Engenharia de Software

1 – A disciplina de Engenharia de Software

É recomendável que o desenvolvimento de um software complexo ocorra de forma disciplinada uma vez que envolve o trabalho de muitas pessoas, restrições de tempo e de recursos.

A chance de atraso, descaminhos e erros é sempre muito grande em função de mudanças que podem ocorrer ao longo do caminho.

Com o amadurecimento da área de desenvolvimento de software nas empresas, notou-se que uma série de práticas poderiam melhorar o produto final, tanto em termos de metodologias de desenvolvimento, quanto na qualidade do próprio produto.

Várias metodologias foram se consagrando, algumas com mais valor acadêmico, enquanto outras foram sendo desenvolvidas e praticadas diretamente pelas empresas. Muitas vezes a experiência empírica revela quais as melhores práticas.

Chamamos de **Engenharia de Software** a disciplina da computação que reúne metodologias, recursos e ferramentas a serem utilizadas desde a percepção de um problema até o momento em o sistema desenvolvido torna-se operacional. Visa resolver problemas inerentes ao processo de desenvolvimento e ao produto de software buscando-se sempre:

- Baixar custo

- Produção mais rápida

- Melhoria da qualidade.

2 – Requisitos de desenvolvimento

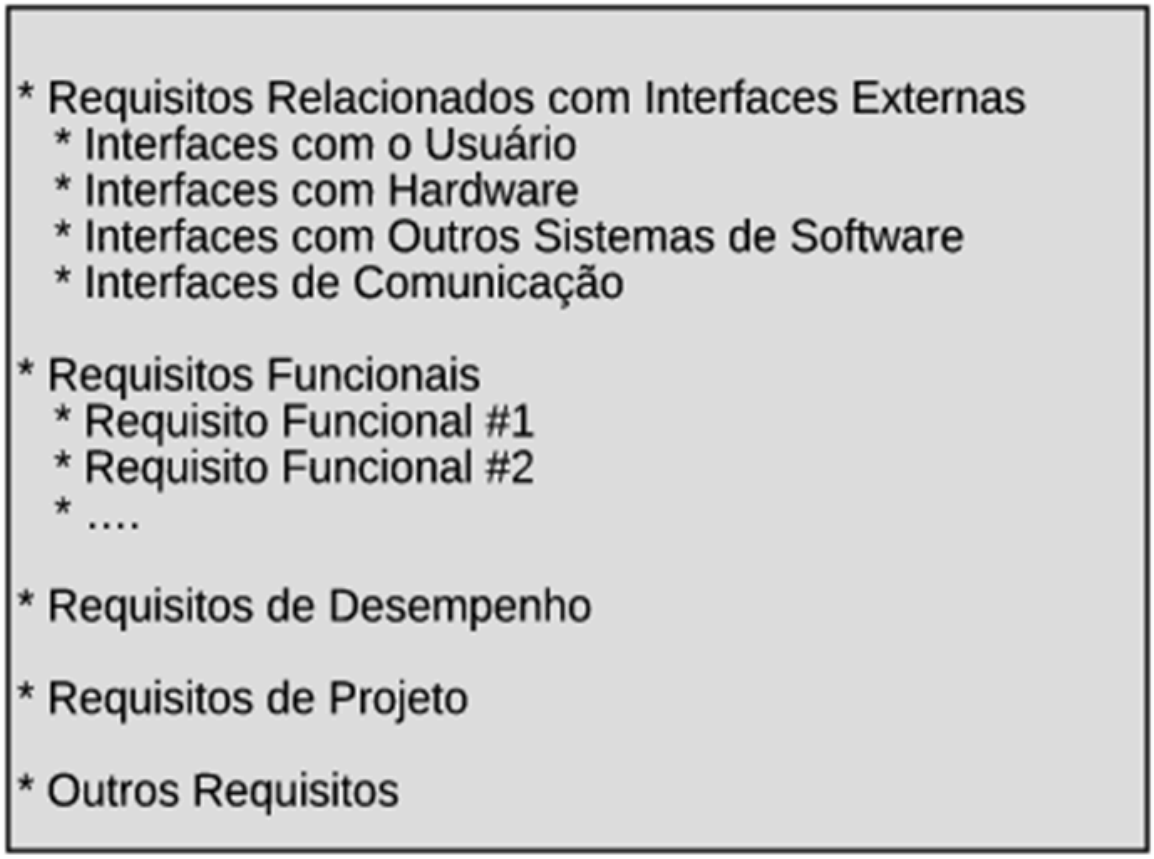
Requisitos correspondem ao conjunto de necessidades do usuário em relação ao sistema. Requisitos podem ser **funcionais** (o que o sistema deveria fazer) e **não funcionais** (restrições, isto é, o que o sistema não deveria fazer).

Exemplo: em um sistema bancário, informar o extrato de uma conta, realizar transferências, pagar contas são requisitos funcionais). No mesmo sistema, questões de desempenho, nível de segurança, portabilidade...são requisitos não funcionais.

A **extração de requisitos** é uma das tarefas mais desafiadoras na área de desenvolvimento de sistemas, envolvendo várias técnicas tais como entrevistas, formulários, workshops, documentação, análise de cenários, etc.

A extração de requisitos pode resultar em um documento de requisitos definido pelo padrão IEEE 830.

Documento de requisitos no padrão IEEE830



Um documento de requisitos protege tanto o desenvolvedor quanto o cliente. O cliente tem em um documento o que espera encontrar e o desenvolvedor sabe o que deve entregar.

“Entrevista” como técnicas de levantamento de requisitos

Entrevista é uma das técnicas mais utilizadas por ser relativamente simples e de baixo custo.

Envolve uma série de detalhes de planejamento, sendo recomendado, por exemplo, que o entrevistador prepare as perguntas antecipadamente e que considere que pode ter uma longa duração e se repetir muitas vezes.

Um dos desafios é exatamente fazer com que o entrevistador fique “neutro”, ou seja, não utilize sua experiência para antecipar uma solução e induzir o usuário a aceitá-la.

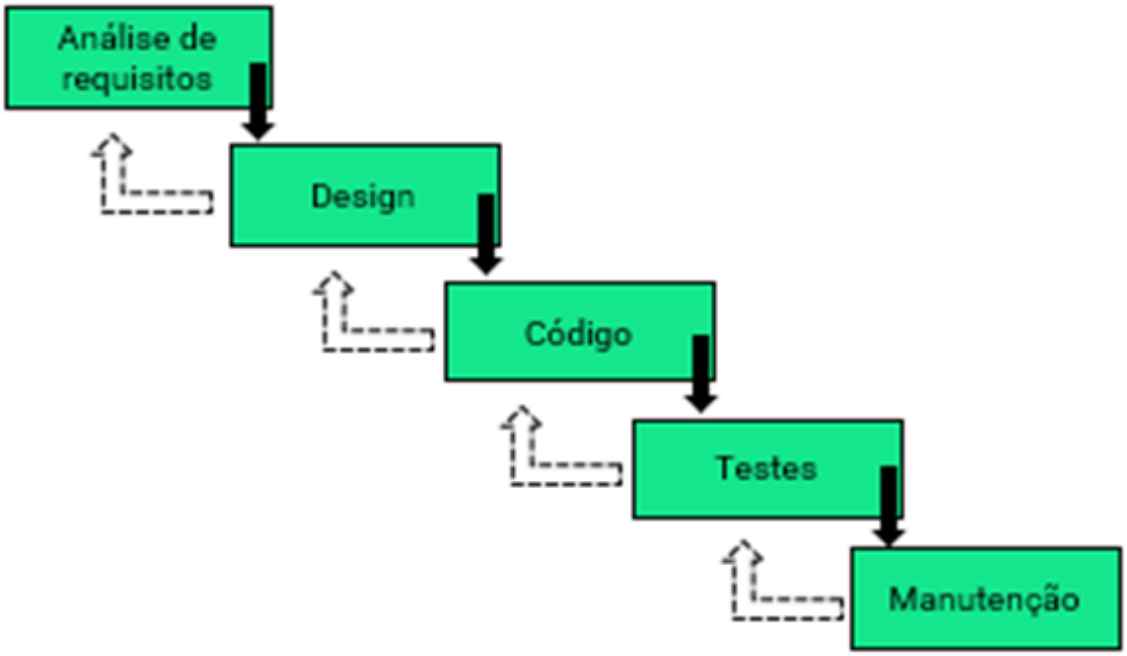
3 - Ciclos de Vida e modelos

O ciclo de vida de um software é uma estrutura que indica processos e atividades envolvidas no desenvolvimento, operação e manutenção de um software, abrangendo, de fato, toda a vida do sistema. Neste ciclo, foram imaginados modelos que definem como o software será desenvolvido, lançado, aprimorado e finalizado.

Ciclo de vida é, portanto, o conjunto de etapas para o desenvolvimento da aplicação, desde o levantamento das necessidades do usuário, até a entrega do produto. Mesmo após a entrega, o ciclo pode continuar se desenvolvendo, se estiver trabalhando no sistema.

Modelo clássico

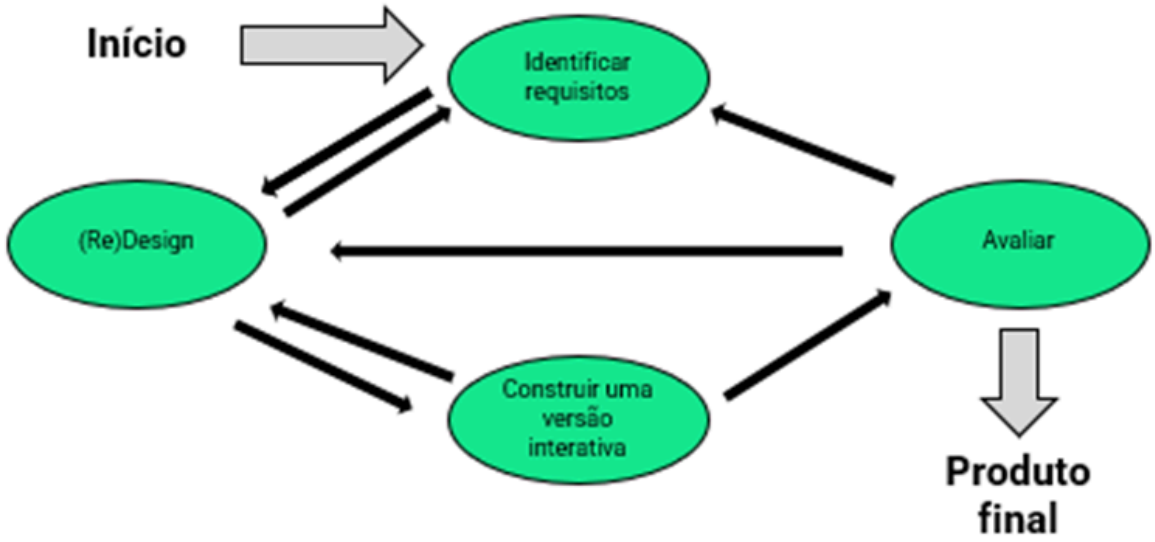
A figura abaixo procura ilustrar uma percepção tradicional sobre um ciclo de vida de um sistema. Esta primeira abordagem é denominada de desenvolvimento clássico. Esta abordagem sugere que as etapas de desenvolvimento sejam encadeadas de forma linear, uma etapa precedendo a outra. Tudo começa pela extração de requisitos, na sequência o design ou projeto, em seguida a codificação, depois testes sobre o código e finalmente a etapa de manutenção.



Esta visão clássica de desenvolvimento prevê retornos e revisões mas de forma linear. Por exemplo, testes podem levar a uma necessidade de melhoria de código que remete a uma revisão de design e até uma verificação dos requisitos. O processo costuma ser bem controlado e organizado, mas pode ser muito demorado na prática. Além disso, não deixa clara a participação do cliente, que fica um tanto restrita à etapa de análise e extração de requisitos.

Modelo evolucionário

Um outro modelo de ciclo de vida bastante conhecido é denominado modelo evolutivo ou evolucionário (ou incremental). Nesse modelo existem trocas mais claras entre as etapas e diferentes protótipos do sistema são desenvolvidos até que se tenha uma versão madura que corresponda ao produto final. Diferente do modelo clássico anterior, o modelo evolutivo costuma ser mais ágil e rápido mas pode implicar um desenvolvimento mais frágil em termos de controles e documentação.



Modelo em Espiral

O modelo em espiral é um processo evolucionário, ou seja, adequado para softwares que precisam passar por inúmeras evoluções na medida que o desenvolvimento acontece. Mas diferente do Modelo Incremental que entrega partes prontas uma de cada vez, o Modelo Espiral deixa mais claros os sucessivos refinamentos. Outras novidades são os conceitos de Prototipagem e Gerenciamento de Riscos, constantes e embutidos em cada nova versão.

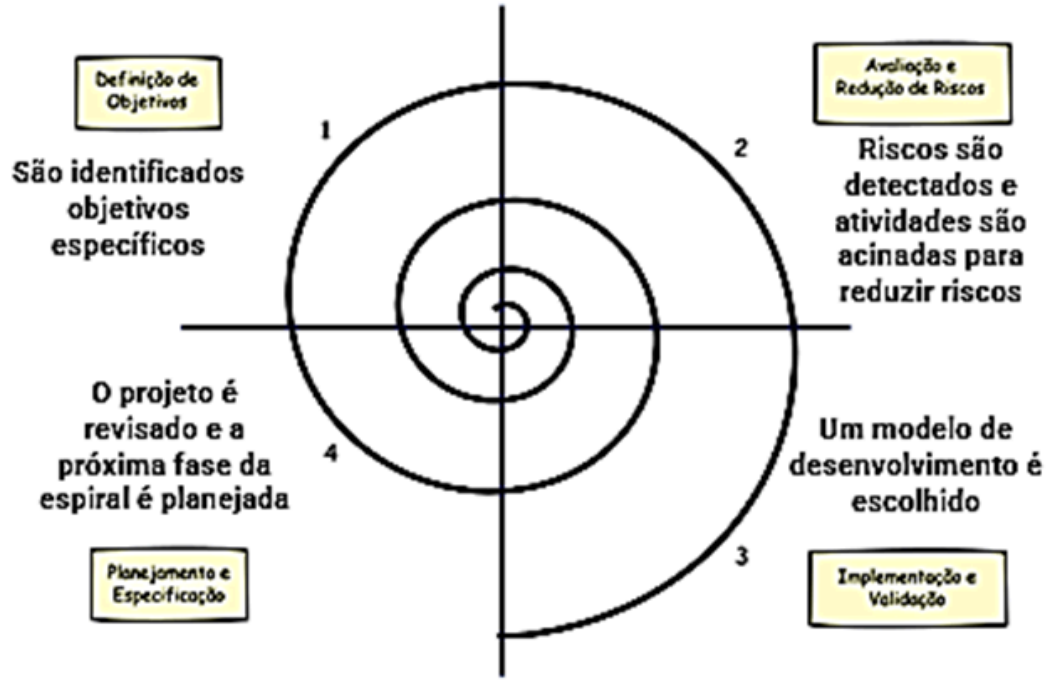


Imagem: https://medium.com/contexto-delimitado/o-modelo-em-espiral-de-boehm-ed1d85b7df(adaptado)

O modelo em espiral “combina prevenção e tolerância a mudanças, assume que mudanças são um resultado de riscos de projeto e inclui atividades explícitas de gerenciamento de riscos para sua redução”. É visto como “uma abordagem realista do desenvolvimento de sistemas e softwares de grande porte ... usando a prototipagem como mecanismo de redução de riscos”. Note na figura que a cada nova versão do sistema, as etapas se sucedem de forma um tanto sequencial e organizada: objetivos são revistos, risco são avaliados, ocorre validação, um novo planejamento, inicia-se uma nova versão e assim por diante.

4 - **Exemplos de ferramentas de análise e modelagem**

A engenharia de software admite diferentes abordagens, tanto com prioridade a aspectos mais conceituais ou até documentais, quanto à ênfase na análise de sistemas ou modelagem propriamente. Nesta última visão, prioriza-se fortemente o uso de ferramentas tanto clássicas quanto mais atuais, dados que a área em si sofre constantes revisões. Em geral nos cursos de Computação as ferramentas de modelagem tem grande destaque, apenas enumeramos algumas aqui dada a importância na engenharia de software.

A modelagem clássica de diagrama de fluxo de dados pode ser ainda encontrada. Veja abaixo uma representação dela. Pedidos completos dependem da entidade *Constrói pedido* mas esta depende de informações do cliente e da solicitação do pedido, daí as entidades *Processa novo cliente* e *Consulta mercadorias*. Esta última lê de um *Banco de dados de mercadorias* as informações das mercadorias.

Uma outra ferramenta bastante comum na construção de sistemas e prevista em muitos livros é MER modelo de entidade e relacionamento, voltado para banco de dados. Colocamos aqui apenas a título de ilustração: *um professor leciona em uma classe, um estudante frequenta várias classes*

Outra forma de enxergar uma parte do sistema é representada abaixo. *Um cliente preenche um formulário de pedido. Muitos clientes pesquisam um catálogo...*

5 - Garantia de Qualidade

Garantia de qualidade significa a definição de procedimentos, processos e padrões que visam reforçar que a qualidade de software seja atingida. A garantia de qualidade também inclui todo o gerenciamento de configuração, atividades de verificação e validação aplicadas após o produto ter sido entregue por uma equipe de desenvolvimento.

Verificação: Consiste em avaliar se existem falhas e problemas com o **software** (seja no código, nas funcionalidades, interface, dentre muitos outros detalhes) antes que ele seja entregue ao cliente ou disponibilizado para o público final.

Validação: A **validação de software** é um processo que comprova documentalmente que o sistema cumpre com as funções das quais foi designado, em conformidade com as especificações dos requisitos do usuário e com a garantia de segurança e rastreabilidade de informações.

Tipos de Testes

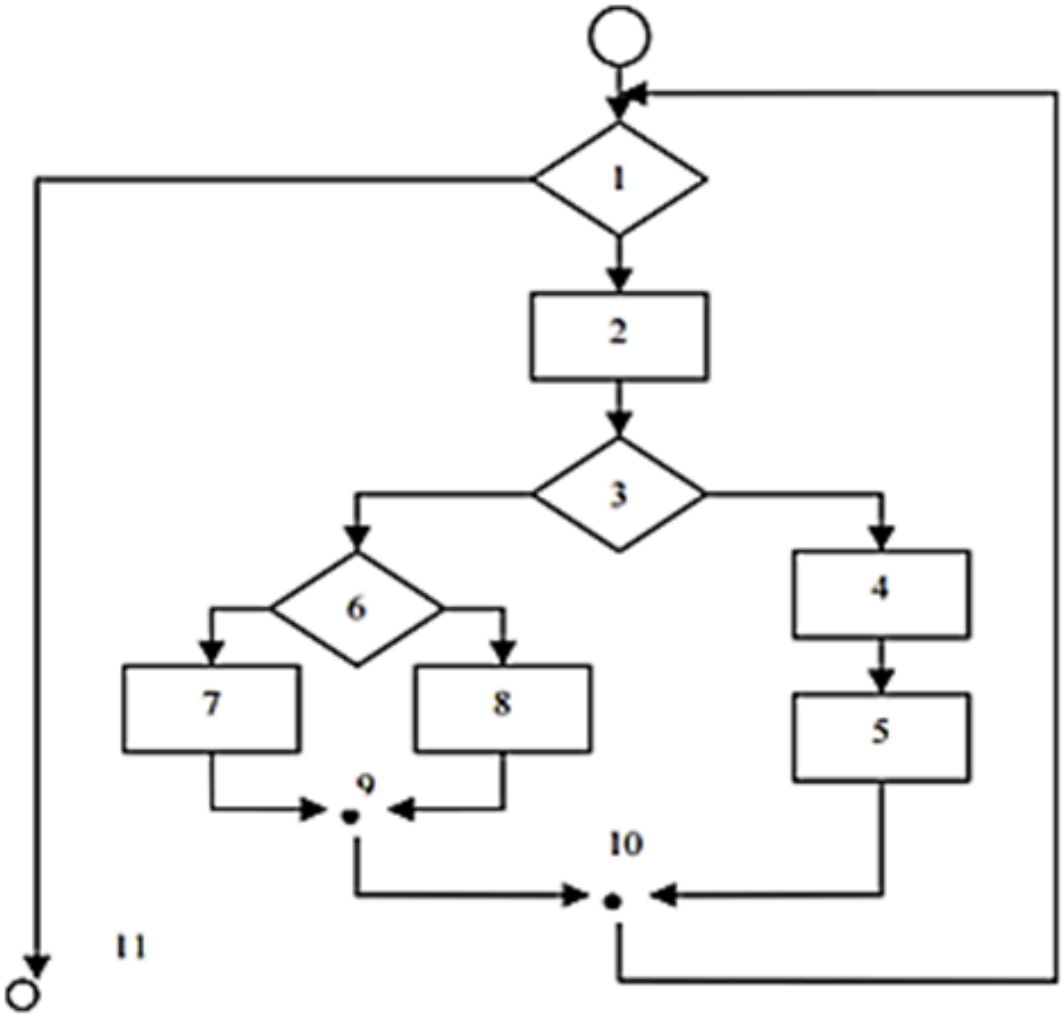
Existem muitos tipos de testes de software, sendo esta uma das áreas que mais tem crescido no escopo da engenharia de software.

Entre os muitos tipos de testes, podemos apontar testes do tipo:

- Caixa Branca
- Caixa Preta
- Regressão
- Usabilidade
- Segurança
- Integração
- Performance
- Instalação
- Manutenção
- Funcional

Teste do tipo caixa branca

Neste teste o usuário tem acesso ao funcionamento interno do sistema. O usuário analisa por qual caminho ocorre o fluxo de dados e é possível verificar se há a passagem correta em todas as condições esperadas. Na representação abaixo, existem inúmeros caminhos possíveis, entre eles os caminhos 1, 11; 1,2,3,4,5,10,1, 11; 1,2,3,6,8,9,10, 1, 11 entre outros. Um dos objetivos deste teste é verificar, por exemplo, se todas as partes do sistema serão executadas sob determinadas circunstâncias.



Testes do tipo caixa preta

Neste tipo de teste o usuário não tem acesso ao código fonte e nem a sua estrutura. Como é baseado nos requisitos funcionais, ele também é chamado de **teste funcional.**

Imagine uma situação em que o sistema prevê a entrada de 4 a 10 valores e que cada valor pode estar somente no intervalo entre 10.000 e 99.999.

O teste do tipo caixa preta busca detectar neste caso como o sistema reagirá mediante diferentes números de entrada e diferentes valores. Para que os testes não durem uma eternidade, são projetadas *partições de equivalência*. Em cada partição de equivalência, o comportamento deve ser o mesmo, fazendo com que o número de casos de testes para aplicar no sistema caia até um número satisfatório de casos.

Referência Bibliográfica

PRESSMAN, R.S. **Engenharia de Software: uma abordagem profissional.** Porto Alegre: AMG, 2011.

SOMMERVILLE, I. **Engenharia de Software.** São Paulo: Pearson, 2018.

Ir para exercício