system evolution

**Question 40** Which one is NOT a control variable in the iron triangle of project management?

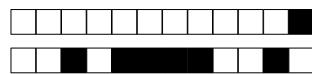
- A Resources (cost, budget)
- B Teams (people)
- C Scope (features, functionality)
- D Schedule (time)



+1/9/52+



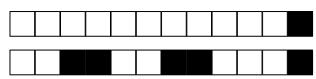
+1/10/51+

**Answer sheet:**

--

*Answers for grading must be given exclusively on this sheet: answers given on other sheets will be ignored. Fill with a pen the answer box.*

QUESTION 1 :  A  B  C  DQUESTION 2 :  A  B  C  DQUESTION 3 :  A  B  C  DQUESTION 4 :  A  B  C  DQUESTION 5 :  A  B  C  DQUESTION 6 :  A  B  C  DQUESTION 7 :  A  B  C  DQUESTION 8 :  A  B  C  DQUESTION 9 :  A  B  C  DQUESTION 10 :  A  B  C  DQUESTION 11 :  A  B  C  DQUESTION 12 :  A  B  C  DQUESTION 13 :  A  B  C  DQUESTION 14 :  A  B  C  DQUESTION 15 :  A  B  C  DQUESTION 16 :  A  B  C  DQUESTION 17 :  A  B  C  DQUESTION 18 :  A  B  C  DQUESTION 19 :  A  B  C  DQUESTION 20 :  A  B  C  DQUESTION 21 :  A  B  C  DQUESTION 22 :  A  B  C  DQUESTION 23 :  A  B  C  DQUESTION 24 :  A  B  C  DQUESTION 25 :  A  B  C  DQUESTION 26 :  A  B  C  DQUESTION 27 :  A  B  C  DQUESTION 28 :  A  B  C  DQUESTION 29 :  A  B  C  DQUESTION 30 :  A  B  C  DQUESTION 31 :  A  B  C  DQUESTION 32 :  A  B  C  DQUESTION 33 :  A  B  C  DQUESTION 34 :  A  B  C  DQUESTION 35 :  A  B  C  DQUESTION 36 :  A  B  C  DQUESTION 37 :  A  B  C  DQUESTION 38 :  A  B  C  DQUESTION 39 :  A  B  C  DQUESTION 40 :  A  B  C  D



+1/12/49+



## L.EIC - ES EXAM

### June 20th, 2022

Duration : 60 min + 5 min for copying your answer options to the answer sheet.  
No consultation allowed. You can select only one answer option per question. You earn 0.5 points for a correct answer and lose 0.166 points for an incorrect one.  
Questions with multiple answers will get 0 points. Return all the sheets in the end.

**Question 1** Which of the following IS NOT a software engineering knowledge area according to SWEBOK?

- A Software Testing
- B Software Requirements
- C Software Design
- D Software Business

**Question 2** Regarding the advantages of iterative and incremental development, which of the following statements is FALSE?

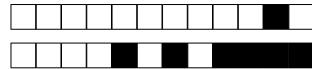
- A The cost of accommodating changing requirements is reduced
- B System functionality is available earlier
- C More frequent & early customer feedback reduces failure risk
- D Reduces the need for code refactoring

**Question 3** What is FALSE about how pair programming works?

- A Two programmers work together at one machine
- B One of the programmers is designated to commit all the code
- C Periodically switch roles and pairs
- D Driver enters code, while navigator critiques it

**Question 4** Which of the following is a non-functional requirement of a system?

- A The system should allow the custom to purchase tickets online.
- B The system should be developed in a functional programming language.
- C The system should allow the custom to purchase multiple tickets at once
- D The system should allow the custom to purchase tickets at the counter.



**Question 5** Which one is NOT a desired “INVEST” property of a user story?

- A Valuable
- B Traceable
- C Negotiable
- D Independent

**Question 6** According to the IEEE Standard Glossary of Software Engineering Terminology, a software engineering approach should be (select the WRONG answer):

- A systematic
- B quantifiable
- C complex
- D disciplined

**Question 7** The MoSCoW method is used to assign \_\_\_\_\_ to product backlog items

- A profitability estimates
- B effort estimates
- C priorities
- D schedule estimates

**Question 8** “An ordered list of items (epics, stories, work items, etc.) needed to deliver the whole product” is called:

- A Sprint backlog
- B Backlog
- C Product backlog
- D Iteration backlog

**Question 9** Which one is NOT a control variable in the iron triangle of project management?

- A Teams (people)
- B Resources (cost, budget)
- C Scope (features, functionality)
- D Schedule (time)



**Question 10** Which one is NOT part of the “Definition of Done” (DoD) of user stories and work items?

- A Tested
- B Estimated
- C Implemented
- D Accepted

**Question 11** User interface prototyping is useful to (select the WRONG answer)

- A Validate previously identified requirements
- B Discover new requirements
- C Address areas of uncertainty
- D Validate internal system architecture

**Question 12** Which of the following is NOT a responsibility of the Scrum Master?

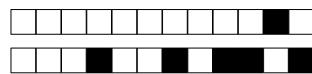
- A Ensure that the team is fully functional and productive
- B Enact Scrum values and practices
- C Shield the team from external interferences
- D Represent the Product Owner to the project

**Question 13** The purpose of a use case model is to show:

- A system workflow
- B the main concepts and their relationships in a domain
- C internal system structure
- D system functionalities or services as perceived by users

**Question 14** Which of the following is NOT a valid element or relationship in UML deployment diagrams?

- A site
- B association
- C node
- D artifact



**Question 15** The role of software architecture includes which of the following aims?

- A To reduce system integrity
- B To ensure required quality attributes, such as scalability, interoperability, modularity, etc.
- C To avoid decisions that influence system evolution
- D To manage different development teams

**Question 16** What is the structure of a test scenario in Gherkin?

- A Given - When - Then
- B Preconditions - Flow of events - Postconditions
- C Setup - Body - Teardown
- D Setup - Assert - Teardown

**Question 17** Which UML diagram shows messages exchanged between a set of participants over time?

- A State machine diagram
- B Activity diagram
- C Sequence diagram
- D Use case diagram

**Question 18** Which of the following is a key practice of Scrum?

- A The Scrum Master decides how to do the work in the iteration
- B The management is free to engage and interrupt the team during a work cycle
- C The team measures its own performance
- D Work goals can be adapted at any time

**Question 19** Which UML diagram is used to model the lifecycle of objects or systems with a discrete event-driven behavior?

- A State machine diagram
- B Composite structure diagram
- C Object diagram
- D Activity diagram



**Question 20** Which one is a UML diagram suited to show processing steps, using a mixed control and data flow model?

- A Data flow diagram
- B Activity diagram
- C Workflow diagram
- D Process diagram

**Question 21** In the use case “Purchase tickets online”, “The customer gets the electronic tickets with a QR code” is a:

- A constraint
- B postcondition
- C invariant
- D precondition

**Question 22** “Find actors and use cases” is an activity of which RUP discipline?

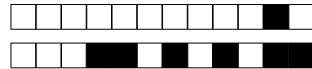
- A Test
- B Implementation
- C Requirements
- D Analysis and Design

**Question 23** Which one is NOT a phase in the Rational Unified Process (RUP)?

- A Delivery
- B Elaboration
- C Construction
- D Inception

**Question 24** “A subset of items taken from the product backlog to be implemented in an iteration” is called the:

- A Iteration whish list
- B Iteration goal
- C Iteration plan
- D Iteration backlog



**Question 25** What statement is FALSE about the structure of apps in Flutter?

- [A] The application is structured as a tree of UI objects, that derive from the `Widget` class
- [B] The main function calls another function, from the Flutter library, passing as argument a widget of any type defining the root of the tree of UI objects
- [C] Almost all widgets are `StatelessWidget` or `StatefulWidget`
- [D] Anything related to UI (displayed objects, formatting, interaction gestures, ...) is a widget

**Question 26** T-shirt sizes are used in agile project management to estimate

- [A] size of software items
- [B] cost of human resources
- [C] duration of backlog items
- [D] effort of backlog items

**Question 27** Which package should be used for automating acceptance tests in Flutter?

- [A] `acceptance_test`
- [B] `test`
- [C] `flutter_test`
- [D] `flutter_gherkin`

**Question 28** What is the typical structure of a user story?

- [A] As a ... I want ... so that ...
- [B] Given ... when ... then ...
- [C] Preconditions ... flow of events ... postconditions ....
- [D] For ... who ... the ... is a ... that ...

**Question 29** In agile processes, if the planned work is not completed at the end of an iteration, what should be done?

- [A] Move the work remaining to the product backlog
- [B] Assign the work remaining to developers outside the team
- [C] Extend the iteration until the work is completed
- [D] Keep the work in the iteration backlog of this iteration



**Question 30** Which of the following is NOT a view in the 4+1 view model of software architecture?

- A Dynamic View
- B Implementation View
- C Logical View
- D Deployment View

**Question 31** What is the programming language used in Flutter?

- A Java
- B Kotlin
- C Gherkin
- D Dart

**Question 32** Which of the following is NOT a test phase?

- A integration testing
- B unit testing
- C acceptance testing
- D delivery testing

**Question 33** Which one is NOT a typical column in a project board (kanban board) on GitHub?

- A To do (or Product backlog, Iteration backlog, etc.)
- B Deferred
- C In progress
- D Done

**Question 34** UML package diagrams are best suited to describe which view?

- A Implementation View
- B Logical View
- C Deployment View
- D Process View

**Question 35** When should the waterfall model be applicable?

- A In the development of ultra large management applications
- B In large, multi-site, projects, with stable requirements
- C In projects with frequent requirements changes
- D Never



**Question 36** “Studying stakeholder needs to arrive at a definition of system, hardware or software requirements” is called:

- A requirements engineering
- B requirements study
- C software engineering
- D requirements specification

**Question 37** Which of the following IS NOT a XP practice?

- A Continuous Integration
- B Refactoring
- C Individual Code Ownership
- D Test-driven Development

**Question 38** Which statement is FALSE about pull requests in GitHub?

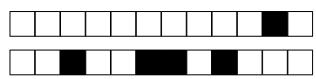
- A Pull requests let you tell others about changes you have pushed to a branch in a repository
- B A pull request must be associated with a work item in the project board
- C For the changes proposed in the pull request to be merged, the pull request must be approved
- D Once you have created a pull request other contributors can review your proposed changes and add review comments

**Question 39** Which of the following is NOT a purpose of a domain model:

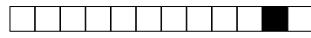
- A identify relevant entities, attributes and relationships
- B organize the vocabulary of the problem domain
- C capture information requirements
- D describe the dynamic system behavior

**Question 40** What is the normal sequence of Github Flow?

- A branch > commits > pull request > review > deploy > merge
- B pull request > branch > commits > review > deploy > merge
- C branch > pull request > commits > review > deploy > merge
- D branch > commits > review > deploy > pull request > merge



+2/9/40+



+2/10/39+

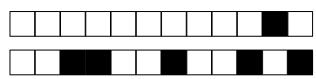


**Answer sheet:**

--

*Answers for grading must be given exclusively on this sheet: answers given on other sheets will be ignored. Fill with a pen the answer box.*

QUESTION 1 :  A  B  C  DQUESTION 2 :  A  B  C  DQUESTION 3 :  A  B  C  DQUESTION 4 :  A  B  C  DQUESTION 5 :  A  B  C  DQUESTION 6 :  A  B  C  DQUESTION 7 :  A  B  C  DQUESTION 8 :  A  B  C  DQUESTION 9 :  A  B  C  DQUESTION 10 :  A  B  C  DQUESTION 11 :  A  B  C  DQUESTION 12 :  A  B  C  DQUESTION 13 :  A  B  C  DQUESTION 14 :  A  B  C  DQUESTION 15 :  A  B  C  DQUESTION 16 :  A  B  C  DQUESTION 17 :  A  B  C  DQUESTION 18 :  A  B  C  DQUESTION 19 :  A  B  C  DQUESTION 20 :  A  B  C  DQUESTION 21 :  A  B  C  DQUESTION 22 :  A  B  C  DQUESTION 23 :  A  B  C  DQUESTION 24 :  A  B  C  DQUESTION 25 :  A  B  C  DQUESTION 26 :  A  B  C  DQUESTION 27 :  A  B  C  DQUESTION 28 :  A  B  C  DQUESTION 29 :  A  B  C  DQUESTION 30 :  A  B  C  DQUESTION 31 :  A  B  C  DQUESTION 32 :  A  B  C  DQUESTION 33 :  A  B  C  DQUESTION 34 :  A  B  C  DQUESTION 35 :  A  B  C  DQUESTION 36 :  A  B  C  DQUESTION 37 :  A  B  C  DQUESTION 38 :  A  B  C  DQUESTION 39 :  A  B  C  DQUESTION 40 :  A  B  C  D



+2/12/37+



--

## L.EIC - ES EXAM

### June 20th, 2022

Duration : 60 min + 5 min for copying your answer options to the answer sheet.  
No consultation allowed. You can select only one answer option per question. You earn 0.5 points for a correct answer and lose 0.166 points for an incorrect one.  
Questions with multiple answers will get 0 points. Return all the sheets in the end.

**Question 1** Which of the following is a non-functional requirement of a system?

- A The system should allow the custom to purchase multiple tickets at once
- B The system should allow the custom to purchase tickets online.
- C The system should allow the custom to purchase tickets at the counter.
- D The system should be developed in a functional programming language.

**Question 2** Which one is NOT part of the “Definition of Done” (DoD) of user stories and work items?

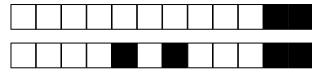
- A Accepted
- B Estimated
- C Implemented
- D Tested

**Question 3** “An ordered list of items (epics, stories, work items, etc.) needed to deliver the whole product” is called:

- A Product backlog
- B Backlog
- C Sprint backlog
- D Iteration backlog

**Question 4** Which of the following is a key practice of Scrum?

- A The Scrum Master decides how to do the work in the iteration
- B Work goals can be adapted at any time
- C The management is free to engage and interrupt the team during a work cycle
- D The team measures its own performance



**Question 5** What is the structure of a test scenario in Gherkin?

- A Setup - Body - Teardown
- B Preconditions - Flow of events - Postconditions
- C Given - When - Then
- D Setup - Assert - Teardown

**Question 6** Which package should be used for automating acceptance tests in Flutter?

- A test
- B flutter\_gherkin
- C acceptance\_test
- D flutter\_test

**Question 7** Which of the following is NOT a valid element or relationship in UML deployment diagrams?

- A node
- B artifact
- C site
- D association

**Question 8** Which UML diagram shows messages exchanged between a set of participants over time?

- A Activity diagram
- B Use case diagram
- C State machine diagram
- D Sequence diagram

**Question 9** What is the normal sequence of Github Flow?

- A branch > commits > review > deploy > pull request > merge
- B branch > commits > pull request > review > deploy > merge
- C branch > pull request > commits > review > deploy > merge
- D pull request > branch > commits > review > deploy > merge



**Question 10** The MoSCoW method is used to assign \_\_\_\_\_ to product backlog items

- A priorities
- B profitability estimates
- C effort estimates
- D schedule estimates

**Question 11** T-shirt sizes are used in agile project management to estimate

- A size of software items
- B cost of human resources
- C duration of backlog items
- D effort of backlog items

**Question 12** “A subset of items taken from the product backlog to be implemented in an iteration” is called the:

- A Iteration goal
- B Iteration plan
- C Iteration backlog
- D Iteration wish list

**Question 13** “Studying stakeholder needs to arrive at a definition of system, hardware or software requirements” is called:

- A software engineering
- B requirements specification
- C requirements engineering
- D requirements study

**Question 14** What statement is FALSE about the structure of apps in Flutter?

- A The application is structured as a tree of UI objects, that derive from the `Widget` class
- B Anything related to UI (displayed objects, formatting, interaction gestures, ...) is a widget
- C Almost all widgets are `StatelessWidget` or `StatefulWidget`
- D The main function calls another function, from the Flutter library, passing as argument a widget of any type defining the root of the tree of UI objects



**Question 15** Which one is a UML diagram suited to show processing steps, using a mixed control and data flow model?

- A Workflow diagram
- B Process diagram
- C Data flow diagram
- D Activity diagram

**Question 16** What is FALSE about how pair programming works?

- A Two programmers work together at one machine
- B One of the programmers is designated to commit all the code
- C Periodically switch roles and pairs
- D Driver enters code, while navigator critiques it

**Question 17** According to the IEEE Standard Glossary of Software Engineering Terminology, a software engineering approach should be (select the WRONG answer):

- A systematic
- B quantifiable
- C complex
- D disciplined

**Question 18** Which of the following is NOT a view in the 4+1 view model of software architecture?

- A Logical View
- B Implementation View
- C Dynamic View
- D Deployment View

**Question 19** Which of the following is NOT a responsibility of the Scrum Master?

- A Enact Scrum values and practices
- B Shield the team from external interferences
- C Ensure that the team is fully functional and productive
- D Represent the Product Owner to the project



**Question 20** Which statement is FALSE about pull requests in GitHub?

- A Pull request must be associated with a work item in the project board
- B Once you have created a pull request other contributors can review your proposed changes and add review comments
- C Pull requests let you tell others about changes you have pushed to a branch in a repository
- D For the changes proposed in the pull request to be merged, the pull request must be approved

**Question 21** Which one is NOT a phase in the Rational Unified Process (RUP)?

- A Inception
- B Elaboration
- C Construction
- D Delivery

**Question 22** What is the programming language used in Flutter?

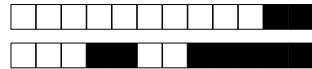
- A Dart
- B Kotlin
- C Gherkin
- D Java

**Question 23** Which UML diagram is used to model the lifecycle of objects or systems with a discrete event-driven behavior?

- A Object diagram
- B Activity diagram
- C Composite structure diagram
- D State machine diagram

**Question 24** Which of the following is NOT a purpose of a domain model:

- A identify relevant entities, attributes and relationships
- B capture information requirements
- C describe the dynamic system behavior
- D organize the vocabulary of the problem domain



**Question 25** “Find actors and use cases” is an activity of which RUP discipline?

- A Implementation
- B Analysis and Design
- C Requirements
- D Test

**Question 26** UML package diagrams are best suited to describe which view?

- A Deployment View
- B Process View
- C Logical View
- D Implementation View

**Question 27** Which of the following IS NOT a software engineering knowledge area according to SWEBOK?

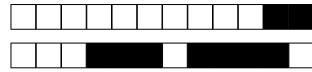
- A Software Design
- B Software Testing
- C Software Requirements
- D Software Business

**Question 28** In the use case “Purchase tickets online”, “The customer gets the electronic tickets with a QR code” is a:

- A invariant
- B precondition
- C postcondition
- D constraint

**Question 29** In agile processes, if the planned work is not completed at the end of an iteration, what should be done?

- A Extend the iteration until the work is completed
- B Move the work remaining to the product backlog
- C Assign the work remaining to developers outside the team
- D Keep the work in the iteration backlog of this iteration



**Question 30** Which one is NOT a desired “INVEST” property of a user story?

- A Independent
- B Negotiable
- C Valuable
- D Traceable

**Question 31** What is the typical structure of a user story?

- A Preconditions ... flow of events ... postconditions ....
- B For ... who ... the ... is a ... that ...
- C As a ... I want ... so that ...
- D Given ... when ... then ...

**Question 32** Which one is NOT a typical column in a project board (kanban board) on GitHub?

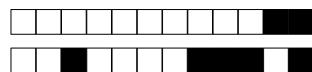
- A To do (or Product backlog, Iteration backlog, etc.)
- B Deferred
- C In progress
- D Done

**Question 33** Which one is NOT a control variable in the iron triangle of project management?

- A Scope (features, functionality)
- B Schedule (time)
- C Teams (people)
- D Resources (cost, budget)

**Question 34** Regarding the advantages of iterative and incremental development, which of the following statements is FALSE?

- A More frequent & early customer feedback reduces failure risk
- B Reduces the need for code refactoring
- C System functionality is available earlier
- D The cost of accommodating changing requirements is reduced



**Question 35** The purpose of a use case model is to show:

- A system functionalities or services as perceived by users
- B the main concepts and their relationships in a domain
- C internal system structure
- D system workflow

**Question 36** Which of the following is NOT a test phase?

- A unit testing
- B integration testing
- C acceptance testing
- D delivery testing

**Question 37** User interface prototyping is useful to (select the WRONG answer)

- A Validate previously identified requirements
- B Discover new requirements
- C Validate internal system architecture
- D Address areas of uncertainty

**Question 38** Which of the following IS NOT a XP practice?

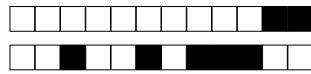
- A Continuous Integration
- B Test-driven Development
- C Individual Code Ownership
- D Refactoring

**Question 39** When should the waterfall model be applicable?

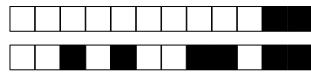
- A Never
- B In large, multi-site, projects, with stable requirements
- C In the development of ultra large management applications
- D In projects with frequent requirements changes

**Question 40** The role of software architecture includes which of the following aims?

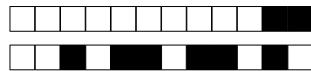
- A To ensure required quality attributes, such as scalability, interoperability, modularity, etc.
- B To manage different development teams
- C To reduce system integrity
- D To avoid decisions that influence system evolution



+3/9/28+



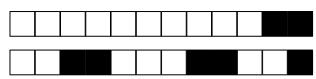
+3/10/27+

**Answer sheet:**

--

*Answers for grading must be given exclusively on this sheet: answers given on other sheets will be ignored. Fill with a pen the answer box.*

QUESTION 1 :  A  B  C  DQUESTION 2 :  A  B  C  DQUESTION 3 :  A  B  C  DQUESTION 4 :  A  B  C  DQUESTION 5 :  A  B  C  DQUESTION 6 :  A  B  C  DQUESTION 7 :  A  B  C  DQUESTION 8 :  A  B  C  DQUESTION 9 :  A  B  C  DQUESTION 10 :  A  B  C  DQUESTION 11 :  A  B  C  DQUESTION 12 :  A  B  C  DQUESTION 13 :  A  B  C  DQUESTION 14 :  A  B  C  DQUESTION 15 :  A  B  C  DQUESTION 16 :  A  B  C  DQUESTION 17 :  A  B  C  DQUESTION 18 :  A  B  C  DQUESTION 19 :  A  B  C  DQUESTION 20 :  A  B  C  DQUESTION 21 :  A  B  C  DQUESTION 22 :  A  B  C  DQUESTION 23 :  A  B  C  DQUESTION 24 :  A  B  C  DQUESTION 25 :  A  B  C  DQUESTION 26 :  A  B  C  DQUESTION 27 :  A  B  C  DQUESTION 28 :  A  B  C  DQUESTION 29 :  A  B  C  DQUESTION 30 :  A  B  C  DQUESTION 31 :  A  B  C  DQUESTION 32 :  A  B  C  DQUESTION 33 :  A  B  C  DQUESTION 34 :  A  B  C  DQUESTION 35 :  A  B  C  DQUESTION 36 :  A  B  C  DQUESTION 37 :  A  B  C  DQUESTION 38 :  A  B  C  DQUESTION 39 :  A  B  C  DQUESTION 40 :  A  B  C  D



+3/12/25+