Introdução à modelagem de software (parte 2)

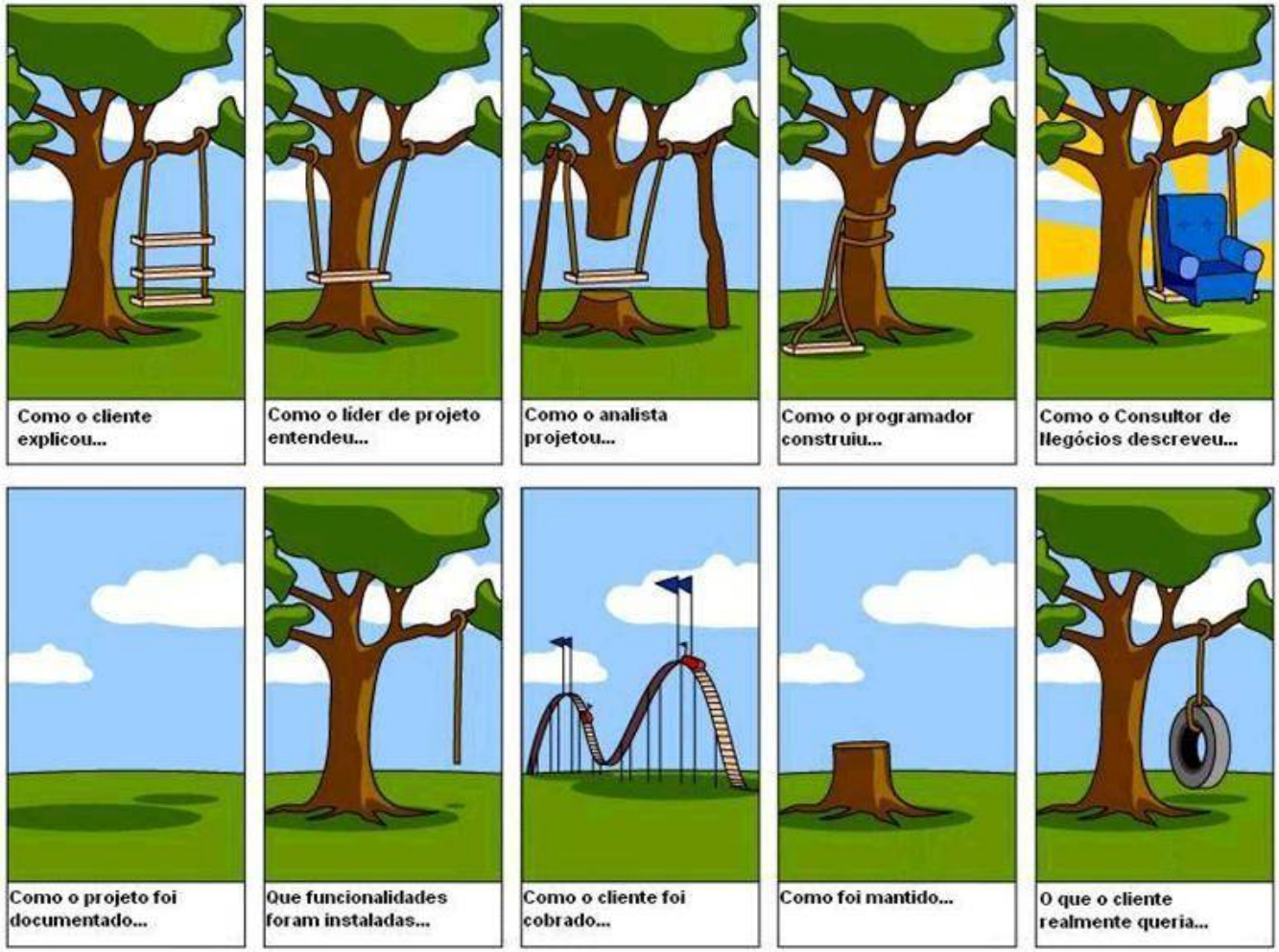
Prof. Murillo G. Carneiro
FACOM/UFU

Material baseado nos slides disponibilizados pelo Prof. Ricardo Pereira e Silva (UFSC)

Objetivos

- Compreender o paradoxo da indústria de software
- Introduzir etapas básicas do ciclo de vida do software
- Apresentar requisitos para especificação e abstração de software

Modelagem de Software



Prática primitiva de desenvolvimento

- Cultura se resume à capacitação para usar linguagem de programação
- A partir de uma necessidade de desenvolvimento, parte para a produção de código
- Sem noção clara e completa dos requisitos
- Resultado: código frequentemente mal estruturado, que passou por sucessivos remendos e praticamente nenhum outro produto
- Pouca preocupação com manutenibilidade

Complexidade no desenvolvimento

- Primeiro aspecto destacado na definição de Engenharia de Software (*Webopedia*)
- Mais relevante que o aumento do esforço do desenvolvimento ou do tamanho do produto
- Dificuldade de administrar complexidade
  - Pode levar ao fracasso do desenvolvimento

Complexidade – exemplo

- Localizar “**6io**”: onde é mais fácil?

Vjrektofawefuiher4tye8793o6ke9v7c2435k4g906iosfdgjh4tjsaifhs948utisrfs

Jkfnvuit7jfn547t67sdj24389tfdnv7836t53urd89rv53tfdjkabsmcks035iree54
9yencs8743th53u734fusnf5696f6b5463786vcsrnt5396wr7efui35t89538675
e89vyruehth59u6987gshbcreth537r75496hi9eg89re754jiosh9as839q39439
ok89vgyte7t049j6o46yj0evys784i543of98sr89f95th67594y8tg89dg6p4jy05
79y46897g8sd8f54ojty56oy87489g749th54y5698g7e85uth4698567g8tg4o
6857y9rtsvj5t234753t4tijgdf53857i46j74v8uet958u646oj3t9eug8948tu466j
o439tu4g789et8u46tjyo4gfugf48t746jyopekfzxkvmxcnvbmxfcvbh734t5j45h
tfd8v89e5i6j54otjreu89fd7g5io654ojtfd7v98r7d08t9456ioj4p3495t90r8vd8
g00t49634opjt9043u9fd890vbfdoignkngiou89789uvifvdjg878gre8985906u
54oini0f4u8fe8g9085406540jt0eg89fd7g985486u40j3klfgjfd8g789e48u654
imfpwrjke89rf7985364eijfiosduf379576943jjgrfdj9f79547t84u6io4j3gkfsjdv
urs89f73975897ok56jgfeya47857892536894 3hg89f798g75tu0958g854725

Complexidade e modelagem

- Modelagem é um instrumento de administração de complexidade
  - Permite tratar software mais complexo sem perder o controle do conjunto de informações nele contido
  - De mais fácil compreensão e permite uma mais rápida localização de informações, em comparação com código (linguagem de programação)

Aumento contínuo da complexidade

- Aumento ao longo das últimas décadas
- Complexidade do software é limitada pela capacidade do hardware
  - Computadores mais poderosos vêm se tornando economicamente acessíveis
  - Hardware mais poderoso → aumento da demanda por software mais complexo

Paradoxo da indústria do software

É preciso buscar a capacidade de desenvolver software continuamente mais complexo e o desenvolvimento deve ocorrer em um espaço de tempo cada vez menor, bem como demandando menos esforço

Caminhos para sobreviver ao paradoxo

- Abordagens metódicas de desenvolvimento
  - Evitar o desenvolvimento baseado em tentativa e erro
- Automação
  - Desenvolvimento e uso de ferramentas que automatizem atividades
  - Permitir que elas sejam realizadas mais rapidamente e demandando menos esforço humano
- Reuso de software
  - Reaproveitar ideias, soluções de projeto ou trechos de código previamente elaborados
  - Não ser necessário desenvolver todas as partes de um software

Ciclo de vida do software

- Etapas por que um software passa ao longo de sua existência
- Divisão de etapas adotado:

análise, projeto, implementação, testes, manutenção

Ciclo de vida do software

análise, projeto, implementação, testes, manutenção

- Análise + projeto = planejamento do que construir
- Implementação + testes = construção
- Manutenção → alteração posterior ao desenvolvimento

Análise

- Convenção adotada: o objetivo da análise é a modelagem do domínio do problema
- Domínio do problema → conjunto de fatos e conceitos associados ao problema que se deseja tratar
  - Exemplo: para um jogo-da-velha, o domínio do problema corresponde ao conjunto de informações associado a esse assunto, como os elementos do jogo, as regras, estratégias
- O tratamento do domínio do problema deve ignorar tanto quanto possível a existência do mundo computacional

Projeto

- Convenção adotada: o objetivo do projeto é a modelagem do domínio da solução computacional
- Selecionar as alternativas que constituirão a solução computacional
  - Interface, persistência, segurança etc.

Análise e projeto

ANÁLISE do problema

PROJETO da solução

Resultado dos esforços de análise e projeto

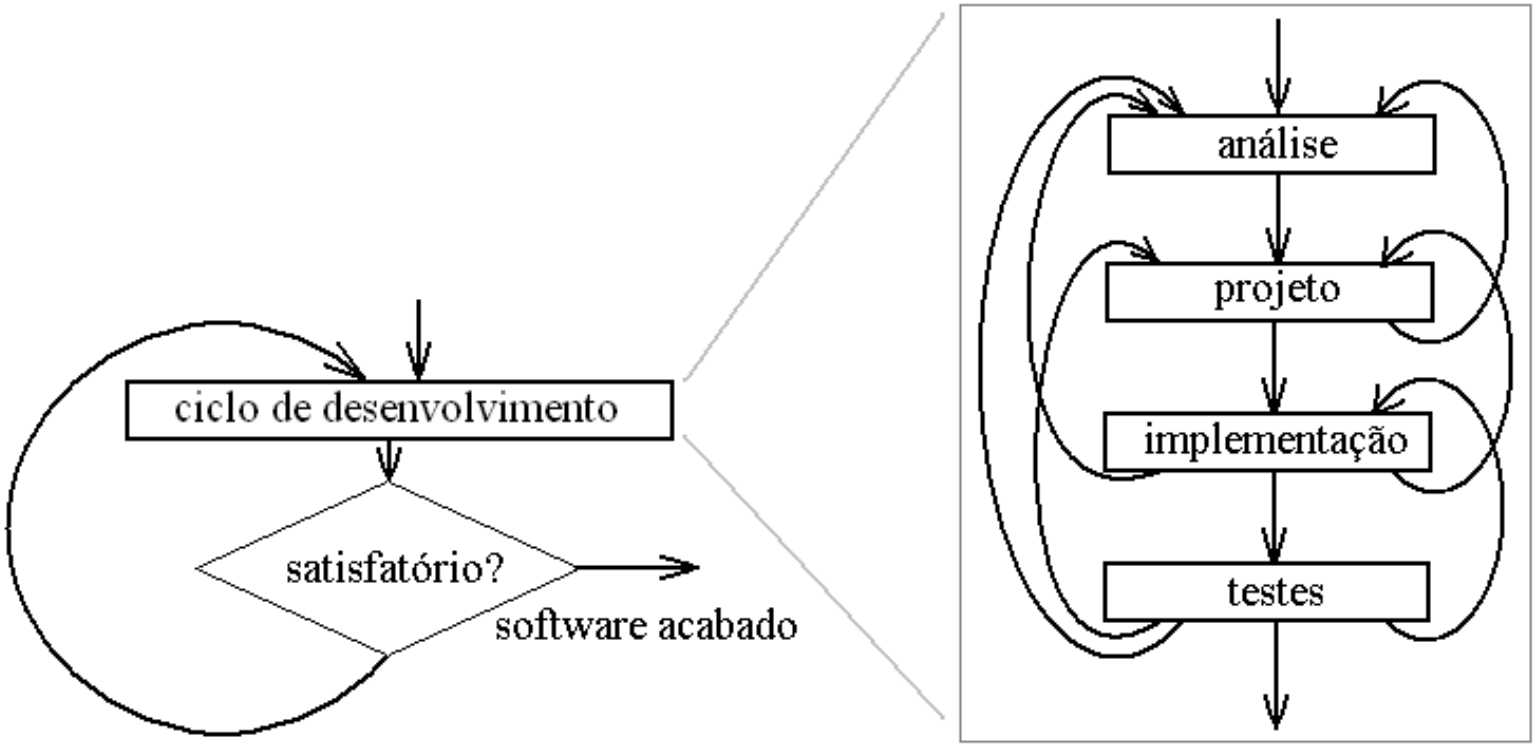
- Possibilidade 1: duas especificações
  - uma resultante do esforço de análise
  - uma segunda, resultante do esforço de projeto
- Possibilidade 2: uma única especificação de projeto
  - extensão da especificação de análise
- A segunda alternativa é adotada neste curso

Processo de desenvolvimento de software

- Conjunto de soluções adotado por uma pessoa ou grupo para desenvolver software
- Composto por
  - Um conjunto de etapas
  - A sequência das etapas
  - As atividades associadas a cada etapa
  - As metodologias, modelos, padrões e ferramentas utilizados em cada atividade

Modelo de ciclo de vida incremental

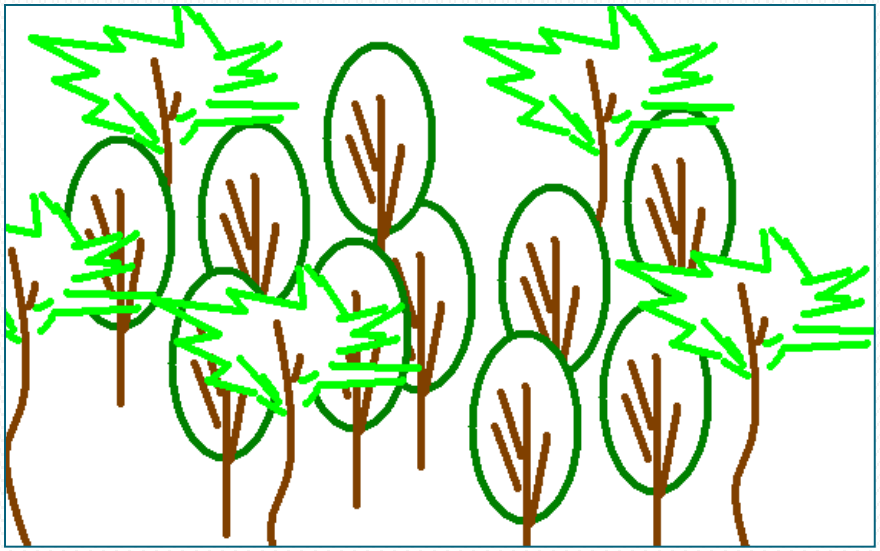
- Estabelece uma possível sequência de etapas
- Percurso cíclico

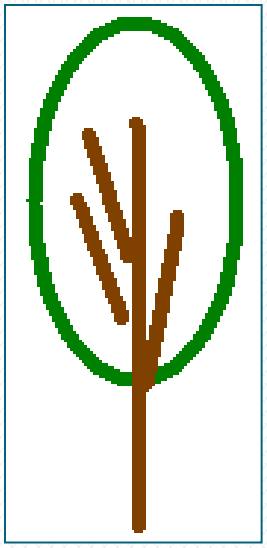


Requisitos para especificação de software

- Necessidade de registro das decisões de projeto
  - Mecanismo de registro: linguagem
- Possibilidade de tratar a globalidade ou o detalhe
  - Linguagem deve suportar diferentes níveis de abstração

Níveis de abstração





Níveis de abstração

- Floresta: visão da totalidade
  - Abstraindo os detalhes
  - O mais alto nível de abstração
- Árvore: nível de abstração imediatamente inferior
  - Menos detalhes abstraídos
- Folha: o mais baixo nível de abstração
  - Foco no detalhe
  - Para evitar descrição densa, sem visão da globalidade

Níveis de abstração

- No registro do projeto de software: necessidade de diferentes níveis de abstração
- Código (linguagem de programação): apenas nível de abstração mais baixo
  - Inadequado para registro de projeto

Uso de especificações de projeto

- Durante o desenvolvimento
- Para compreender software existente

Uso de especificações durante o desenvolvimento

- Auxilia a identificação e seleção de alternativas mais adequadas
  - Possibilidade de tratar o software em diferentes níveis de abstração
  - Sob diferentes pontos de vista
- Permite chegar a um resultado de melhor qualidade, isto é, melhor estruturado

Uso de especificações para compreender software existente

- Caso típico da etapa de manutenção
  - o primeiro passo é compreender o que está implementado
  - menos trabalhoso compreender uma especificação de projeto do que compreender código
- É consenso o reconhecimento do benefício de dispor de especificação de projeto para compreender um software
  - O obstáculo costuma ser convencer quem não tem o hábito a produzir especificação de projeto

Considerações sobre esta aula

- Discutimos a complexidade do desenvolvimento de software e o paradoxo da indústria de software
- Introduzimos ciclo de vida e processo de desenvolvimento de software, com especial atenção para Análise e Projeto
- Introduzimos conceitos de requisitos de especificação de software e níveis de abstração

Referências

Booch, G.; Jacobson, I. e Rumbauch, J. **UML: Guia do Usuário**. Campus, 2006.

Silva, R. P. **UML 2 em modelagem orientada a objetos**. Visual Books, 2007.