

# ENGENHARIA DE SOFTWARE

## Introdução

- OTAN - Alemanha, 1968.
  - ➔ Primeira vez que o termo "Engenharia de Software" foi usado.
  - ➔ "Certos sistemas estão colocando demandas que estão além das nossas capacidades... Em algumas aplicações não existe uma crise... Mas estamos tendo dificuldades com grandes aplicações."
- A Engenharia de Software é a área da Computação destinada a investigar os desafios e propor soluções que permitam desenvolver sistemas de software — principalmente aqueles mais complexos e de maior tamanho — de forma produtiva e com qualidade.
- Tópicos da ES:
  - ➔ Engenharia de Requisitos;
  - ➔ Projeto de Software;
  - ➔ Construção de Software;
  - ➔ Testes de Software;
  - ➔ Manutenção de Software;
  - ➔ Gerência de Configuração;
  - ➔ Gerência de Projetos;
  - ➔ Processos de Software;
  - ➔ Modelos de Software;
  - ➔ Qualidade de Software;
  - ➔ Prática Profissional;
  - ➔ Aspectos Econômicos.
- Não existe "bala de prata" na Engenharia de Software. A complexidade, conformidade, facilidade de mudanças e invisibilidade tornam a ES diferente de outras.

## Áreas do SWEBOK

- (1) Requisitos de Software:
  - ➔ Funcionais: "O que" um sistema deve fazer.
  - ➔ Não-funcionais: "Como" um sistema deve operar. Engloba desempenho, disponibilidade, capacidade, tolerância a falhas, segurança, privacidade, interoperabilidade, manutenibilidade e usabilidade.
- (2) Projeto de Software: Definição dos principais componentes de um sistema e de suas interfaces.
- (3) Construção de Software: Implementação do sistema.
  - ➔ Algoritmos e estruturas de dados;
  - ➔ Ferramentas: IDEs, depuradores, etc;
  - ➔ Bibliotecas e frameworks;
  - ➔ Padrões de nome.
- (4) Testes de Software: Verificam se um programa apresenta um resultado esperado, ao ser executado com casos de teste. Podem ser manuais ou automatizados.
  - ➔ Um código com defeito/bug causa uma falha quando é executado.
- "Testes de software mostram a presença de bugs, mas não a sua ausência."
  - Edsger W. Dijkstra.
- "All software is tested. The real question is whether the tester is you or your users."
  - Allen Holub.

- **Verificação:** Estamos implementando o sistema **corretamente**?

- De acordo com os requisitos e especificações.

- **Validação:** Estamos implementando o sistema **correto**?

- O sistema que os clientes querem.

- **(5) Manutenção de Software:**

- Corretiva (corrige erro);

- Preventiva (previne erro);

- Adaptativa (atualização de tec.);

- Refactoring (estrutura);

- Evolutiva (nova funcionalidade).

- **Sistemas Legados** são **sistemas antigos**, usando linguagens, SOs, BDs antigos.

- Manutenção custosa e arriscada.

- Legado != irrelevante (ex.: COBOL).

- **(6) Gerência de Configuração:** Todo software é desenvolvido usando um sistema de **controle de versões**.

- Atua como uma "fonte da verdade" sobre o código.

- Permite recuperar versões antigas.

- **(7) Gerência de Projetos:**

- Negociação de contratos (prazos, valores, cronograma);

- Gerência de RH (contratação, treinamento, etc);

- Gerenciamento de riscos;

- Acompanhamento da concorrência, marketing, etc.

- **Lei de Brooks:** Incluir novos devs em um projeto, que está atrasado, vai deixá-lo mais atrasado ainda.

- **Stakeholders:** Todas as "partes interessadas" em um projeto de software.

- São aqueles que **afetam** ou **são afetados** pelo projeto.

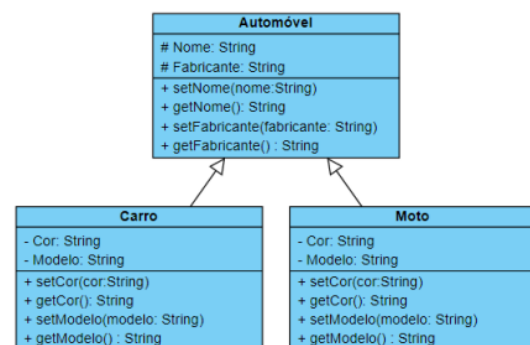
- **(8) Processo de Desenvolvimento de Software:** Define quais atividades devem ser seguidas para construir um sistema de software.

- **(9) Modelagem de Software:**

- Modelo = representação mais alto nível do que o código.

- Modelos como sketches.

- **Unified Modeling Language (UML):** Linguagem gráfica para modelagem de software.



- **(10) Qualidade de Software:**

- **Qualidade Externa:** Correção, robustez, extensibilidade, reusabilidade, eficiência, compatibilidade, facilidade de uso, portabilidade.

- **Qualidade Interna:** Modularidade, legibilidade, testabilidade, manutenibilidade, etc.

- **(11) Prática Profissional:**

- Ética;

- Certificações;

- Regulamentação da Profissão;

- ➡ Cursos de graduação;
- ➡ Currículo de Referência.

#### • (12) Aspectos Econômicos:

- ➡ Qual o retorno de investimento?
- ➡ Como monetizar meu software? (anúncios, assinaturas, etc)
- ➡ Quanto cobrar pelo desenvolvimento de um sistema?
- ➡ Qual a melhor licença para o meu software?

#### Tipos ABC de sistemas

- Classificação proposta por Bertrand Meyer.

#### • Sistemas C (Casuais):

- ➡ Tipo muito comum de sistema.
- ➡ Sistemas pequenos, sem muita importância.
- ➡ Podem ter bugs; às vezes, são descartáveis.
- ➡ Desenvolvidos por 1-2 engenheiros.
- ➡ Risco: "over-engineering".

#### • Sistemas B (Business):

- ➡ Sistemas importantes para uma organização.
- ➡ Risco: não usarem técnicas de ES e se tornarem um passivo, em vez de um ativo para as organizações.

#### • Sistemas A (Acute):

- ➡ Sistemas onde nada pode dar errado, pois o custo é imenso, em termos de vidas humanas e/ou \$\$\$.
- ➡ Também chamados sistemas de missão crítica.
- ➡ Requerem certificações.



## Processos

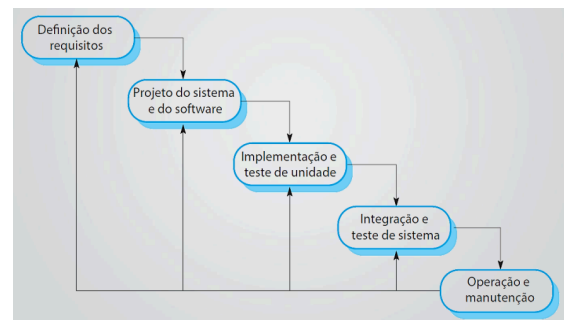
- Software é abstrato e "adaptável".

- A maior parte dos processos de software de grandes sistemas se baseia em um modelo genérico, mas frequentemente incorpora características de outros modelos.

#### Modelo em Cascata

#### • Waterfall:

- ➡ Inspirado em processos usados em engenharias tradicionais.
- ➡ Proposto na década de 70 e muito usado até ~1990.
- ➡ Adotado pelo Departamento de Defesa Norte-Americano (1985).

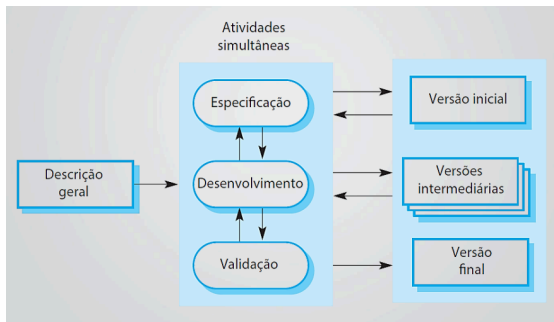


#### • Problemas com o Waterfall:

- ➡ Requisitos mudam com frequência;
- ➡ Documentações são verbosas.
- ➡ Rapidamente se tornam obsoletas.

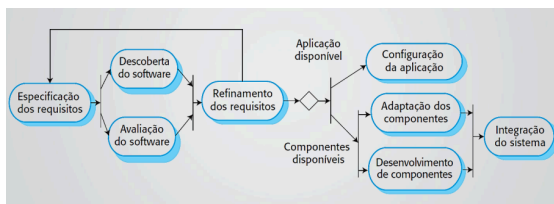
#### Modelo Incremental

- Desenvolvimento incremental:



## Modelo orientado a reuso

- ES orientada para reuso:



## Método Ágil

- Manifesto Ágil, 2001.

- ➔ Encontro de 17 engenheiros de software em Utah.
- ➔ Crítica a modelos sequenciais e pesados.

Valoriza	mais que
indivíduos e interações	processos e ferramentas
software em funcionamento	documentação abrangente
colaboração com o cliente	negociação de contratos
responder a mudanças	seguir um plano

- Método Ágil:

- ➔ Incremental e iterativo.
- ➔ Menor ênfase em documentação.

- ➔ Menor ênfase em big upfront design (planejamento detalhado).
- ➔ Processo ajuda a não cometer certos "grandes erros".

- Organização:

- ➔ Define um processo, mesmo que leve.
- ➔ Workflow, eventos, papéis, práticas, princípios, etc.
- ➔ Envolvimento constante do cliente (product owner).
- ➔ Novas práticas de programação: Testes, refactoring, integração contínua, etc.
- ➔ Processos não são adotados 100% igual ao manual.

- Quando vale a pena usar métodos ágeis:

- ➔ Para projetos com grau de "incerteza", método ágil faz mais sentido.

- Quando não vale a pena usar métodos ágeis:

- ➔ Condições do mercado são estáveis e previsíveis.
- ➔ Requisitos são claros no início do projeto e irão permanecer estáveis.
- ➔ Clientes não estão disponíveis para colaboração frequente.
- ➔ Sistema semelhante já foi feito antes; logo, já se conhece a solução.
- ➔ Problemas podem ser resolvidos sequencialmente, em silos funcionais.
- ➔ Clientes não conseguem testar partes do produto, antes de completo.
- ➔ Mudanças no final do projeto são caras ou impossíveis.
- ➔ Impacto de mudanças provisórias pode ser catastrófico.

## Extreme Programming

### • Extreme Programming (XP):

- ➔ Criado por Kent Beck, um dos signatários do Manifesto Ágil.
- ➔ XP = Valores + Princípios + Práticas.

### • Valores:

- ➔ Comunicação;
- ➔ Simplicidade;
- ➔ Feedback;
- ➔ Coragem;
- ➔ Respeito;
- ➔ Qualidade de Vida (semana 40 hrs).

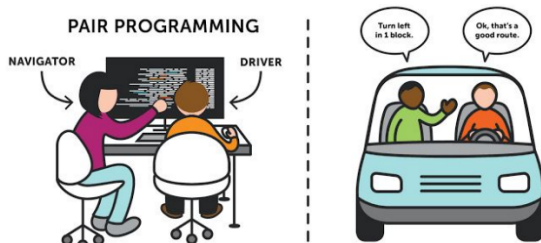
### • Princípios:

- ➔ Economicidade;
- ➔ Melhorias Contínuas;
- ➔ Falhas Acontecem;
- ➔ Baby Steps;
- ➔ Responsabilidade Pessoal.

### • Práticas:

Práticas sobre o Processo de Desenvolvimento	Práticas de Programação	Práticas de Gerenciamento de Projetos
Representante dos Clientes Histórias de Usuário Iterações Releases Planejamento de Releases Planejamento de Iterações Planning Poker Slack	Design Incremental Programação Pareada Testes Automatizados Desenvolvimento Dirigido por Testes (TDD) Build Automatizado Integração Contínua	Ambiente de Trabalho Contratos com Escopo Aberto Métricas

### • Pair Programming:



### • Vantagens:

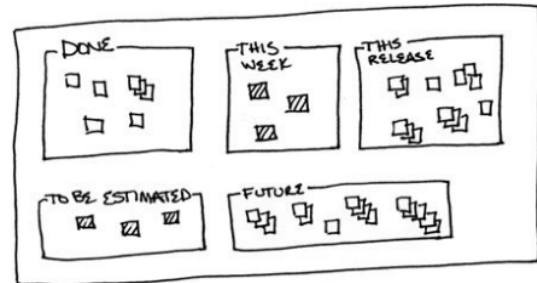
- ➔ Redução de bugs;
- ➔ Código de melhor qualidade;
- ➔ Disseminação de conhecimento;

- ➔ Aprendizado com os pares.

### • Desvantagem:

- ➔ Custo.

### • Ambiente de Trabalho:



### • Contratos com Escopo Aberto:

- ➔ Exige maturidade e acompanhamento do cliente.
- ➔ Privilegia qualidade.
- ➔ Não vai ser "enganado" ("entregar por entregar").
- ➔ Pode mudar de fornecedor.

### • Escopo fechado:

- ➔ Cliente define requisitos ("fecha escopo");
- ➔ Empresa desenvolvedora: preço + prazo.

### • Escopo aberto:

- ➔ Escopo definido a cada iteração;
- ➔ Pagamento por trabalhador/hora;
- ➔ Contrato renovado a cada iteração.

## Scrum

### • Scrum:

- ➔ Proposto por Jeffrey Sutherland e Ken Schwaber (OOPSLA 1995).

### • Scrum vs XP:

- ➔ Scrum não é apenas para projetos de software, logo, não define práticas de programação, como XP.

➡ Scrum define um "processo" mais rígido que XP, com eventos, papéis e artefatos bem claros.

#### • Sprint (evento):

➡ Como em qualquer método ágil, desenvolvimento é dividido em sprints (iterações).

➡ Duração de um sprint: até 1 mês, normalmente 15 dias.

➡ Implementa-se algumas histórias dos usuários (funcionalidades do sistema).

#### • Time Scrum (papéis):

➡ Todos têm o mesmo nível hierárquico.

➡ Os devs têm autonomia para dizer que não vão conseguir implementar tudo que o PO quer em um único sprint.

➡ "Duas pizzas": 5 a 11 membros, sendo 1 Product Owner, 3 a 9 desenvolvedores e 1 Scrum Master.

#### • Product Owner (PO):

➡ Membro (papel) obrigatório em times Scrum.

➡ Especialista no domínio do sistema.

➡ Escreve as histórias de usuário.

➡ Faz a ponte entre os stakeholders e os devs.

➡ Explica histórias para os devs, durante o sprint.

➡ Define "testes de aceitação" de histórias.

➡ Mantém o backlog do produto.

#### • Scrum Master:

➡ Especialista em Scrum, ajuda o time a seguir Scrum e seus eventos.

➡ Removedor de "impedimentos" não-técnicos.

➡ Pode pertencer a mais de um time.

#### • Backlog do Produto (artefato):

➡ Lista de histórias do usuário, que foram escritas pelo PO.

➡ Priorizada: histórias do topo têm maior prioridade.

➡ Dinâmica: histórias podem sair e entrar, à medida que o sistema evolui.

#### • Backlog do Sprint (artefato):

➡ Lista de tarefas do sprint, com responsáveis e duração estimada.

#### • Planejamento do sprint:

➡ Reunião para decidir quais histórias vão entrar no próximo sprint.

➡ PO propõe histórias que gostaria de ver implementadas.

➡ Devs decidem se têm velocidade para implementá-las.

➡ Primeiro, definem-se as histórias do sprint. Em seguida, as histórias são quebradas em tarefas e as tarefas são alocadas a devs.

#### • Reuniões Diárias:

➡ 15 minutos de duração, onde cada participante diz o que fez ontem, o que pretende fazer hoje e se está tendo alguma dificuldade.

➡ Feitas com o objetivos de melhorar a comunicação e antecipar problemas.

#### • Revisão do Sprint:

➡ Time mostra o resultado do sprint para PO e stakeholders.

➡ A implementação das histórias pode ser aprovada, aprovada parcialmente ou reprovada.

➡ Nos dois últimos casos, a história volta para o backlog do produto.

#### • Retrospectiva do Sprint:

➡ Último evento do sprint.



➔ Time se reúne para decidir o que melhorar.

#### • Time-box:

➔ As atividades têm uma duração bem definida.






Evento	Time-box
Planejamento do Sprint	máximo de 8 horas
Sprint	menos de 1 mês
Reunião Diária	15 minutos
Revisão do Sprint	máximo de 4 horas
Retrospectiva	máximo de 3 horas

#### • Critérios para conclusão de histórias:

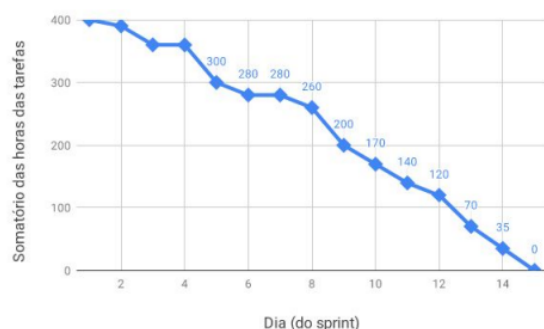
➔ Qualidade externa: Testes de aceitação (caixa preta ou funcionais) e testes não-funcionais (ex.: desempenho, usabilidade).

➔ Qualidade interna: Testes de unidade e revisão de código.

#### • Scrum Board:

Backlog	To Do	Doing	Testing	Done
				

#### • Burndown chart:



#### • Story Points:

➔ Unidade (inteiro) para comparação do tamanho de histórias.

➔ Definição de story points é "empírica".

➔ A velocidade de um time é definida pelo número de story points que o time consegue implementar em um sprint.

#### Kanban

#### • Kanban:

➔ Origem na década de 50 no Japão.

➔ Sistema de Produção da Toyota.

➔ Manufatura Lean.

➔ Produção just-in time.

➔ Kanban é mais adequado para times mais maduros.

#### • Kanban vs Scrum:

➔ Kanban é mais simples.

➔ Não existem sprints.

➔ Não é obrigatório usar papéis e eventos.

➔ Time define os papéis e eventos.

Backlog	Especificação WIP		Implementação WIP		Revisão de Código WIP	
	em espec.	especificadas	em implementação	implementadas	em revisão	revisadas

#### • Sistema pull:

➔ Os membros "puxam" trabalho.

➔ Escolhem uma tarefa para trabalhar e, quando concluem a tarefa, a movem para frente no quadro.

#### • Limites WIP (Work in Progress):

➔ Número máximo de tarefas em um passo.

➔ Contando: tarefas em andamento e concluídas.

#### • Objetivos dos Limites WIP:

- ➡ Criar um fluxo de trabalho sustentável, evitando que o time fique sobrecarregado de trabalho.
- ➡ Evitar que o trabalho fique concentrado em um passo.

#### • WIP de Especificação:

- ➡ Número de histórias em especificação + número de grupos de tarefas especificadas.
- ➡ Tarefas na mesma linha = resultantes da mesma história.

#### • WIP de Revisão:

- ➡ Conta apenas tarefas na primeira coluna (em revisão).

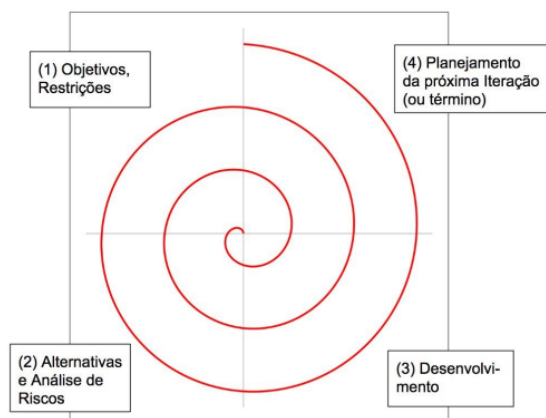
A transição

#### • Transição de Waterfall para Ágil:

- ➡ Antes da disseminação dos princípios ágeis, alguns métodos iterativos ou evolucionários foram propostos.
- ➡ Transição Waterfall (~1970) e Ágil (~2000) foi gradativa.

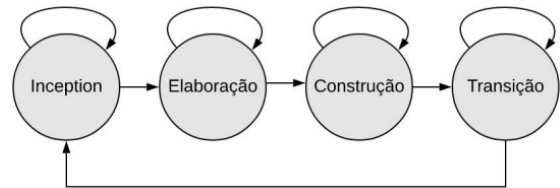
#### • Espiral:

- ➡ Proposto por Barry Boehm.
- ➡ Iterações: 6 a 24 meses (logo, mais que em XP e Scrum).



#### • RUP:

- ➡ Proposto pela Rational, depois comprada pela IBM.
- ➡ Duas características principais: Diagramas UML e ferramentas CASE (Computer-Aided Software Engineering).



- ➡ Inception: análise de viabilidade.
- ➡ Elaboração: requisitos + arquitetura.
- ➡ Construção: projeto de baixo nível + implementação.
- ➡ Transição: implantação (deployment).



## Atividades do Processo

- Os processos de software reais são **seqüências intercaladas de atividades técnicas, colaborativas e gerenciais**, cujo objetivo global é especificar, projetar, implementar e testar um sistema de software.

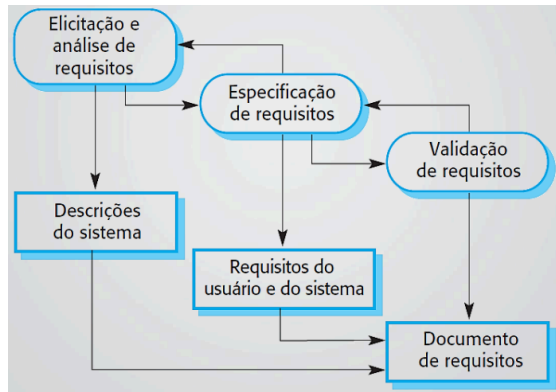
#### • Atividades de processo básicas:

- ➡ Especificação;
- ➡ Desenvolvimento;
- ➡ Validação;
- ➡ Evolução.

Especificação

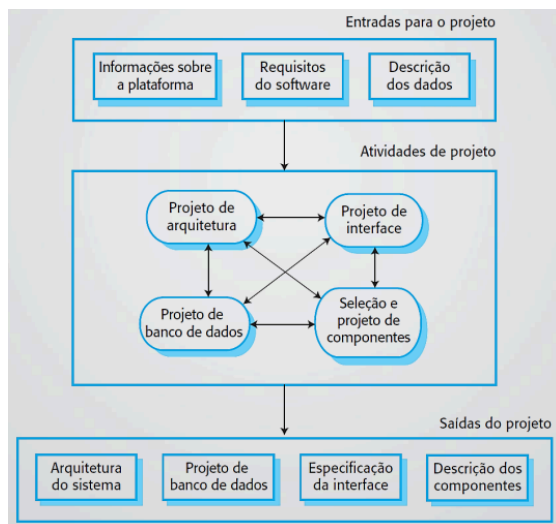


- O processo de engenharia de requisitos:



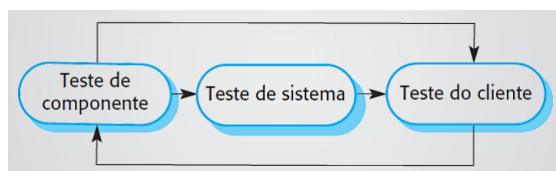
## Desenvolvimento

- Projeto e implementação do software:

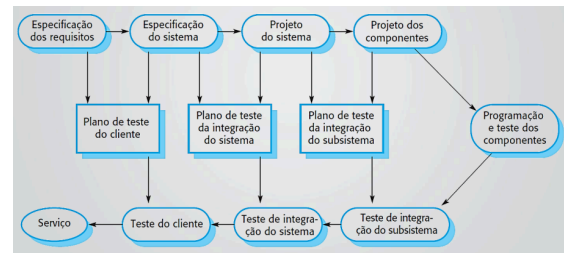


## Validação

- Estágios do teste:

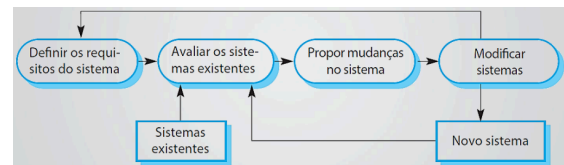


- Fases de teste em um processo de software dirigido por plano:



## Evolução

- Evolução do sistema de software:



## Lidando com mudanças

- A mudança é inevitável em todos os grandes projetos de software.

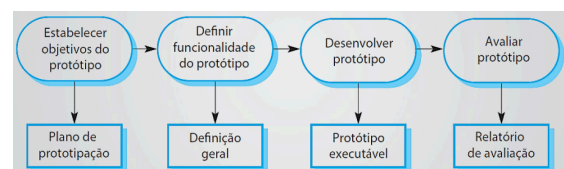
- Duas abordagens relacionadas podem ser utilizadas para reduzir os custos de retrabalho:

- Antecipação da mudança.
- Tolerância à mudança.

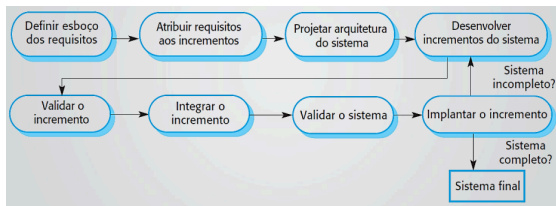
- Duas maneiras de lidar com as mudanças e com as variações nos requisitos do sistema são:

- Prototipação do sistema.
- Entrega incremental.

- Desenvolvimento de um protótipo:

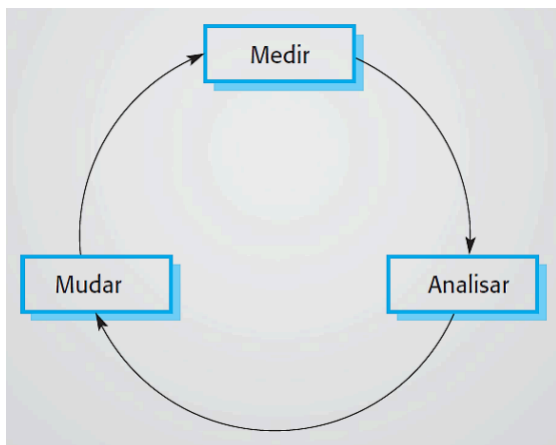


- Entrega incremental:

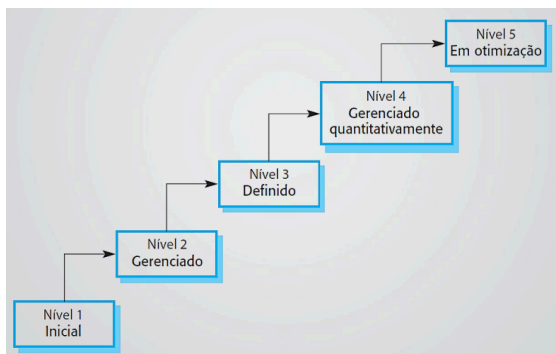


## Melhoria de processo

- O ciclo de melhoria do processo:



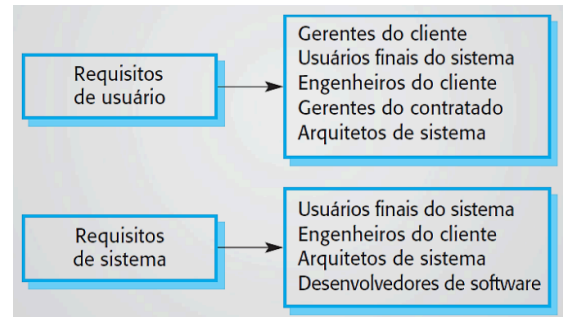
- Níveis de maturidade da capacidade:



## Engenharia de Requisitos

- "A parte mais difícil da construção de um software é a definição do que se deve construir".  
- Fred Brooks.

- Leitores dos diferentes tipos de especificação de requisitos:



- Requisitos funcionais:

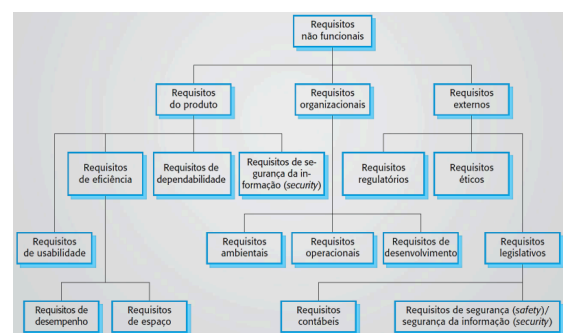
➔ São **declarações dos serviços que o sistema deve fornecer**, do modo como o sistema deve reagir a determinadas entradas e de como deve se comportar em determinadas situações.

- Requisitos não funcionais:

➔ São **restrições sobre os serviços ou funções oferecidas pelo sistema**.

➔ Os requisitos não funcionais se aplicam, frequentemente, ao **sistema como um todo**, em vez de às características individuais ou aos serviços.

- Tipos de requisitos não funcionais:

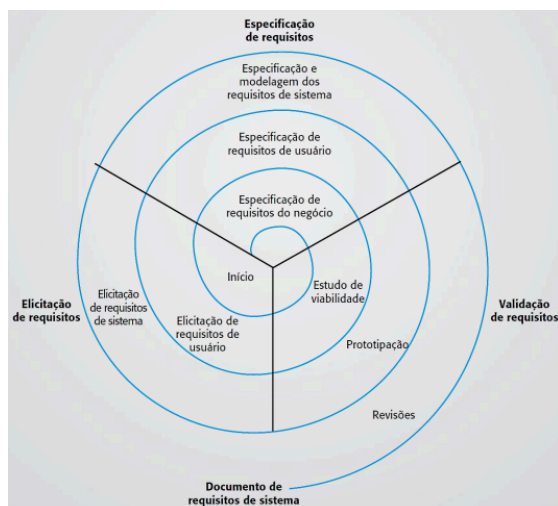


- Métricas para especificar requisitos não funcionais:

Propriedade	Métrica
Velocidade	Transações processadas/segundo Tempo de resposta do usuário/evento Tempo de atualização da tela
Tamanho	Megabytes/número de chips de ROM
Facilidade de uso	Tempo de treinamento Número de quadros de ajuda
Confiabilidade	Tempo médio até a falha Probabilidade de indisponibilidade Taxa de ocorrência de falhas Disponibilidade
Robustez	Tempo para reiniciar após a falha Porcentagem de eventos causando falhas Probabilidade de corromper dados em uma falha
Portabilidade	Porcentagem de declarações dependentes do sistema-alvo Número de sistemas-alvo

## Processos de engenharia de requisitos

- Uma visão em espiral do processo de engenharia de requisitos:



## Elicitação de requisitos

- Descobrir e entender os principais requisitos do sistema que se pretende construir.

### Desafios:

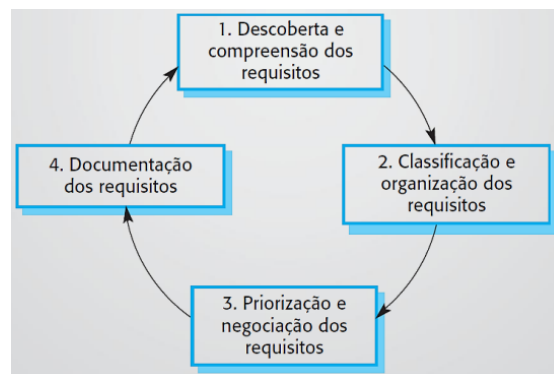
- ➔ Muitas vezes os stakeholders não sabem o que querem.
- ➔ É natural que os stakeholders expressem os requisitos em seus próprios termos e com conhecimento implícito de seu próprio trabalho.

➔ Diferentes stakeholders, com requisitos distintos, podem expressá-los de maneiras variadas.

➔ Fatores políticos podem influenciar os requisitos de um sistema.

➔ O ambiente econômico e de negócios no qual a análise ocorre é dinâmico.

- O processo de elicitação e análise de requisitos:



### Técnicas:

➔ Entrevistas, em que há uma conversa com as pessoas a respeito do que elas fazem.

➔ Observação ou etnografia, em que se observa as pessoas executando seu trabalho para ver quais artefatos elas usam, como usam, etc.

## Histórias e cenários

- Descrição de como o sistema pode ser utilizado em alguma tarefa em particular.

➔ Descrevem o que as pessoas fazem, quais informações usam e produzem e quais sistemas podem adotar nesse processo.

- As histórias são escritas como texto narrativo e apresentam uma descrição

de alto nível do uso do sistema; os cenários normalmente são estruturados com informações específicas coletadas, como entradas e saídas.

- **Formato de escrita de histórias:**

- Como um [certo tipo de usuário], eu gostaria de [realizar algo com o sistema].

- Características de boas histórias (INVEST).

- Independente;
- Aberta para **Negociação**;
- Agrega **Valor**;
- Estimável;
- Sucinta;
- Testável.

- **3C's:**

- **Cartão**: "lembrete" para conversar sobre o requisito durante o sprint.
- **Conversa**: diálogo estabelecido a partir do cartão.
- **Confirmação**: cenários que serão usados pelo PO para aceitar a implementação da história (escritos no verso do cartão).

## Especificação de requisitos

- **Notações para escrever requisitos de sistema:**

Notação	Descrição
Sentenças em linguagem natural	Os requisitos são escritos usando frases numeradas em linguagem natural. Cada frase deve expressar um requisito.
Linguagem natural estruturada	Os requisitos são escritos em linguagem natural em um formulário ou <i>template</i> . Cada campo fornece informações sobre um aspecto do requisito.
Notações gráficas	Modelos gráficos, suplementados por anotações em texto, são utilizados para definir os requisitos funcionais do sistema. São utilizados com frequência os diagramas de casos de uso e de sequência da UML.
Especificações matemáticas	Essas notações se baseiam em conceitos matemáticos como as máquinas de estados finitos ou conjuntos. Embora essas especificações inequívocas possam reduzir a ambiguidade em um documento de requisitos, a maioria dos clientes não compreende uma especificação formal. Eles não conseguem averiguar se ela representa o que desejam e relutam em aceitar essa especificação como um contrato do sistema (discutirei essa abordagem no Capítulo 10, que aborda a dependabilidade do sistema).

- **Especificação em linguagem natural:**

- Criar um formato padrão.
- Distinguir requisitos obrigatórios e desejáveis.
- Destacar partes importantes.
- Não supor que os leitores compreendem a linguagem técnica.
- Associar um racional a cada requisito de usuário.

- **Especificação estruturada:**

- Um template é empregado.
- Descrição da função ou entidade especificada.
- Descrição das entradas e suas origens.
- Descrição das saídas e sua destinação.
- Dados necessários para computar.
- Descrição da ação a ser tomada.
- Descrição dos efeitos colaterais da operação.
- Se for utilizada uma abordagem funcional, uma pré-condição estabelecendo o que deve ser verdadeiro antes da função ser invocada e uma pós-condição especificando o que é verdadeiro após a função ser invocada.
- As tabelas são úteis quando existe uma série de possíveis situações alternativas e é preciso descrever as ações a serem tomadas para cada uma delas.

## Casos de uso

- Documento mais detalhado de especificação de requisitos.

- Um ator realizando alguma operação com o sistema.
- Incluem fluxo normal e extensões (exceções/erros e detalhamento).

- Uso não é tão comum com métodos ágeis.

#### Transferir Valores entre Contas

Ator: Cliente do Banco

Fluxo normal:

- 1 - Autenticar Cliente
- 2 - Cliente informa agência e conta de destino da transferência
- 3 - Cliente informa valor que deseja transferir
- 4 - Cliente informa a data em que pretende realizar a operação
- 5 - Sistema efetua transferência
- 6 - Sistema pergunta se o cliente deseja realizar uma nova transferência

Extensões:

- 2a - Se conta e agência incorretas, solicitar nova conta e agência
- 3a - Se valor acima do saldo atual, solicitar novo valor
- 4a - Data informada deve ser a data atual ou no máximo um ano a frente
- 5a - Se data informada é a data atual, transferir imediatamente
- 5b - Se data informada é uma data futura, agendar transferência



#### Documento de requisitos

- Documento de requisitos de software:

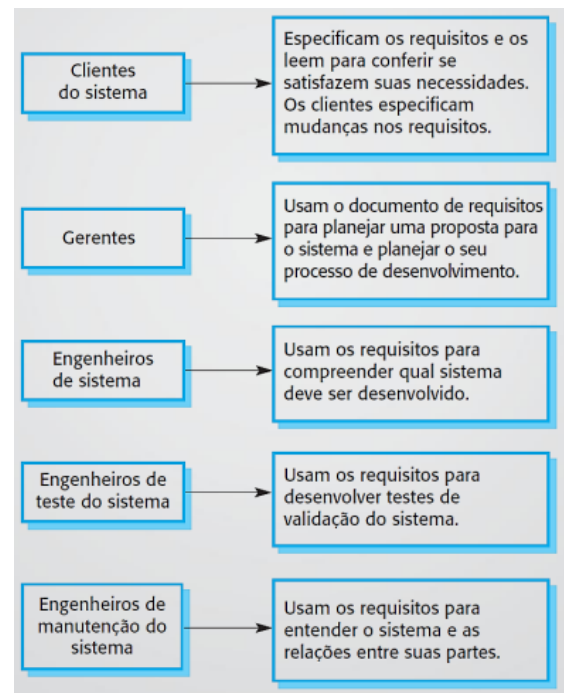
➡ Declaração oficial do que os desenvolvedores do sistema devem implementar.

➡ Pode incluir os requisitos de usuário para um sistema e uma especificação detalhada dos requisitos de sistema.

➡ Às vezes, os requisitos de usuário e de sistema são integrados em uma única descrição.

➡ Em outros casos, os requisitos de usuário são descritos em um capítulo introdutório na especificação de requisitos de sistema.

- Usuários de um documento de requisitos:



#### Validação de requisitos

- Conferências executadas:

- ➡ Conferência da validade;
- ➡ Conferência da consistência;
- ➡ Conferência da completude;
- ➡ Conferência do realismo;
- ➡ Verificabilidade.

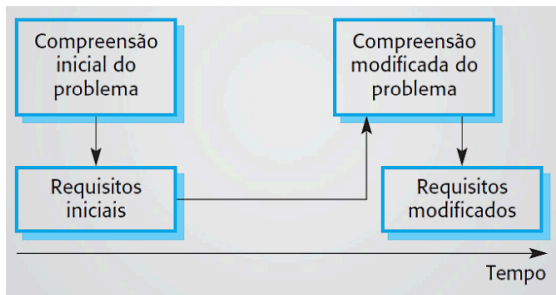
- Uma série de técnicas de validação de requisitos pode ser utilizada individualmente ou em conjunto:

- ➡ Revisões de requisitos.
- ➡ Prototipação.
- ➡ Geração de casos de teste.

- Os usuários precisam imaginar o sistema em operação e como ele se encaixaria em seu trabalho.

#### Mudança de requisitos

- Evolução dos requisitos:



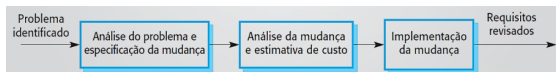
## Planejamento do gerenciamento de requisitos

### • Questões:

- Identificação dos requisitos;
- Processo de gerenciamento de mudança;
- Políticas de rastreabilidade;
- Apoio de ferramentas.

### • É preciso de apoio de ferramentas para:

- Armazenamento de requisitos;
- Gerenciamento de mudança;
- Gerenciamento da rastreabilidade.



## Gerenciamento de requisitos

- Deve ser aplicado a todas as mudanças propostas para os requisitos de um sistema após a aprovação do documento de requisitos.

### • Estágios principais:

- Análise do problema e especificação da mudança;
- Análise da mudança e estimativa de custo;
- Implementação da mudança.

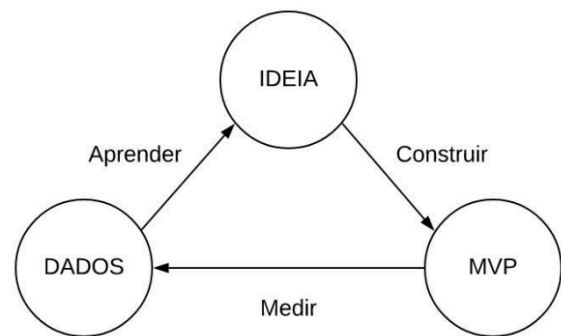
## Produto Mínimo Viável

- The Lean Startup, Eric Ries.

### • MVP:

- Produto: já pode ser usado.
- Mínimo: menor conjunto de funcionalidades essenciais.
- Viável: o objetivo é obter dados e validar uma ideia.
- Não precisa usar todas as melhores práticas de ES.

### • Ciclo Construir-Medir-Aprender:



### • Baixo risco e usuários conhecidos:

- Sistema conhecido.
- Necessidade e viabilidade são óbvias
- Histórias de usuários funcionam bem!

### • Alto risco e "sucesso" incerto:

- Sistemas típicos de startups, mas não exclusivos.
- Como o risco é alto, a implementação (ideia) tem que ser rapidamente validada com usuários reais.

## Testes A/B

- Cenário onde duas implementações de requisitos "competem" entre si.

- Versão original vs nova versão.



➡ Vale a pena migrar para a nova versão?

➡ Testes A/B requerem amostras grandes.

• Versões de controle e tratamento:

➡ Versão A: controle.

➡ Versão B: tratamento.

```
Unset
version = Math.Random();

if(version < 0.5)
    "versão de controle"
else
    "versão de tratamento"
```

• Final do experimento:

➡ Analisam-se os dados, isto é, alguma métrica.

➡ Versão B introduz ganhos estatisticamente relevantes?

• Calculadora de tamanho de amostras:

➡ Informa-se a taxa de conversão atual e o ganho pretendido.

➡ A calculadora informa o tamanho necessário da amostra.

