**Aluno: José Mauricio Barbisan Zottis**

**Atualização em 27/05/2016**

# Atividade 01:

Descrever e /ou modelar um processo de desenvolvimento de software, identificando no mesmo

a configuração dos artefatos envolvidos e o gerenciamento de liberação e entrega.

**Um processo de desenvolvimento de software possui 4 etapas básicas:**

* **Especificação** definição das funcionalidades do software e premissas para sua execução.
* **Projeto** construção do software de acordo com a especificação
* **Validação** validar o software para verificar se ele atende as necessidades dos usuários
* **Evolução** evolução do software de modo a atender as modificações das necessidades dos usuários.

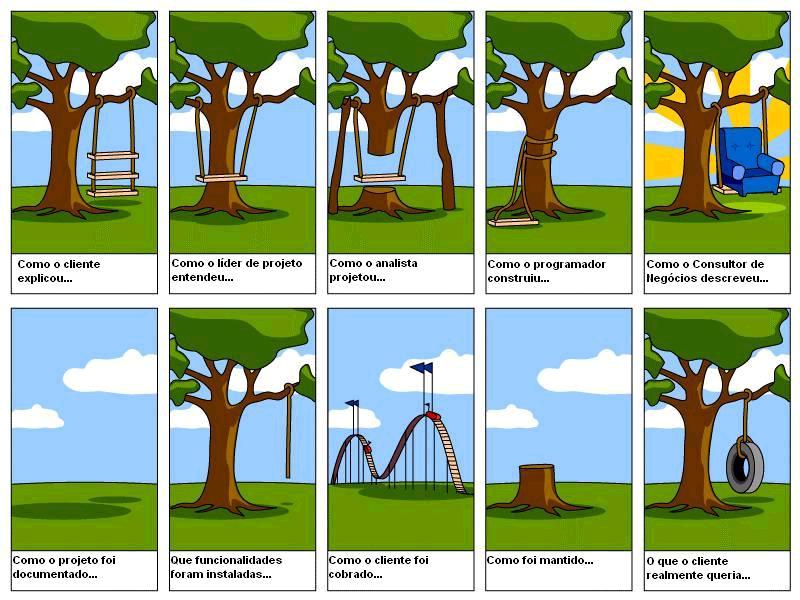
## A Fase de Especificação

### Criar relatório inicial de investigação (para construir os casos de uso)

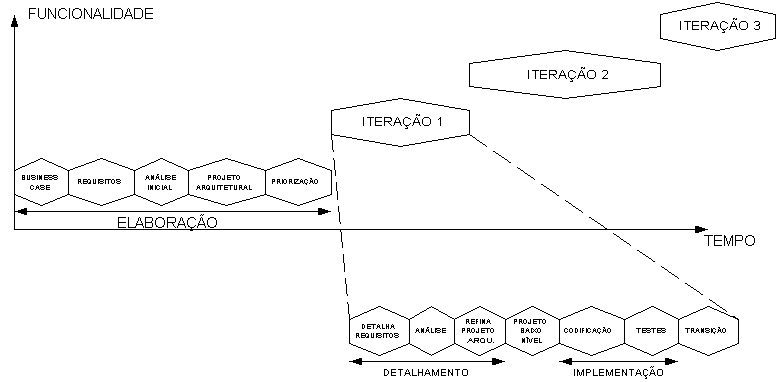
1. Levantar requisitos funcionais e não funcionais
2. Construir glossário (ao longo da fase)
3. Definir modelo conceitual inicial (análise inicial)
4. Projetar arquitetura
5. Priorizar a funcionalidade e distribuí-la entre as iterações

## Detalhes sobre a fase de Construção

* Hoje, é considerado errado ter um processo que gere um "big bang!"
  + Não se deve ter o software inteiro funcionando por inteiro no primeiro release
  + O risco é grande demais!
* Um processo de desenvolvimento deve ser:
  + Iterativo (ter várias iterações no tempo)
  + Incremental (gerar novas versões incrementadas a cada release)
  + Uma iteração dura entre 2 semanas e 2 meses
* Motivos:
  + Sempre tem algo para entregar para o cliente apressado (a última iteração)
  + Os requisitos mudam com tempo e um processo iterativo mantém freqüentes contatos com o cliente o que ajuda a manter os requisitos sincronizados
  + Altamente motivador para a equipe de desenvolvimento (e o cliente) ver o software funcionando cedo
  + Para evitar isso:



* O que é feito a cada iteração?
  + Análise (refinamento de requisitos, refinamento do modelo conceitual)
  + Projeto (refinamento do projeto arquitetural, projeto de baixo nível)
  + Implementação (codificação e testes)
  + Transição para produto (documentação, instalação, ...)



## Detalhes sobre a análise

* A análise gera um modelo para entender o domínio do problema
* Análise também trata em alto nível de como uma solução possível pode ser montada para atender aos requisitos
  + Acaba gerando uma especificação, mas sempre do ponto de vista do usuário e tratando apenas do domínio do problema
* Não trata de detalhes de implementação
* Objetos tratados são sempre do domínio do problema (business objects)
* Muitos diagramas UML podem ser usados
  + O modelo é para o cliente e não para o programador
* Atividades típicas durante a análise

1. Refinar use cases
2. Refinar modelo conceitual
3. Refinar glossário
4. Definir diagramas de seqüência (opcional)
5. Definir contratos de operação (opcional)
6. Definir diagramas de estado (opcional)

## Detalhes sobre o projeto (design)

* O projeto é uma extensão do modelo de análise visando sua implementação num computador
* Novos objetos aparecem, mas não são do domínio do problema
* O resultado é para o programador ver, não o cliente
* Objetos da análise são (geralmente) mantidos e são embutidos numa infra-estrutura técnica
  + As classes técnicas ajudam os business objects a:
    - Serem persistentes
    - Se comunicarem
    - Se apresentarem na interface do usuário
    - Terem desempenho aceitável (usando caches ou threads, por exemplo)
* As atividades de projeto incluem:
  + Fase de refinamento da arquitetura (high-level design)
    - Definição de pacotes (módulos), interfaces entre pacotes
    - Decisão sobre uso/criação de bibliotecas e/ou componentes
    - Falaremos disso em detalhes [adiante](http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/arqu/arquitetura.htm)
  + Fase de projeto detalhado (low-level design)
    - Atribuição de responsabilidades entre os objetos
    - Construção de diagramas de classes
      * Pode incluir documentação javadoc (ideal)
    - Construção de diagramas de interação (opcional)
    - Levantamento de necessidades de concorrência
    - Considerações de tratamento de falhas
    - Detalhamento do formato de saída (interface com usuário, relatórios, transações enviadas para outros sistemas, ...)
    - Definição do esquema do BD
    - Mapeamento de objetos para tabelas se o BD for relacional
  + Advertência: se você usar Test-Driven Development, então o design é feito bolando testes e escrevendo o software ao mesmo tempo
    - Neste caso, fazer diagramas ou Javadoc antes de codificar não funciona.

## Detalhes sobre a implementação

* Escrita do código
* Relativamente simples se o projeto tiver sido bem feito
* Programadores devem normalmente seguir regras de codificação da empresa
* Atividades incluem code reviews
* Não se deve chegar a esta fase cedo demais!
  + Mais cedo você agarra o teclado, mais vai demorar a terminar!
* Poucos novos diagramas nesta fase

## Detalhes sobre os testes

* Inclui várias fases de testes
* Testes feitos pelo próprio programador durante a programação
  + Unit test: teste de classes individuais (ou de grupos de classes relacionadas)
  + Functional test: teste de funções inteiras (item de menu, p. ex.)
  + Component test: teste de componentes inteiros (exe, dll, ...) sem (ou com pouco) scaffolding
* Testes feitos por equipes independentes de teste
  + System test: testa a integração entre todos os componentes do produto
  + Alpha test: teste de produto inteiro dentro de casa
  + Beta test: teste de produto inteiro fora de casa
* Testes devem ser automatizados

# Atividade 02:

Definir as ferramentas necessárias para o processo de desenvolvimento descrito, identificando como as mesmas serão utilizadas para o controle de modificações, versões e construções.

## Controle de Modificações

O controle de modificação ou mudança é uma das atividades necessárias da Gerência de Configuração. Seu papel principal é registrar, avaliar e rastrear todas as mudanças aplicadas ao projeto, desde o momento em que são propostas, até o momento em que são implementadas (ou não) nos itens de configuração.

Exemplo de ferramentas usadas: Trac, bugzilla, jira, scarab, mantis, etc.

## Controle de versões

Um sistema de controle de versão (como o próprio nome já diz) tem a finalidade de gerenciar diferentes versões de um documento. Com isso ele te oferece uma maneira muito mais inteligente e eficaz de organizar seu projeto, pois é possível acompanhar um histórico de desenvolvimento, desenvolver paralelamente e ainda te oferecer outras vantagens, como exemplo, customizar uma versão, incluir outros requisitos, finalidades especificas, layout e afins sem mexer no projeto principal ou resgatar o sistema em um ponto que estava estável, isso tudo sem mexer na versão principal.

Exemplos de ferramentas:

Subversion, git, Mercurial, etc.

**Funcionamento:**

Basicamente, os arquivos do projeto ficam armazenados em um repositório (um servidor em outras palavras) e o histórico de suas versões é salvo nele. Os desenvolvedores podem acessar e resgatar a ultima versão disponível e fazer uma cópia local, na qual poderão trabalhar em cima dela e continuar o processo de desenvolvimento. A cada alteração feita, é possível enviar novamente ao servidor e atualizar a sua versão a partir outras feitas pelos demais desenvolvedores.



## Controle de Construção

È um conjunto de ferramentas que automatizam a construção do software assegurando que tenha sido montado o conjunto apropriado de componentes validados.

automatiza o processo de transformação dos diversos artefatos do software que compõem um projeto em um sistema executável. Este processo é nomeado construção do software que, por exemplo, testa e empacota uma aplicação Java como um arquivo jar. Este processo ocorre de forma aderente às normas, procedimentos, políticas e padrões definidos para o projeto. Exemplos de ferramentas: Maven, Ant, Gradle, Hudson, Jenkins.

# Atividade 03:

Escolher e defender um modelo de controle de versão para o processo de desenvolvimento descrito.

R. Para essa finalidade, eu uso o SVN(Tortoise) pois consigo um melhor controle de usuários que podem apenas baixar(update) ou atualizar(commitar) os fontes.

# Atividade 04:

**O que são check-out e check-in?**

**Check-out:** é a opção de de “Baixar” ou obter a última versão modificada de um artefato ou arquivo de um projeto no servidor de controle de versão.

**Check-in:** é quando você transfere uma atualização efetuada para o servidor de controle de versão.

**Qual a diferença entre as políticas otimistas e pessimista?**

**Pessimista:** é quando o uso de check-out reservado é destacado, fazendo o bloqueio e não permitindo o desenvolvimento paralelo de um mesmo artefato.

**Otimista:** é quando temos um cenário onde a quantidade de conflitos é naturalmente baixa e será fácil de se tartar cada conflito individualmente quando ocorrer.

**O que é mainline?**

Por definição,é a primeira linha de desenvolvimento definida de um projeto

**O que são branches?**

São implementações de novas funcionalidades desenvolvidads por uma equipe em paralelo totalmente independente e isolada das modificações de outros desenvolvedores, é como se fosse um rio tivesse um ponto onde ele se dividiria em dois ou mais para na frente se juntar novamente.

# Atividade 05:

# Atividade 06: