# Aula 2 - Desenvolvimento de software

Jayme Anchante

10 de junho de 2020

# Desenvolvimento de software

# Conceito

Desenvolvimento de software é o processo de conceber, especificar, desenhar, programar, documentar, testar e consertar bugs envolvendo a criação e manutenção de aplicações, paradigmas ou outros componentes de software

# Objetivos

Atender a necessidade de cliente/negócio, atender uma necessidade futura tanto comercialmente como de forma *open source*, e para uso pessoal

# Engenharia de software

A necessidade de melhor controle de qualidade no desenvolvimento de software fez surgir a engenharia de software, ou seja, aplicar o paradigma da engenharia para o desenvolvimento de software

# Coders, programmer, software developer, software engineer

- Coder: escrever código (que pode não ser parte/um programa ou sistema de software), código pode ser marcação/markup (HTML, CSS, XML) ou funções em uma planilha
- Programmer: pessoa que programa, para desenvolver software muitas vezes não é necessário programar
- Desenvolvedor de software: precisa entender de todas as fases do desenvolvimento de software, não apenas a implementação
- Engenheiro de software: um desenvolvedor de software com um grau de educação formal e/ou muita experiência, escrita em muitas linguagens, habilidades em arquitetura, modularização, scaling,

Mais destas definições nessa **thread no Reddit**

# Atividades principais

- Processos/procedimentos
- Requisitos
- Design
- Engenharia
- Construção
- Testes
- Debugging
- Implantação/deployment
- Manutenção

# Processos

# Conceito

Processo de desenvolvimento de software (também conhecido como metodologia, modelo ou ciclo de vida) é o paradigma usado para estruturar, planejar e controlar o processo.

Existem diversos paradigmas com diferentes características, forças e fraquezas, e nenhuma é a melhor em todos os casos

# Etapas gerais

Também conhecido como software development life cycle (SDLC)

- Análise o problema
- Pesquisa de mercado
- Requisitos para solução de negócio
- Criação de um plano ou design para uma solução baseada em software
- Implementação/codar o software
- Testes
- Implantação
- Manutenção e correção de bugs

# Identificação da necessidade

Ideias para novos softwares podem vir de pesquisas de mercado, clientes atuais, prospects de venda.

Ideias devem ser avaliadas pela equipe de marketing para viabilidade econômica, adaptação/fit com os atuais canais de distribuição e produtos, características requeridas, após essa primeira triagem o projeto pode ir adiante

Poucas pessoas conhecem engenharia, marketing e finanças necessárias e suficientes para lançar um produto de software

Projeto de desenvolvimento de software pode acabar indo para outras direções menos técnicas como recursos humanos, gerenciamento de riscos, orçamentos e outros, fazendo com que o desenvolvimento de negócios tome conta

# Planejamento

Levantamento de requisitos: stakeholders têm uma ideia abstrata do que gostariam, mas não o que ou como o software deveria fazer

Engenheiros experientes reconhecem requisitos ambíguos, incompletos ou contraditórios

Um documento com análise de escopo deve ser redigido de forma clara

# Design

Design preliminar e alto nível dos principais módulos com o quadro geral de como cada parte se relaciona

Linguagem, sistema operacional, softwares auxiliares, hardware devem ser conhecidos

Um design mais detalhado deve ser criado, talvez com um protótipo ou com uma POC (*proof of concept*) para reafirmar os requisitos

# Implementação, documentação e testes

Implementação é quando ocorre a parte da escrita do programa

Testes garantem que os defeitos são reconhecidos rapidamente, em algumas práticas (test-driven development) os testes são desenvolvidos antes da implementação servindo como guia de boas práticas

Documentação tem a função de ajudar a futura manutenção e melhora do software e é feita ao longo do desenvolvimento.

# Implantação e manutenção

A implantação inicia assim que o código passa pelos testes e é aprovado para lançamento e vendido ou colocado em um ambiente de produção

Treinamento e suporte são muito importantes para seu uso correto

Manutenção e melhora para corrigir falhas ou novos requisitos, grande parte dos casos manutenção regular é necessária para garantir o funcionamento contínuo

# Requisitos

# Conceito

Análise dos requisitos é a tarefa que determina as necessidades ou condições de novos projetos ou alterações destes, levando em consideração os diversos stakeholders, outros softwares e sistemas

É um aspecto chave para o sucesso de um projeto. É um documento que deve mensurável, testável e bem explicado

# Etapas

1. Extração dos requisitos: documentação de processos de negócios, entrevistas com stakeholders
2. Documentação de requisitos: formato de lista, documentos por extenso, casos de uso, histórias do usuário, especificação de processos e outros modelos
3. Análise dos requisitos: determinar se os requisitos estão claros, completos, concisos, válidos, e resolvidos outros conflitos

# Algumas práticas

Desenvolvimento de cenários ou histórias de usuários em ágil

Etnografia (observação de participantes), entrevistas, grupos focais

Protótipo pode ilustrar melhor a definição para stakeholders (parte visual/telas feita pelo deisgner/UX)

# Sessões de Desenvolvimento Conjunto de Requisitos

Requisitos tem implicações que às vezes são desconhecidas para stakeholers individuais

Essas implicações podem ser descobertas por meio de sessões reunindo diferentes stakeholders em que um facilitador treinado pode provocar para que surjam

# Lista de requisitos

A maneira tradicional de documentar requisitos é por meio de listas, ela oferece uma maneira simples de *checks*, um contrato sponsor e os desenvolvedores; entretanto ela pode ser muito longa se forem muitos requisitos, podem ser abstratas, difícil

Em ágil, recomenda-se o uso de histórias do usuário

# Objetivos mensuráveis

Após ter a lista de requisitos, é interessante pergunta “por que?” para todos os itens até que o verdadeiro propósito seja descoberto

Ideação de maneiras de testar quando cada requisito for alcançado

# Protótipos

É um programa que exibe parte das propriedades de um outro programa de computador.

Uma forma popular é o *mockup* (maquete, modelo) para ajudar usuários a terem uma ideia de como o sistema será

Ainda podem ser *wireframes* ou aplicações funcionais com menos funções, em escala de cinza para prevenir confusão (protótipo vs produto final)

# Casos de uso

Cada uso de caso provê uma série de cenários de como o sistema irá interagir com humanos ou outros sistemas para atingir um certo objetivo de negócios, evitando jargão, e sendo na linguagem do usuário final ou especialista do domínio

# Tipos de requisitos

- Cliente
- Arquitetural
- Estrutural
- Funcional
- Performance
- Design

# Problemas

- Stakeholders: não entendem o que querem, não se comprometem com requisitos por escrito, insistem em novos requisitos após orçamento e cronograma estarem fixados, não participam de reviews, não conhecem o processo de desenvolvimento e as tecnologia
- Engenheiros/desenvolvedores: escrever código antes dos requisitos estarem claros, uso de diferentes vocabulários levando a erroneamente achar que todos estão concordando

# Design

# Conceito

Processo de criar uma especificação de um artefato de software para alcançar objetivos usando componentes primitivos e certas restrições

Etapa após os requisitos e antes da programação em si

Envolve tanto parte de baixo nível de componentes e algoritmos como alto nível de arquitetura

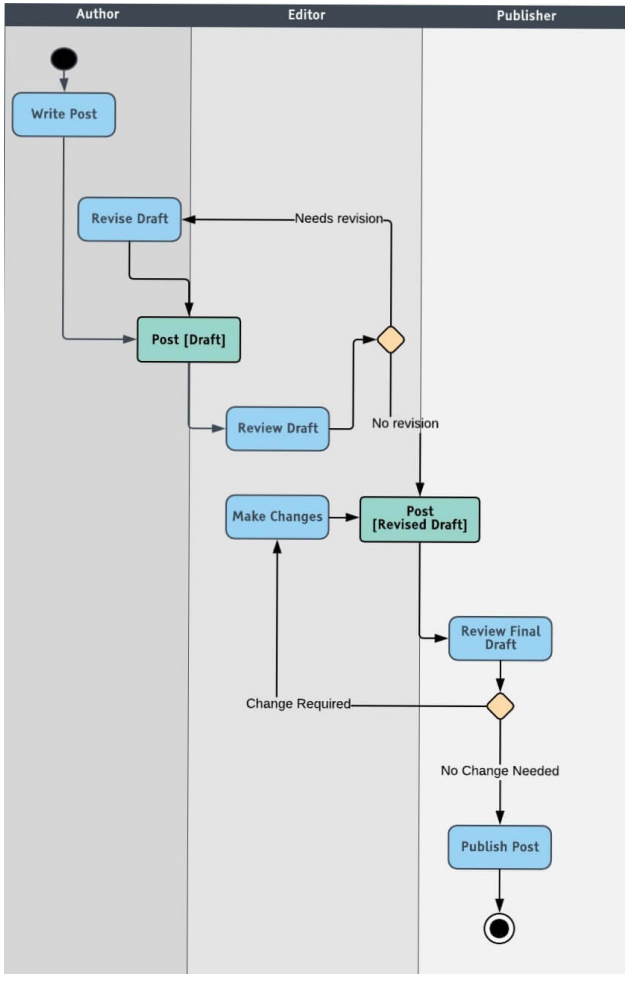
# Análise dos requisitos

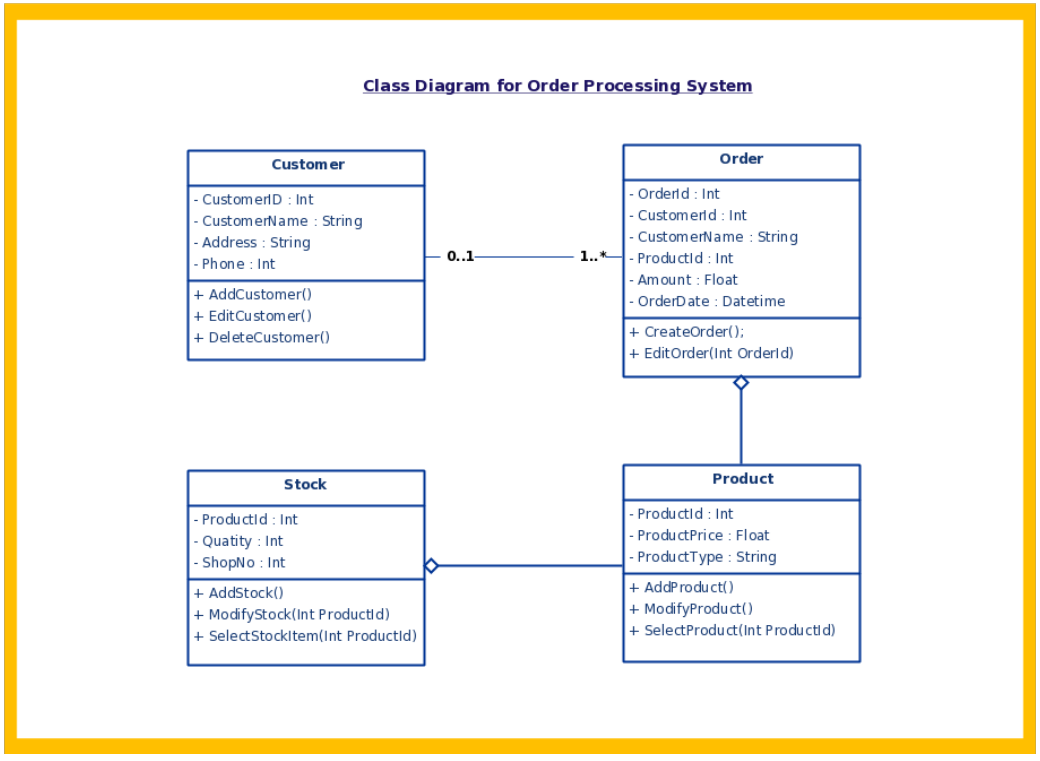
Se o software for semiautomatizado ou centrado no usuário, ele precisará de um design de experiência do usuário que resultará num *storyboard*

Se o software for completamente automatizado, pode um gráfico de fluxo ou um simples texto descrevendo as etapas

# Unified Modeling Language

É uma linguagem de modelagem de propósito geral que provê uma forma padrão de visualizar o design de sistemas



# Fundamental modeling concepts

Paradigma para descrever sistemas de software.

*Agentes* são componentes ativos do sistema (retângulos). *Armazéns* são componentes passivos (formas curvas). Agentes se comunicam via *canais* (linhas e setas).

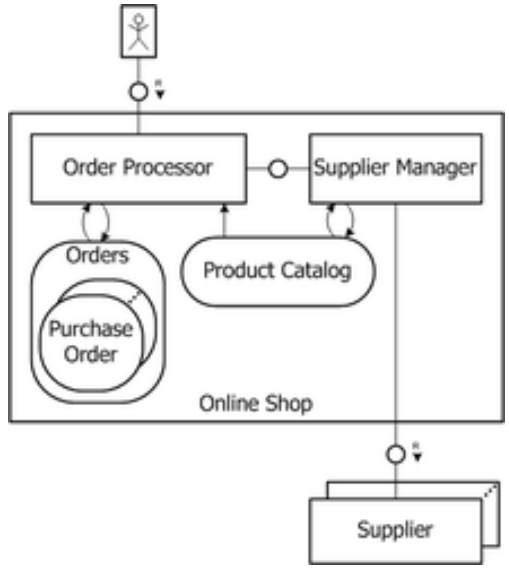



Figure 1: FMC

# Análise de software

Consiste em reduzir o software em problemas menores para resolver

A análise não deve diferir muito entre pessoas da mesma equipe ou entre grupos

Já design de software é mais subjetivo e vários designs para o mesmo problema podem coexistir

# Processo e modelagem

Design do processo é a sequencia de passos para descrever todos os aspectos da construção do software

A modelagem pode ser comparada ao plano do arquiteto para uma casa (começa com uma representação alto nível 3D, depois vai ganhando detalhes). O modelo oferece diversas visões do software

# Considerações

- Considerar abordagens alternativas
- Paralelo entre o design e os requisitos
- Não deve “reinventar a roda”
- Estar preparado para acomodar mudanças
- Design não é codar, e codar não é design
- A qualidade do deisgn deve ser julgado no momento da criação, não depois

# Conceitos

- Abstração
- Modularidade
- Arquitetura
- Hierarquia
- Estrutura de dados

# Construção

# Conceito

É a disciplina que reuni codificação, verificação, teste unitário, teste de integração e debugging

# Fundamentos

- Minimizar complexidade
- Antecipar mudanças
- Construção para verificação
- Código reutilizável
- Utilizar padrões (da linguagem, da equipe, da plataforma)

# Linguagens

- Configuração: linguagem com um conjunto finito de opções para criar ou configurar um instalação. E.g. arquivos em /etc do Unix
- Toolkit: usados para criar aplicações um pouco mais complexas. E.g. GTK+, Qt
- Scripting: linguagens interpretadas em vez de compiladas, tem um propósito específico. E.g. awk, sed
- Programação: mais flexível e completa

# Técnicas de codificação

- Código fonte legível, utilizando nome de variáveis significativos (podem ser longos)
- Usar classes, tipos, variáveis, constantes e outras entidades
- Controle de fluxo: não utilizar muitos níveis em laços para melhorar a leitura e a propensão ao erro
- Lidar com erros e exceções
- Uso de recurso de forma eficiente, usando threads, locks em banco de dados
- Organização em classes, pacotes, bibliotecas
- Documentação no código

# Testes de construção

- Teste unitário: testa uma seção da aplicação (função, método, classe)
- Teste de integração: as várias unidades são combinadas e testadas conjuntamente

# Testes

# Conceito

Investigação para prover os stakeholders de informação sobre a qualidade do software

Visão objetiva e independente

Verificar se atende os requisitos e o design de construção, responde corretamente frente a inputs, performa dentro do tempo esperado, pode ser instalado e rodada nos ambientes e sistemas

# Limitações

Determinam o correto funcionamento sob certas hipóteses, não consegue identificar todos os defeitos

As condições podem ser as especificações dos requisitos, contratos, comparação com outros produtos, versões passadas do mesmo produto, inferência sobre o comportamento esperado, expectativas do usuário

# Cargo

Inicialmente o tester era uma habilidade dos programadores, hoje é muitas vezes um cargo a parte

Algumas nomenclaturas são: tester, analista de testes, designer de testes, desenvolvedor de automação, quality assurance (QA)

# Tipos

- Estáticos: code reviews, passo a passo (quem escreveu o código explica seu funcionamento ao tester), inspecionamento
- Dinâmico: execução de software para testes de caso

# Caixa transparente

Verifica as estruturas internas do programa, also inacessível para o usuário final

Tester escolhe certos inputs que irão fluir para um certo caminho e serão determinados e comparados os outputs

Teste de APIs, cobertura de código, resistência a falhas

# Caixa opaca

Examinam a funcionalidade sem saber nada do código fonte

Muitas vezes os testes de caso são necessários

Não necessita de conhecimentos de programação, como testes visuais

# Testes Alpha, Beta

- Alpha: uso por alguns clientes/usuários potenciais ou um time de testes independente, pessoas dentro da organização
- Beta: depois do alpha, uma audiência limitada de pessoas têm acesso a aplicação

# Testes automatizados

Oposto de testes manuais

Necessário em um ambiente com integração contínua: software passa por uma bateria de testes cada vez que uma mudança é feita num sistema de versionamento de código

# Debugging

# Conceito

Processo de encontrar e resolver problemas em um programa

# Origem do termo

A Almirante nos anos 1940 estava trabalhando num computador Mark II em Harvard encontrou uma traça presa no computador impedindo seu funcionamento

Debugging significa remover insetos

Figure 2: Grace Hopper

# Processo

Reproduzir o bug, às vezes não é trivial. Programa rodando em paralelo, programas específicos do ambiente de origem, histórico de uso

Simplificação dos inputs

Usar um debugger que examina os estados (variáveis, chamadas) ou um rastreamento simples (usar “prints”)

# Implantação/deployment

# Conceito

Processo de fazer com que o software se torne disponível para uso

# Atividades

Lançamento/release segue um processo de desenvolvimento completo

(Des)Instalação e (in)ativação envolve estabelecer um comando ou atalho para (remover) executar o software

Atualização e atualização embutida

Rastreamento de versão ajuda o usuário a encontrar e instalar as aplicações

# Papéis

Envolve diversas pessoas do time, sendo liderados pela figura do DevOps, também o engenheiro de software responsável/líder, gestor, administração de sistemas e o administrador de banco de dados

# Manutenção

# Conceito

Modificações que ocorrem após a entrega do software para corrigir falhas ou melhorar a performance ou outras características

# Importância

Garantia para o consumidor que o software será revisado

Software open source é de código aberto mas pode não ter uma empresa responsável (legalmente) - é da comunidade. Empresas querem garantias, SLAs (service level agreement)