FUCAPI

Pós-graduação Lato Sensu em Sistemas de Informação e Aplicações WEB

Módulo de Engenharia de Software

Prof. Tiago Eugenio de Melo, M.Sc.

E-mail: tiago@comunidadesol.org

Data: abril/2008

http://www.tiagodemelo.info

Roteiro

- Introdução à Engenharia de Software
- Processos de desenvolvimento de software
- Projeto de interface homem-máquina
- Verificação e validação de software
- Engenharia reversa e re-engenharia
- *eXtreme Programming*

Engenharia de Software

- A economia de todos os países do mundo dependem do uso de software.
- Cada vez mais o controle dos processos tem sido feito por software.
- A Engenharia de Software consiste no conjunto de teorias, métodos e ferramentas para o desenvolvimento profissional de software.

Engenharia de Software

- Atualmente, os custos de software superam os custos de hardware.
- A manutenção de software é onde se tem os maiores gastos. Principalmente nos sistemas de vida longa.

Questões preliminares

- O que é sistema?
  - É um conjunto de elementos concretos ou abstratos entre os quais se pode encontrar alguma relação.

Questões preliminares

- Quais são os elementos de um sistema?
  - São os elementos que estão dentro da fronteira conceitual.
  - São os elementos que possuem interações fortes entre si.
  - São os elementos que possuem interações fracas com os elementos externos ao sistema.

Questões preliminares

- O que é fronteira conceitual?
  - É o limite que separa o que está dentro do sistema do resto.
    - Os elementos de dentro do sistema devem ser detalhados.
    - Deve-se verificar a interação destes elementos com o ambiente externo.

Questões preliminares

- O que é software?
  - Programas de computador e documentação associada.
  - Existem duas categorias de produtos de software:
    - Produtos genéricos: sistemas produzidos e vendidos no mercado a qualquer pessoa que possa comprá-los.
    - Produtos específicos: sistemas encomendados especialmente por um determinado cliente.

Questões preliminares

- O que é Engenharia de Software?
  - É uma das áreas da Engenharia que trata dos aspectos de produção de software.
  - A Engenharia de Software tem como objetivo estabelecer uma sistemática abordagem de desenvolvimento, através de ferramentas e técnicas apropriadas, dependendo do problema a ser abordado, considerando as restrições e recursos disponíveis.

Questões preliminares

- O que é Engenharia de Software?
  - Em resumo, visa resolver problemas inerentes ao processo e ao produto de software.

Questões preliminares

- Quais são os princípios da Engenharia de Software?
  - Formalidade: produtos mais confiáveis, podem controlar seu custo e ter mais confiança no seu desempenho.
  - Abstração: identificar os aspectos importantes ignorando os detalhes.
  - Decomposição: subdividir o processo em atividades específicas, atribuídas a diferentes especialistas.

Questões preliminares

- Quais são os princípios da Engenharia de Software?
  - Generalização: sendo mais geral, é possível que solução possa ser reutilizada.
  - Flexibilização: modificação com qualidade.

Questões preliminares

- Qual é a diferença entre Engenharia de Software e Ciência da Computação?
  - A Ciência da Computação tem como objetivo o desenvolvimento de teorias e fundamentações; enquanto a Engenharia de Software se preocupa com as práticas de desenvolvimento de software.

Questões preliminares

- Qual é a diferença entre Engenharia de Software e Ciência da Computação?
  - Na prática, as ‘elegantes’ teorias da Ciência da Computação não podem ser aplicadas para os problemas reais e complexos que requerem uma solução de software.

Questões preliminares

- Qual é a diferença entre Engenharia de Sistemas e Engenharia de Software?
  - A Engenharia de Sistemas trata dos sistemas baseados em computadores, que inclui hardware e software. Enquanto a Engenharia de Software trata apenas dos aspectos de desenvolvimento de software.

Questões preliminares

- O que é um processo de software?
  - É um conjunto de atividades que objetivam o desenvolvimento e evolução de software.
  - Ele começa na concepção do problema (solicitação do usuário) e termina quando o sistema sai de uso.

Questões preliminares

- Quais são as principais atividades de um processo de desenvolvimento de software?
  - Especificação: define o que o sistema deverá fazer e as suas restrições.
  - Desenvolvimento: produção do software.

Questões preliminares

- Quais são as principais atividades de um processo de desenvolvimento de software?
  - Validação: checagem se o software faz o que o usuário quer.
  - Evolução: mudanças no software para atender às novas demandas.

Questões preliminares

- O que é um modelo de processo de software?
  - É a representação simplificada de um processo de software, apresentada sob uma perspectiva específica.
  - Exemplos de perspectivas:
    - Fluxo de trabalho (workflow): seqüência de atividades.
    - Fluxo de dados: fluxo das informações.
    - Papel/Ação: quem faz o que.

Questões preliminares

- Quais são os principais modelos?
  - Cascata ou sequencial.
  - Modelo evolutivo.
  - Transformação formal.
  - Integração de componentes reusáveis.

Questões preliminares

- Quais são os objetivos dos modelos?
  - Auxiliar no processo de produção, ou seja, gerar produtos com qualidade, produzidos mais rapidamente e a um custo cada vez menor.

Questões preliminares

- Quais são os objetivos dos modelos?
  - Possibilitam:
    - Ao gerente: controlar o processo de desenvolvimento de sistemas de software.
    - Ao desenvolvedor: obter a base para produzir, de maneira eficiente, softwares que satisfaçam os requisitos pré-estabelecidos.

Questões preliminares

- Quais são os custos da Engenharia de Software?
  - Pesquisas mostram que 60% dos custos é para o desenvolvimento e 40% para os testes.
  - O custo de evolução do software, normalmente, excede o custo de desenvolvimento.
  - O custo depende do tipo de sistema a ser desenvolvido e as suas restrições.
  - A distribuição dos custos depende do modelo de desenvolvimento adotado.

Questões preliminares

- O que são os métodos de Engenharia de Software?
  - São as abordagens estruturadas para o desenvolvimento de software que incluem os modelos de software, notações, regras e maneiras de desenvolvimento.

Questões preliminares

- O que são ferramentas CASE?
  - Softwares que têm o objetivo de fornecer um suporte automatizado para as atividades de processo de software.
  - Operam em dois níveis:

- Alto nível: ferramentas que suportam as atividades iniciais de requisitos e projetos.

- Baixo nível: ferramentas que suportam as atividades de programação, depuração e testes.

Questões preliminares

- Quais são os atributos de um bom software?
  - Atender às funcionalidades exigidas.
  - Eficiente.
  - Manutenível.
  - Usável.

Questões preliminares

- Quais são questões atuais?
  - Sistemas legados: os sistemas antigos devem ser mantidos e atualizados (atividades de migração).
  - Heterogeneidade: sistemas são uma combinação de hardware e software.
  - Prazos de entrega: pressão para um menor prazo de entrega.

Questões de Ética

- Confidencialidade
  - Os engenheiros devem respeitar a confidencialidade dos projetos dos seus empregadores e clientes.
  - Existência de contratos de confidencialidade.

Questões de Ética

- Surgimento de dilemas
  - Quando o analista não está de acordo com as políticas do engenheiro sênior.
  - Quando o empregador age de maneira não ética e as atualizações de sistemas críticos terminam sem a realização dos devidos testes.
  - Participação em projetos que contrariam os princípios do analista. Exemplo: desenvolvimento de projetos militares.

Questionário

- Qual é a importância da Engenharia de Software na sociedade atual?
- O que você entende por software?
- Comente sobre dois princípios da Engenharia de Software.
- Quais são as principais atividades de um processo de desenvolvimento de software?

Questionário

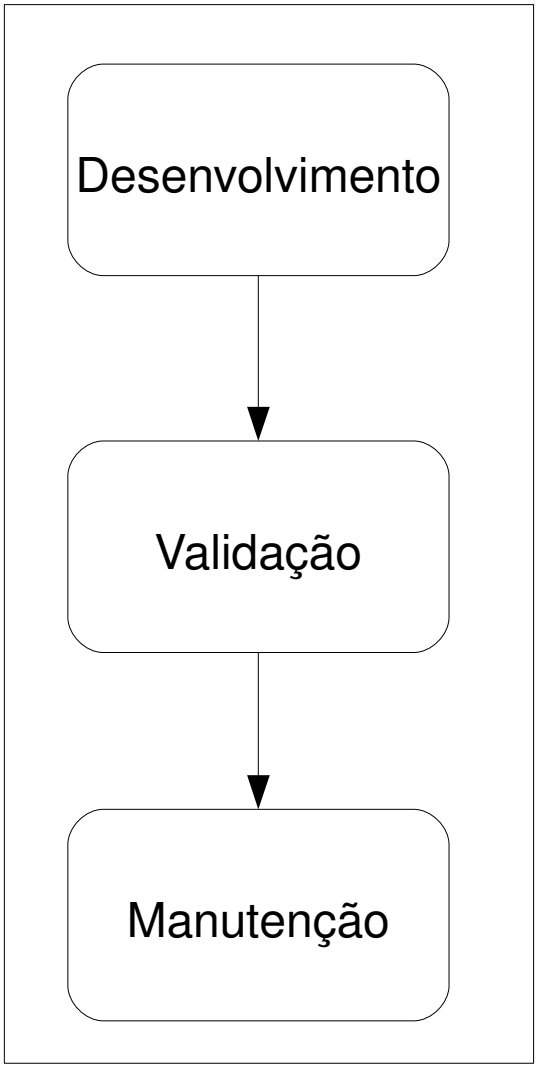
- Qual é a importância da etapa de manutenção no processo de desenvolvimento de software?
- Qual é o papel das ferramentas CASE no desenvolvimento de software?
- O que são sistemas legados? Qual é a dificuldade em trabalhar com sistemas legados ao se desenvolver um novo software?

Processos de desenvolvimento de software

- Definição:
  - Conjunto de atividades para especificar, projetar, implementar e testar sistemas de software.
- As atividades necessárias para o desenvolvimento de software são:
  - Especificação.
  - Projeto.
  - Validação.
  - Evolução.

Processos de desenvolvimento de software

- Etapas:



Processos de desenvolvimento de software

- Ciclo de Vida Clássico (Modelo Cascata)
  - Diferentes fases da especificação e desenvolvimento.
- Desenvolvimento Evolutivo
  - Especificação e desenvolvimento são alternados.
- Desenvolvimento Espiral
  - Faz uma combinação do ciclo de vida clássico e evolutivo.

Processos de desenvolvimento de software

- Desenvolvimento Formal
  - Uso de modelo matemático é formalmente transformado em uma implementação.
- Desenvolvimento Baseado em Reuso
  - O sistema é montado a partir de componentes.

Ciclo de vida clássico

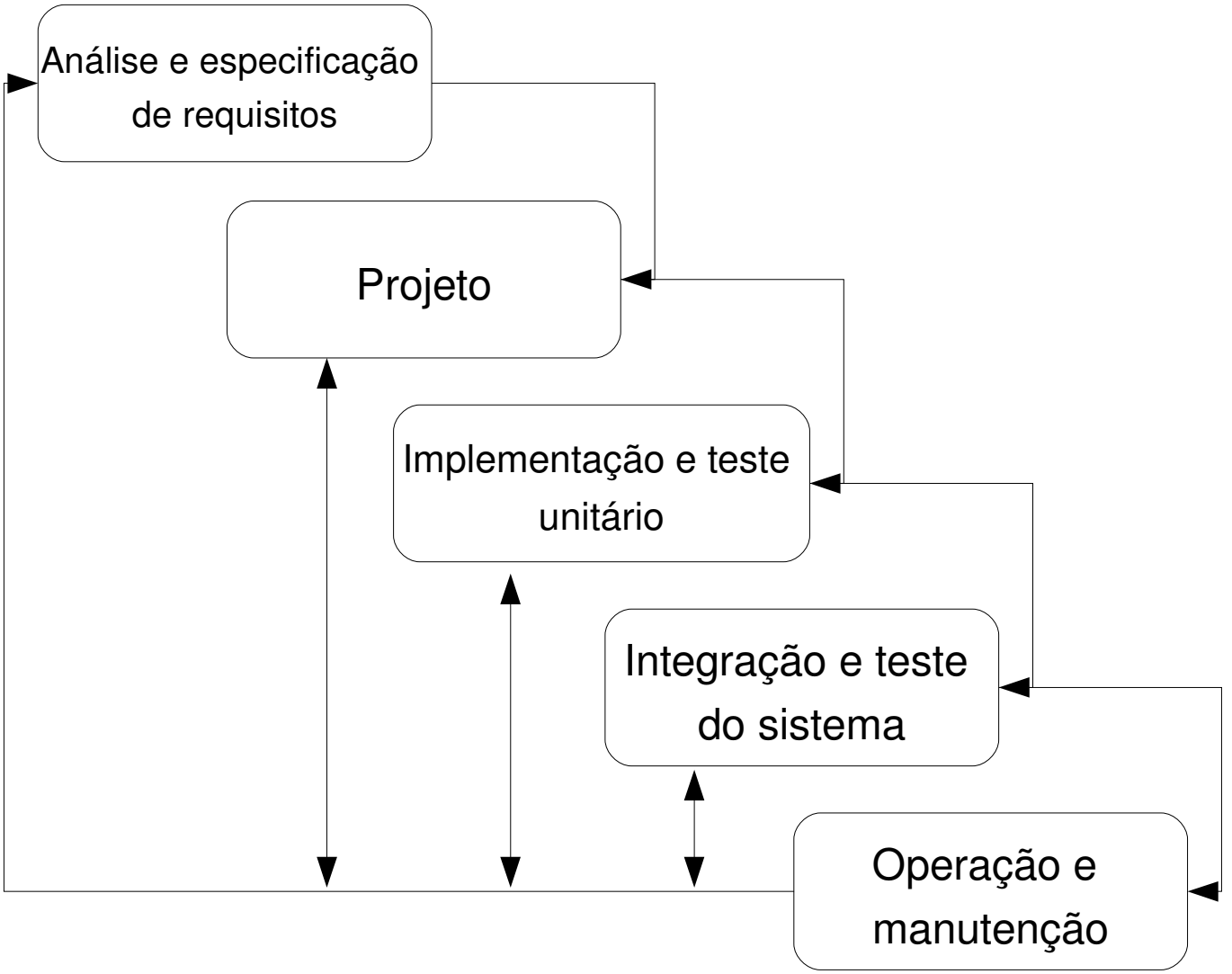
- Método é sistemático e seqüencial.
- O resultado de uma fase se constitui na entrada de outra.
- Também conhecido com cascata.
- Cada fase é estruturada como um conjunto de atividades que podem ser executadas por pessoas diferentes, simultaneamente.

Ciclo de vida clássico

- Fases:
  - Análise e especificação dos requisitos.
  - Projeto do software.
  - Implementação e teste unitário.
  - Integração e teste do sistema.
  - Operação e manutenção.

Ciclo de vida clássico

- Fases:



Ciclo de vida clássico

- Fases de análise e especificação de requisitos:
  - Durante essa fase são identificados, através de consultas aos usuários, os serviços e as metas a serem atingidas, assim como as restrições a serem respeitadas.

Ciclo de vida clássico

- Fases de análise e especificação de requisitos:
  - O resultado dessa fase consiste num documento de especificação de requisitos que tem como objetivos:
    - Ser analisado e confirmado pelo usuário para verificar se ele satisfaz todas as suas expectativas.
    - Ser usado pelos desenvolvedores de software para obter um produto que satisfaça os requisitos.

Ciclo de vida clássico

- O documento gerado
  - Deve ter as seguintes características:
    - Inteligível.
    - Preciso.
    - Completo.
    - Consistente.
    - Não ambíguo.
    - Facilmente modificável.

Ciclo de vida clássico

- Tipos de especificação
  - Funcionais: documenta o que o produto de software faz, usando notações informais, seminformais, formais ou uma combinação delas.
  - Não funcionais: documenta a confiabilidade, acurácia dos resultados, desempenho, problemas de interface, restrições físicas e operacionais, questões de portabilidade, entre outras.

Ciclo de vida clássico

- Tipos de especificação
  - De desenvolvimento e manutenção: documenta os procedimentos de controle de qualidade (teste, prioridades das funções desejadas, mudanças prováveis nos procedimentos de manutenção do sistema, entre outras).

Ciclo de vida clássico

- Fase de projeto
  - A fase de projeto envolve a representação das funções do sistema em uma forma que possa ser transformada em um ou mais programas executáveis.
  - É definida a solução do problema
    - Decomposição do produto sub-sistemas e/ou componentes.
    - Representação das funções do sistema em uma forma que possa ser transformada em programas.

Ciclo de vida clássico

- Fase de projeto
  - Definição de ***como*** o produto deve ser implementado.
  - Pode ser dividido em:
    - Projeto de alto nível (geral).
    - Projeto detalhado.
  - Tem como resultado um documento de especificação do projeto.

Ciclo de vida clássico

- Fase de implementação e teste unitário
  - Na fase de implementação o software é transformado em um programa, ou em unidades de programa, através de uma determinada linguagem de programação.
  - Teste unitário: cada unidade satisfaz suas especificações (planos e casos de teste pré-estabelecidos).
  - Resultado: coleção de programas implementados e testados.

Ciclo de vida clássico

- Fase de integração e teste de sistema
  - Programas ou unidades de programa são integrados e testados como um sistema.
  - Integração incremental: programas ou unidades são integrados à medida em que forem sendo desenvolvidos.
  - Resultado: produto pronto para ser entregue ao cliente.

Ciclo de vida clássico

- Contribuições do modelo
  - O processo de desenvolvimento de software deve ser sujeito à disciplina, planejamento e gerenciamento.
  - A implementação do produto deve ser adiada até que os objetivos tenham sido completamente entendidos.
  - Deve ser utilizado especialmente quando os requisitos estão bem claros no início do desenvolvimento.

Ciclo de vida clássico

- Problemas
  - Utiliza método sistemático e seqüêncial, em que a entrada de uma fase é o resultado da anterior.
  - O reinício do modelo é a dificuldade de acomodar mudanças depois que o processo está no final.
  - Dificuldade em atender às mudanças exigidas posteriormente pelo cliente.

Ciclo de vida clássico

- Problemas
  - Modelo mais adequado quando os requisitos estão muito bem entendidos.
  - Modelo ideal para aplicações maiores e mais complexas.
  - Rigidez, mas o desenvolvimento não é linear.
  - A meta continua sendo tentar a linearidade, para manter o processo previsível e fácil de monitorar.

Ciclo de vida clássico

- Problemas
  - Qualquer desvio é desencorajado, pois vai representar um desvio do plano original e, portanto, requerer um replanejamento.
  - Todo o planejamento é orientado para a entrega do produto de software em uma única data.
  - O processo de desenvolvimento pode ser longo e a aplicação pode ser entregue quando as necessidades do usuário já tiverem sido alteradas.

Desenvolvimento Evolutivo

- Baseado no desenvolvimento e implementação de um produto inicial, que é submetido aos comentários e críticas do usuário.
- O produto vai sendo refinado, através de múltiplas versões, até que o produto de software almejado tenha sido desenvolvido.
- As atividades de desenvolvimento e validação são desempenhadas paralelamente, com uma rápida resposta entre elas.

Desenvolvimento Evolutivo

- Categorias
  - Desenvolvimento Exploratório
    - O objetivo é desenvolver o sistema com o contínuo acompanhamento dos clientes desde da especificação até a entrega do produto. Os requisitos precisam ser bem entendidos.
  - Prototipação
    - O objetivo é entender os requisitos do sistema.

Desenvolvimento Evolutivo

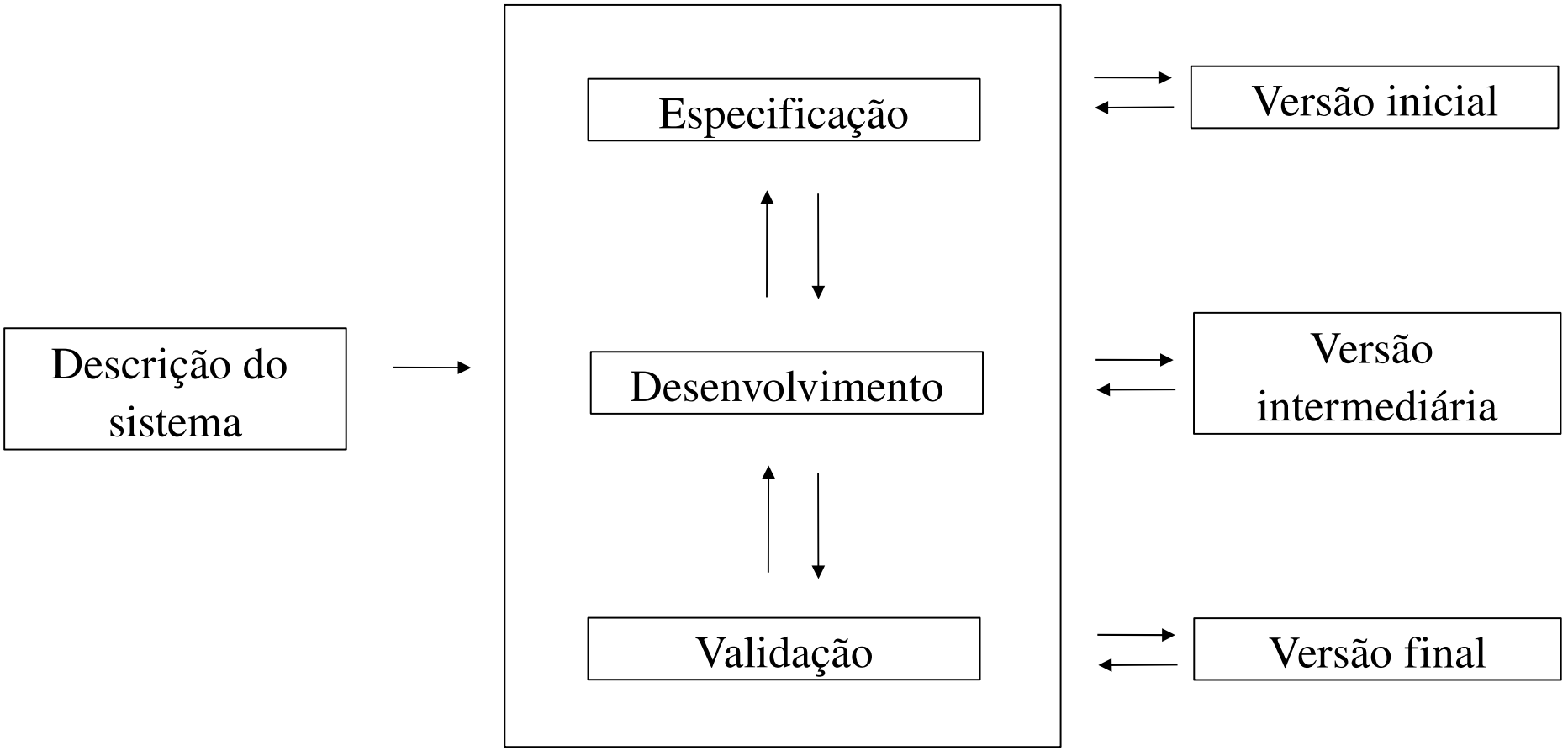
- Desenvolvimento Exploratório
  - O objetivo é trabalhar junto do usuário para descobrir os seus requisitos, de maneira incremental, até que o produto final seja obtido.
  - O desenvolvimento começa com as partes do produto que são melhor entendidas.
  - A evolução acontece quando novas características são adicionadas à medida que são sugeridas pelo usuário.

Desenvolvimento Evolutivo

- Desenvolvimento Exploratório
  - É indicado o seu uso quando é difícil, ou mesmo impossível, estabelecer uma especificação detalhada dos requisitos do sistema a *priori*.
  - A versão inicial do produto é submetida a uma avaliação inicial do usuário.
  - Essa versão é refinada, gerando várias versões, até que o produto almejado tenha sido desenvolvido.

Desenvolvimento Evolutivo

- Desenvolvimento Exploratório



Desenvolvimento Evolutivo

- Prototipação
  - Entender os requisitos do usuário e obter uma melhor definição dos requisitos do sistema.
  - Usado para fazer experimentos com os requisitos que não estão bem entendidos.
  - Enolve projeto, implementação e teste, mas não de maneira formal ou completa.

Desenvolvimento Evolutivo

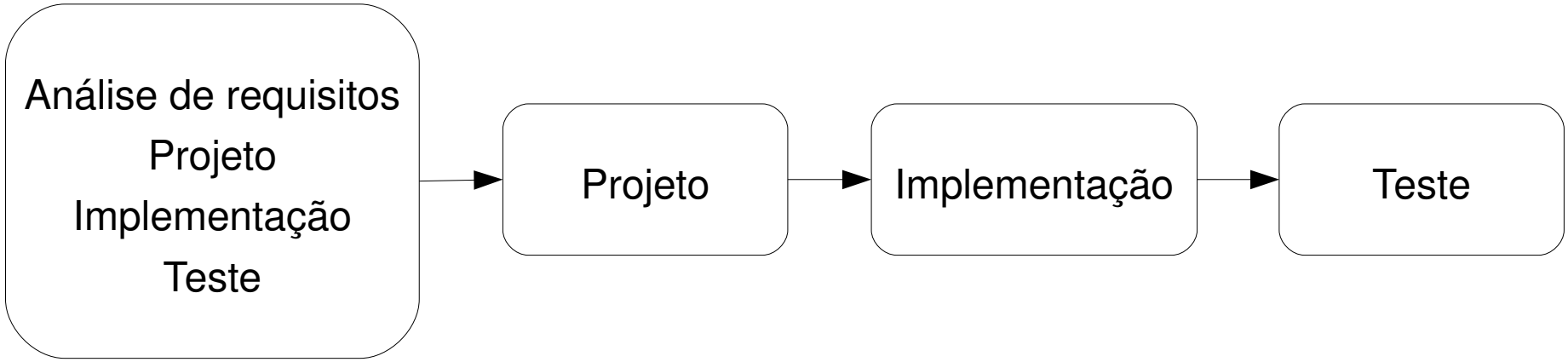
- Prototipação
  - O usuário define uma série de objetivos, mas não consegue identificar detalhes de entrada, processamento ou requisitos de saída.
  - O desenvolvedor está incerto sobre a eficiência de um algoritmo, a adaptação de um sistema operacional, ou ainda sobre a forma de interação homem-máquina.

Desenvolvimento Evolutivo

- Prototipação
  - Possibilita ao desenvolvedor criar um modelo do software que será construído.
  - O desenvolvedor pode perceber as reações iniciais do usuário e obter sugestões para mudar ou inovar o protótipo.
  - O usuário pode relacionar o que vê no protótipo diretamente com os seus requisitos.

Desenvolvimento Evolutivo

- Protótipo descartável
  - O objetivo é entender os requisitos do usuário e, consequentemente, obter uma melhor definição dos requisitos do sistema.



Desenvolvimento Evolutivo

- Problemas
  - Ausência de visibilidade do processo
    - Como o desenvolvimento acontece de maneira rápida, não compensa produzir documentos que reflitam cada versão do produto de software.
  - Sistemas são fracamente estruturados
    - Mudanças constantes tendem a corromper a estrutura do software.

Desenvolvimento Evolutivo

- Problemas
  - Necessidade de ferramentas de rápido desenvolvimento.
  - O usuário vê o que parece ser uma versão em funcionamento do produto de software.
  - O desenvolvedor, muitas vezes, assume certos compromissos de implementação.

Desenvolvimento Evolutivo

- Aplicabilidade:
  - Sistemas de pequeno e médio porte
    - Pois os problemas de mudanças no sistema atual podem ser contornados através da reimplementação do sistema todo quando mudanças substanciais se tornam necessárias.
  - Como parte de um sistema grande (ex.: a interface do usuário).

Desenvolvimento Evolutivo

- Aplicabilidade:
  - Sistema de curta duração.
  - Testes podem ser mais efetivos, visto que testar cada versão do sistema é provavelmente mais fácil do que testar o sistema todo no final.

Desenvolvimento Espiral

- Desenvolvido para englobar as melhores características do ciclo de vida clássico e do paradigma evolutivo.
- Adiciona um novo elemento – a análise de risco – que não existe nos dois paradigmas anteriores.
- Os riscos são as circunstâncias adversas que podem atrapalhar o processo de desenvolvimento e a qualidade do produto a ser desenvolvido.

Desenvolvimento Espiral

- Prevê a eliminação de problemas de alto risco através de um planejamento e projeto cuidadosos.
- As atividades podem ser organizadas como uma espiral que tem muitos ciclos.
- Cada ciclo na espiral representa uma fase do processo de desenvolvimento do software.

Desenvolvimento Espiral

- O primeiro ciclo pode estar relacionado com o estudo de viabilidade e com a operacionalidade do sistema; o segundo ciclo com a definição dos requisitos; o próximo com o projeto do sistema e assim por diante.
- Não existem fases fixas.

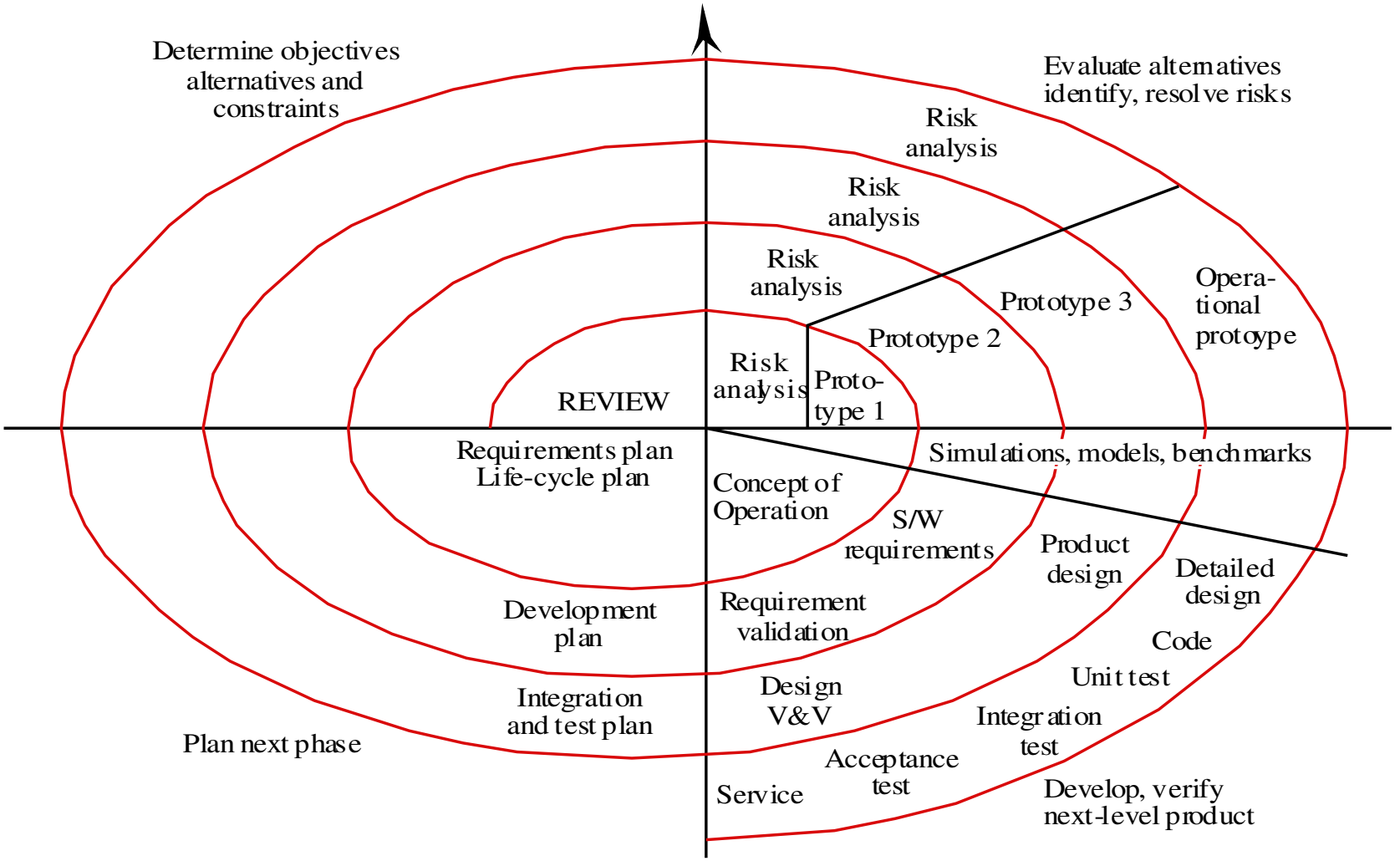
Desenvolvimento Espiral

- Fases do Modelo Espiral:
  - Definição dos objetivos, alternativas e restrições: um plano inicial é esboçado e os riscos do projeto são identificados.
  - Análise de risco: para cada um dos riscos identificados é feita uma análise cuidadosa.

Desenvolvimento Espiral

- Fases do Modelo Espiral:
  - Desenvolvimento e validação: após a avaliação dos riscos, um paradigma de desenvolvimento é escolhido.
  - Planejamento: o projeto é revisado e a decisão de percorrer ou não mais um ciclo na espiral é tomada. Se a decisão for percorrer mais um ciclo, então o próximo passo do desenvolvimento do projeto deve ser planejado.

Desenvolvimento Espiral



Desenvolvimento Formal de Sistemas

- Baseado na transformação de uma especificação matemática através de diferentes representações para um programa executável.
- Consegue alcançar os requisitos da especificação mais facilmente.

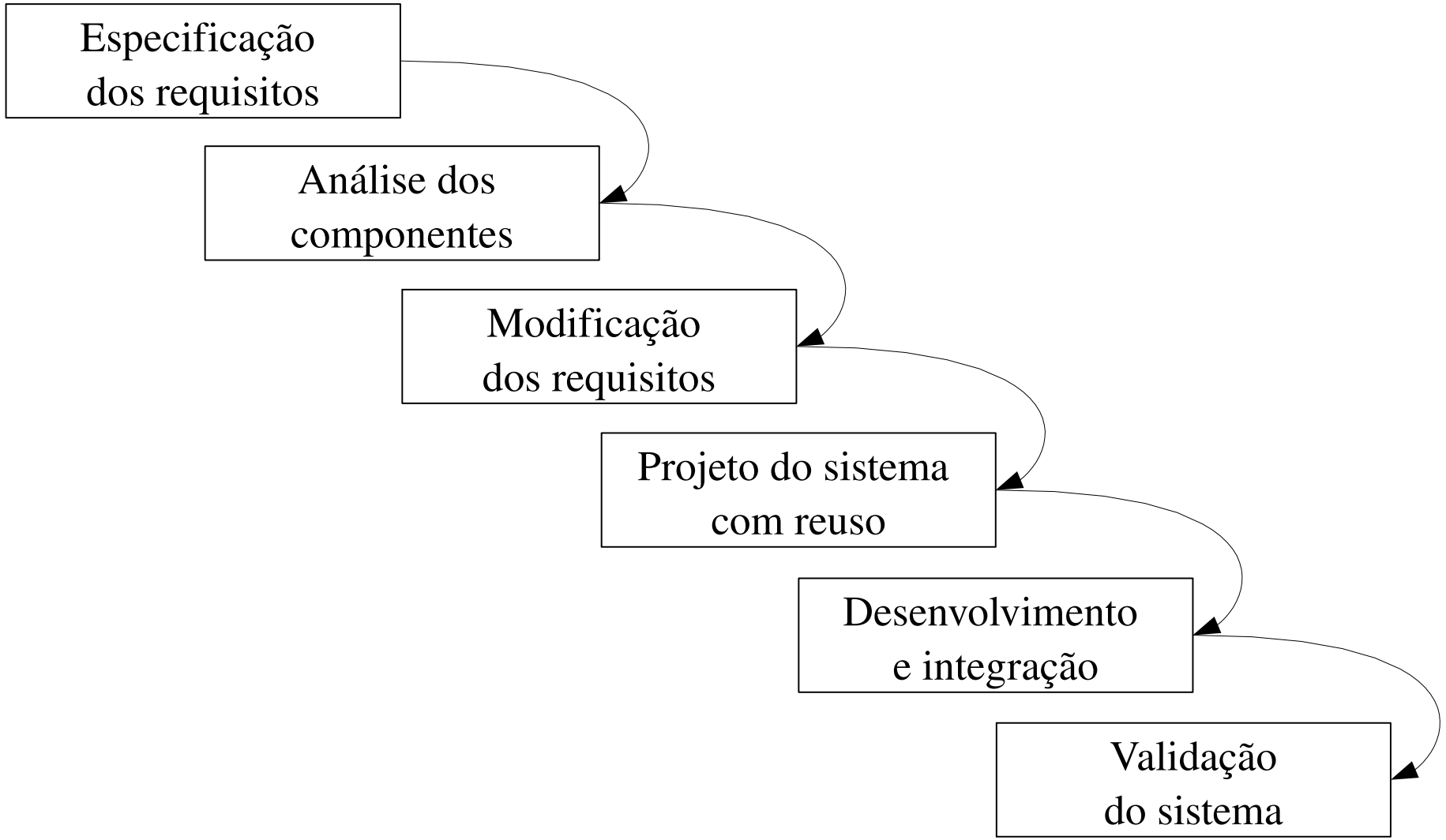
Desenvolvimento Formal de Sistemas

- Problemas:
  - Dificuldade em encontrar profissionais especializados.
  - Dificuldade em especificar determinados aspectos do sistemas, como a interface do usuário.
- Aplicabilidade:
  - Principalmente para sistemas críticos onde não são toleradas falhas.

Desenvolvimento Baseado em Reuso

- Sistemas baseados em componentes já existentes. Semelhantes ao desenvolvimento de hardware.
- Fases do processo:
  - Análise do componente.
  - Modificação dos requisitos.
  - Projeto do sistema com reuso.
  - Desenvolvimento e integração.
- Método que vem crescendo bastante nos últimos tempos.

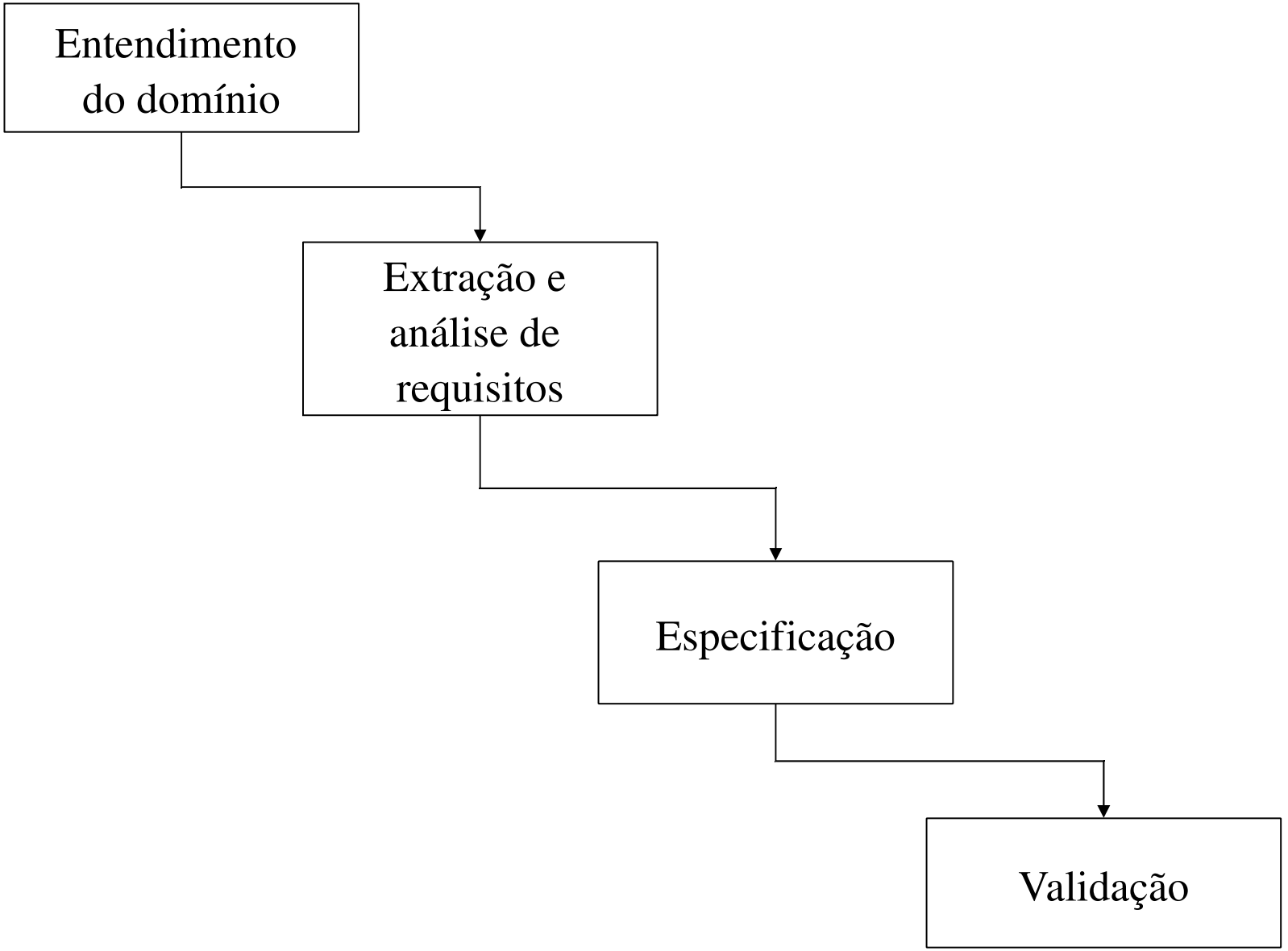
Desenvolvimento Baseado em Reuso



Extração de Requisitos

- Extração de requisitos é o processo de transformação das idéias que estão na mente dos usuários (a entrada) em um documento formal (saída).
- A saída do processo de extração de requisitos é um documento de especificação dos requisitos, que decreve **o que** o produto a ser desenvolvido deverá fazer, sem entretanto, descrever **como** deve ser feito.
- O processo de extração não pode ser totalmente automatizado.

Processo de Extração de Requisitos



Processo de Extração de Requisitos

- Entendimento do domínio: nessa fase, os desenvolvedores devem entender o domínio da aplicação o mais completamente possível.
- Extração e análise de requisitos: nessa etapa acontece a descoberta, revelação e entendimento dos requisitos, através de interação com o(s) usuário(s).

Processo de Extração de Requisitos

- Especificação dos requisitos: nessa etapa ocorre o armazenamento dos requisitos em uma ou mais formas, incluindo língua natural, linguagem semiformal ou formal, representações simbólicas ou gráficas.
- Validação dos requisitos: nessa etapa é feita a verificação dos requisitos, visando determinar se estão completos e condizentes com as necessidades e desejos do usuário.

Exercício

Gostaria que fosse construído um sistema para monitorar a temperatura e a pressão de pacientes da UTI, que deverão ficar ligados on-line à rede de computadores do hospital, que é formada por um computador principal e vários terminais que monitoram os pacientes. Se a temperatura ou pressão paciente lida pelo terminal se tornarem críticas, o computador principal deverá mostrar uma tela de alerta com um histórico das medidas realizadas para o paciente. Um aviso sonoro deve ser ativado nesse caso.

Exercício

A verificação da pressão é feita comparando-se a pressão do paciente com um valor padrão de pressão (máximo e mínimo) a ser digitado pelo responsável e verificando-se se a pressão medida está dentro dos parâmetros considerados normais para o paciente (valores próximos ao máximo e mínimo são permitidos). Temos vários sistemas on-line no computador e todos devem rodar ao mesmo tempo. [1]

Dificuldades Encontradas

- As atividades não podem ser totalmente separadas e executadas linearmente.
- As necessidades do usuário mudam à medida que o ambiente no qual o sistema funciona muda.
- Mudanças dos requisitos acontecem na maioria dos sistemas complexos.
- Falta de conhecimento do usuário das suas reais necessidades e do que o produto de software pode lhe oferecer.
- Falta de conhecimento do desenvolvedor do domínio do problema.

Dificuldades Encontradas

- Domínio do processo de extração de requisitos pelos desenvolvedores de software.
- Comunicação inadequada entre desenvolvedores e usuários.
- Dificuldade de o usuário tomar decisões.
- Problemas de comportamento.
- Questões técnicas.

Agentes da Extração de Requisitos

- Desenvolvedor (engenheiro de requisitos).
- Usuários.
- Gerente.

Técnicas para Extração e Análise de Requisitos

- As técnicas de extração de requisitos podem ser divididas em informais e formais.
- As técnicas informais são baseadas em comunicação estruturada e interação com o usuário, questionários, estudos de documentos etc. Exemplos: *Joint Application Design (JAD), brainstorming,* entrevistas *e* PIECES*.*

Técnicas para Extração e Análise de Requisitos

- As técnicas formais pressupõem a construção de um modelo conceitual do problema analisado, ou de um protótipo do produto de software a ser construído. Exemplos: modelo funcional, modelo de dados e o modelo de objetos.
- A prototipagem é utilizada quando o problema é analisado e os requisitos são entendidos através da interação com os usuário, a partir de um protótipo do produto.

Entrevistas

- Entrevistar não é somente fazer perguntas; é uma técnica estruturada, que pode ser aprendida e na qual os desenvolvedores podem ganhar proficiência com o treino e a prática.
- A entrevista consta de quatro fases:
  - Identificação dos candidatos para entrevista.
  - Preparação para uma entrevista.
  - Condução da entrevista.
  - Finalização da entrevista.

Brainstorming

- *Brainstorming* é uma técnica básica para geração de idéias. Ela consiste em uma ou várias reuniões que permitem que as pessoas sugiram e explorem idéias sem que sejam criticadas ou julgadas.
- Existem duas fases:
  - Geração de idéias.
  - Consolidação.

Brainstorming

- Geração de idéias:
  - É proíbido criticar as idéias.
  - Idéias não convencionais ou estranhas são encorajadas.
  - O número de idéias geradas deve ser bem grande.
  - Deve ser encorajada a participação de todos os agentes.

Brainstorming

- Consolidação das idéias:
  - As idéias são organizadas.
  - É nessa fase que as idéias são avaliadas.

Brainstorming

- Vantagens:
  - Estimula o pensamento imaginativo.
  - Evita a tendência a limitar o problema muito cedo.
  - Técnica fácil de ser aprendida.
- Desvantagem:
  - Por ser um processo não estruturado, pode não atingir o nível de detalhamento esperado.

PIECES

- A técnica de PIECES ajuda o analista a estruturar o processo de extração de requisitos.
- Indicado principalmente para analistas com pouca experiência em extração de requisitos.

PIECES

- PIECES
  - **Performance**
  - **Informação e dados**
  - **Economia**
  - **Controle**
  - **Eficiência**
  - **Serviços**

PIECES

- Perfomance (desempenho)
  - É necessário reconhecer as tarefas que o produto deverá executar e então idenficar o tempo de processamento ou tempo de resposta para cada tipo de tarefa.
  - Durante a análise de um produto de software já existente, é possível descobrir se os usuários experientes já sabem onde existem problemas de desempenho.

PIECES

- Informação e dados
  - Faz parte da natureza dos produtos de software o fornecimento de dados ou informações úteis para a tomada de decisão.

PI**E**CES

- Economia
  - Questões relacionadas ao custo de usar um produto de software são sempre importantes, e, de um modo geral, existem dois fatores de custo inter-relacionados que podem ser considerados no desenvolvimento de um sistema de software: nível de serviço e capacidade de lidar com alta demanda.

PIE**C**ES

- Controle
  - Os sistema são projetados para ter desempenho e saídas previsíveis.
  - Quando o sistema se desvia do desempenho esperado, algum controle deve ser ativado para tomar ações corretivas.

PIEC**E**S

- Eficiência
  - É a medida definida como a relação entre os recursos que resultam em trabalho útil e o total dos recursos gastos.
  - Eficiência é diferente de economia; para melhorar a economia do processo, a quantidade total de recursos utilizados deve ser reduzida; para melhorar a eficiência, a perda no uso desses recursos deve ser reduzida.

PIECE**S**

- Serviços
  - Um produto de software fornece serviços aos usuários, e pode ser muito útil pensar em termos de serviços durante o processo de extração de requisitos.

JAD

- ***J**oint **A**pplication **D**esign* é uma técnica para promover cooperação, entendimento e trabalho em grupo entre usuários e desenvolvedores.
- A técnica apresenta quatro princípios:
  - Dinâmica de grupo.
  - Uso de técnicas visuais.
  - Manutenção do processo organizado e racional.
  - Utilização de documentação-padrão.

Prototipagem

- A prototipagem pode auxiliar os usuários a entender e expressar melhor as suas necessidades através da comparação com um produto de software que sirva de referência.
- Atividades:
  - Estudo preliminar dos requisitos do usuário.
  - Processo interativo de construção do protótipo e avaliação junto dos usuários.

Prototipagem

- A prototipagem é benéfica somente se o protótipo puder ser construído substancialmente mais rápido que o sistema real.

Questionário

- Quais são as principais características do ciclo de vida clássico?
- Qual é a entrada e saída esperada de cada fase do ciclo de vida clássico?
- Qual é a diferença entre uma especificação funcional e não-funcional? Dê um exemplo de cada.
- Qual é a função da fase de integração de sistemas?

Questionário

- Comente dois problemas do modelo de ciclo de vida clássico.
- Qual é a diferença entre prototipação e desenvolvimento exploratório?
- O desenvolvimento evolutivo se aplica melhor ao desenvolvimento de sistemas em grande ou pequena escala? Justifique a sua resposta.

Questionário

- O que é análise de risco?
- Quais são as fases do desenvolvimento em espiral?
- Qual é a principal dificuldade em utilizar o desenvolvimento formal de sistemas?
- Por que o processo de extração de requisitos ainda não pode ser totalmente automatizado?

Questionário

- Na extração de requisitos, como funciona a técnica de brainstorming?
- A técnica chamada PIECES é aplicada em que tipo de sistemas?

Interfaces gráficas

- A maioria dos sistemas atuais apresenta interfaces gráficas, apesar de alguns sistemas legados ainda exigirem interfaces textuais.

Características das GUI

- Janelas
  - Múltiplas janelas permitem a apresentações de diferentes informações na tela do usuário.
- Ícones
  - Os ícones representam os diferentes tipos de informações.
- Menus
  - As operações são realizadas através da manipulação dos menus.

Características das GUI

- Cursores
  - O curso é uma representação do mouse e serve para indicar o tipo de operação que está sendo realizada.
- Gráficos
  - Elementos gráficos podem ser combinados com elementos textuais na mesma tela.

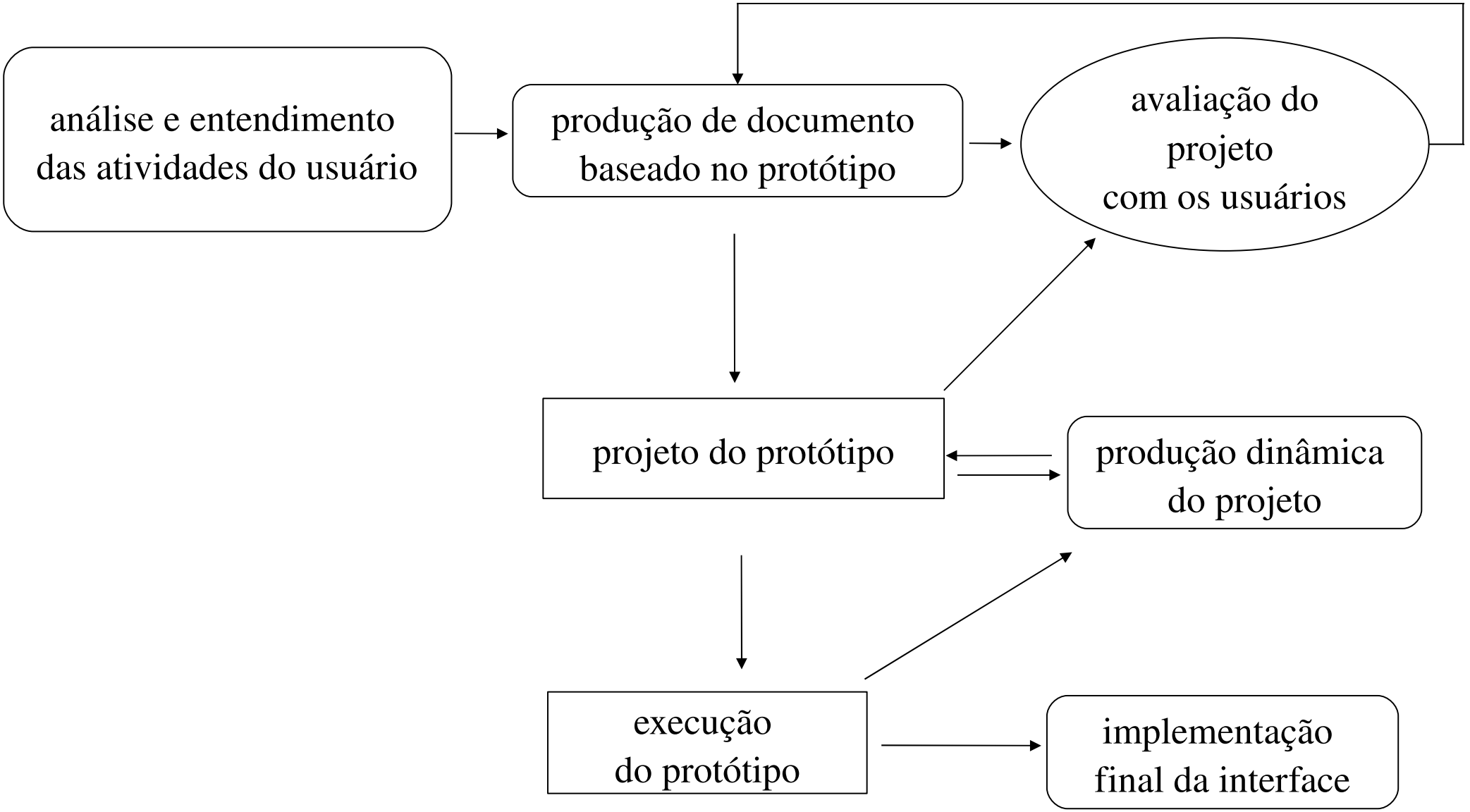
Vantagens GUI

- Elas são mais fáceis de entender e usar.
  - Os usuários sem experiência conseguem aprender mais rapidamente.
- O usuário pode trocar rapidamente a tarefa que está realizando e pode interagir com diferentes aplicações.
- Rapidamente o usuário consegue interagir com qualquer componente da tela.

Projeto centrado no usuário

- É importante que o projetista não se esqueça para quem está desenvolvendo a interface.
- O projeto centrado no usuário tem como parâmetro as necessidades do usuário e o seu envolvimento no processo.
- A prototipação é costumeiramente utilizada durante o processo.

Processo de Projeto de Interface



Princípios

- O projeto deve levar em consideração as necessidades, experiências e capacidades dos usuários.
- Os projetistas devem entender as limitações físicas e mentais dos usuários e reconhecer os erros mais comuns dos usuários.
- Às vezes é necessário fazer escolha de quais princípios não poderão ser utilizados.

Princípios de Projeto

- Familiaridade do usuário
  - A interface deve ser baseada em termos e conceitos orientados aos usuários.
- Consistência
  - O sistema deverá apresentar um nível apropriado de consistência. Por exemplo, comandos e menus devem ter o mesmo formato.

Princípios de Projeto

- Mínima surpresa
  - Se os comandos funcionam de maneira conhecida, o usuário será capaz de “advinhar” o comportamento do sistema.
- Recuperação
  - O sistema deverá ser flexível o suficiente para permitir que os usuários possam recuperar os erros cometidos.

Princípios de Projeto

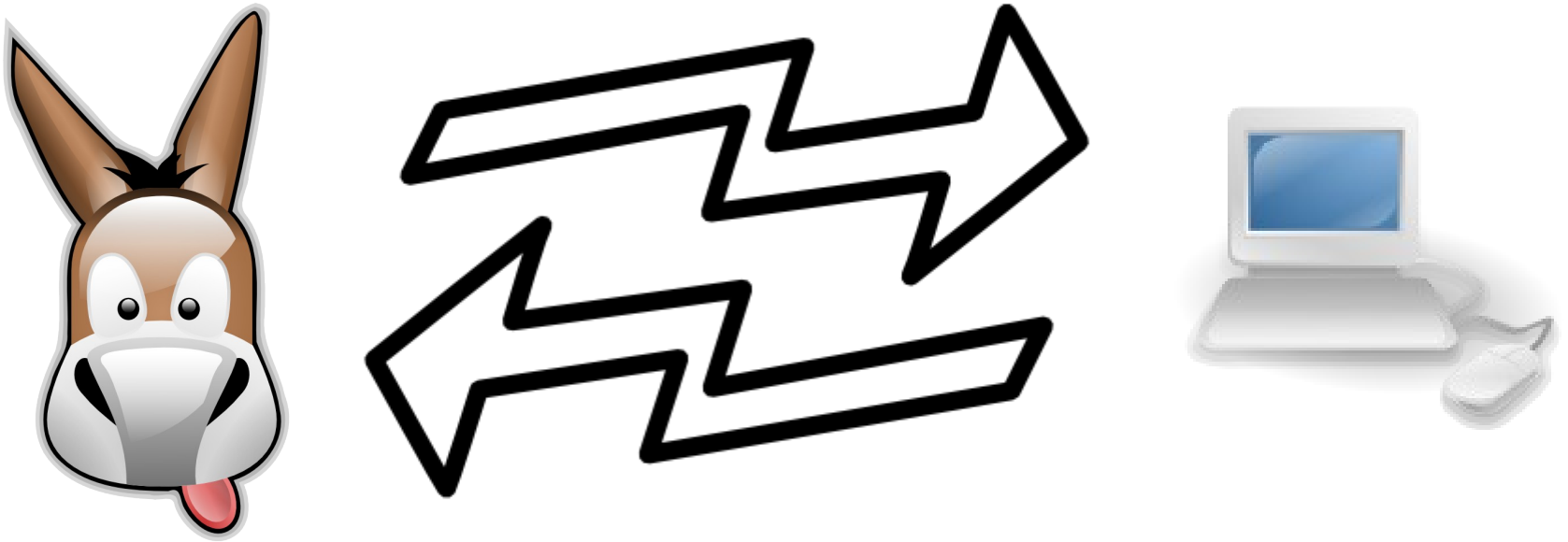
- Auxílio ao usuário
  - Fornecimento de auxílio ao usuário na operação do sistema.
- Diversidade de usuários
  - O sistema deverá suportar os diferentes tipos de usuários.

Interação Homem-Máquina

- Duas importantes questões sobre interação devem ser consideradas:
  - Como as informações serão passadas do usuário para o computador?
  - Como as informações, oriundas do computador, serão apresentadas ao usuário?

Interação Homem-Máquina

- A interação com o usuário e a apresentação da informação devem ser integradas através de um coerente framework, de acordo com a metáfora apresentada ao usuário.



Estilos de Interação

- Manipulação direta
- Seleção de menus
- Preenchimento de formulários
- Linguagem de comandos
- Linguagem natural

Cores

- As cores adicionam uma dimensão extra para uma interface e pode ajudar ao usuário entender complexas estruturas de informação.
- Podem ser utilizadas para destacar eventos excepcionais.
- Erros comuns no uso das cores são:
  - O uso das cores para se comunicar.
  - Excesso de cores na tela.

Dicas de Uso de Cores

- Não utilizar muitas cores.
- Usar códigos de cores.
- Permitir aos usuários manipular os códigos das cores.
- Projeto para interface colorida e monocromática.
- Utilizar o código das cores consistentemente.
- Evitar conflito no uso das cores.

Dicas de Uso de Cores

- A mudança das cores deve representar a mudança de um evento.
- O projeto deve considerar monitores de baixa resolução.

Suporte ao Usuário

- O auxílio ao usuário deve incluir:
  - Help on-line.
  - Mensagens de erro.
  - Manuais.
- O sistema de ajuda deve estar integrado ao sistema.

Mensagens de Erro

- O projeto de mensagens de erro é parte crítica.

  Mensagens ruins podem levar os usuários a não aceitar o sistema.

- As mensagens devem ser educadas, concisas, consistências e construtivas.

- A experiência dos usuários deve ser um fator relevante na construção das mensagens de erro.

Mensagens de Erro

- Aspectos a serem considerados
  - Contexto: as mensagens devem se adaptar ao contexto da operação que está sendo executada pelo usuário.
  - Experiência: o texto das mensagens devem ser escritos de acordo com o nível de experiência de cada usuário.
  - Estilo: deve-se tentar escrever mais mensagens positivas do que negativas.
  - Cultura: se possível, o texto das mensagens deve estar de acordo com a cultura dos usuários.

Documentação do Usuário

- Assim como as informações *on-line*, deverá ser fornecida uma documentação em papel para os usuários.
- A documentação deverá ser preparada para todos os níveis de usuários.
- Assim como os manuais, outros documentos para consulta rápida deverão ser produzidos.

Tipos de Documentos

- Descrição Funcional
  - Rápida descrição do que o sistema pode fazer.
- Manual Introdutório
  - Apresenta uma introdução informal do sistema.
- Manual de Referência do Sistema
  - Descreve todas as funcionalidades, em detalhes, do sistema.
- Manual de Instalação do Sistema
  - Descreve como instalar o sistema.

Tipos de Documentos

- Manual de Administração do Sistema
  - Descreve como gerenciar o sistema quando ele está em funcionamento.

Avaliação da Interface

- Deve ser realizada uma avaliação da interface.
- Avaliação completa do sistema é muito cara e impraticável na maioria dos sistemas.
- Idealmente, uma interface deve ser avaliada de acordo com a especificação do sistema.

Atributos de Usabilidade

- Facilidade de aprendizado
  - Quanto tempo um novato precisa para poder se tornar produtivo ao sistema?
- Velocidade de operação
  - Como o sistema responde às operações do usuário?
- Robustez
  - Qual é o nível de tolerância de erros do sistema?

Atributos de Usabilidade

- Recuperação
  - Qual é a capacidade de recuperação do sistema?
- Adaptabilidade
  - Quão próximo o sistema está do modelo atual de trabalho da empresa?

Simples Técnicas de Avaliação

- Elaboração de questionários para o usuário.
- Gravação de vídeo de uso do sistema e uma posterior avaliação.
- Geração de código para coletar informações sobre a facilidade de uso e os erros dos usuários.
- O fornecimento de componentes que permitam uma avaliação do sistema por parte usuário (*feedback*).

Questionário

- Por que a técnica de prototipação é, costumeiramente, empregada no desenvolvimento de interfaces?
- Familiaridade e facilidade são características similares no desenvolvimento de interfaces para software?
- Qual é a importância na construção de interfaces gráficas?

Questionário

- Quais são os tipos de documentos que devem ser fornecidos juntamente com o software?
- Como é possível avaliar a qualidade da interface de um sistema?

Verificação e Validação

- Consiste na checagem e processo de análise para garantir que o software está de acordo com a sua especificação e que atende às necessidades dos clientes que contrataram o software.

Verificação e Validação

- Diferenças:
  - Validação
    - O software está de acordo com a necessidade do cliente?
  - Verificação
    - O software está de acordo com a especificação?

O Processo de Verificação e Validação

- A validação e verificação devem ser aplicadas em cada estágio do processo de desenvolvimento de software.
- Possui dois principais objetivos:
  - Descobrir defeitos do sistema.
  - Garantir a utilidade e a usabilidade do sistema numa situação operacional.

Objetivos da Verificação e Validação

- Estabelecer a confiança de que o software atende à sua finalidade.
- Isto não garante que o mesmo é TOTALMENTE livre de erros.
- O grau de confiança esperado depende do tipo de sistema.

Confiança na Verificação e Validação

- O grau de confiança depende da finalidade do sistema, das expectativas do usuário e do mercado.
  - Finalidade do software:
    - Quanto mais crítico for para a empresa, maior deverá ser o grau de confiança.
  - Expectativas do usuário:
    - Os usuários podem ter baixas expectativas sobre certos tipos de softwares.

Confiança na Verificação e Validação

- O grau de confiança depende da finalidade do sistema, das expectativas do usuário e do mercado.
  - Expectativas do mercado:
    - Entregar logo um produto ao mercado pode ser mais importante do que encontrar defeitos num programa.

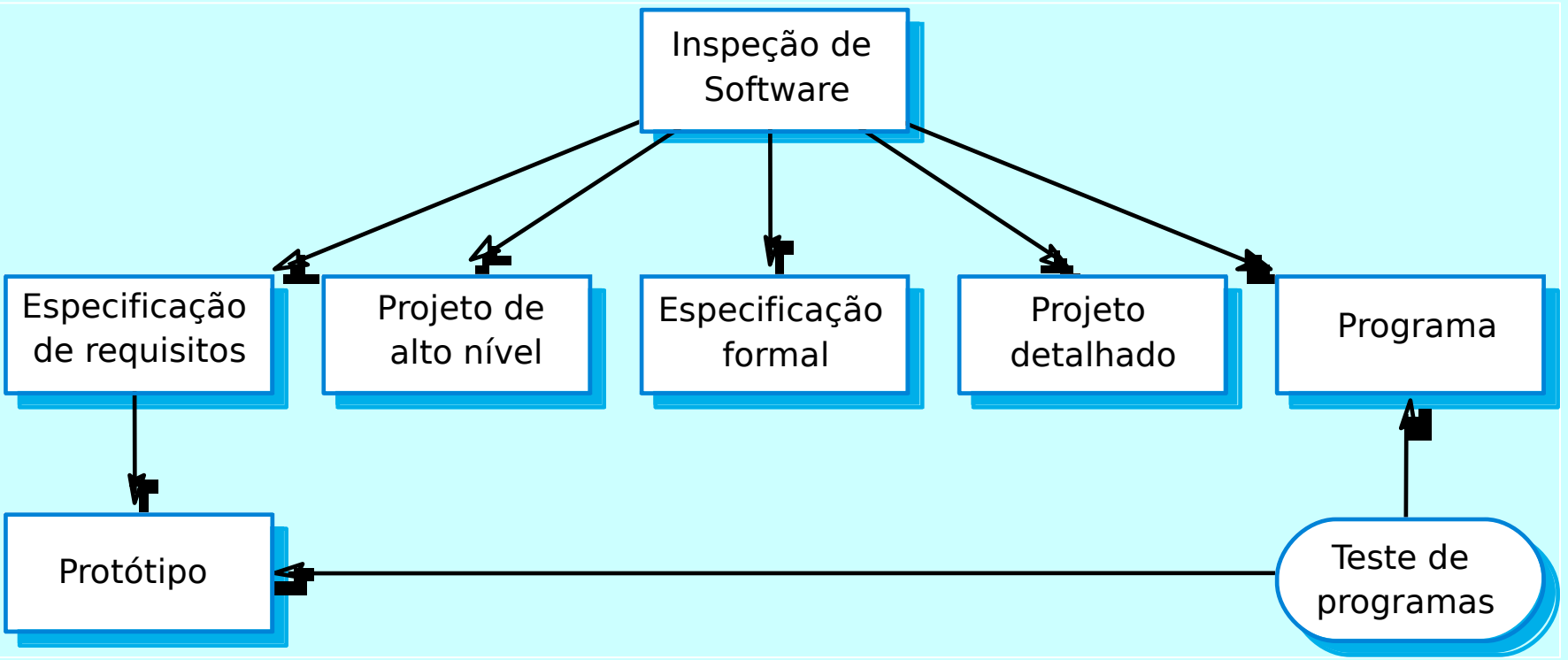
Verificação Estática e Dinâmica

- Técnicas
  - Inspeção de software
    - Análise e checagem do documento de requesitos do sistema, diagramas do projeto e do código fonte.
    - Pode ser aplicada em todos os estágios do processo.
    - É uma técnica estática por não necessitar que o sistema seja executado.

Verificação Estática e Dinâmica

- Técnicas
  - Teste de software
    - Envolve executar uma implementação do software com o teste dos dados e examinar as saídas e o comportamento operacional.
    - É uma técnica dinâmica porque trabalha com a representação executável do sistema.

Verificação Estática e Dinâmica



Testar Programas

- Pode revelar a presença de erros, mas NÃO a ausência.
- Apesar das inspeções de software serem cada vez mais utilizadas, o teste de programas ainda é a principal técnica de verificação e validação.
- O teste envolve o exercício de passar dados, mais próximo do real, pelo programa.

Tipos de Teste

- Teste de defeitos
  - A sua finalidade é descobrir os defeitos do sistema.
  - Um teste de defeito correto é aquele que consegue revelar os defeitos do sistema.

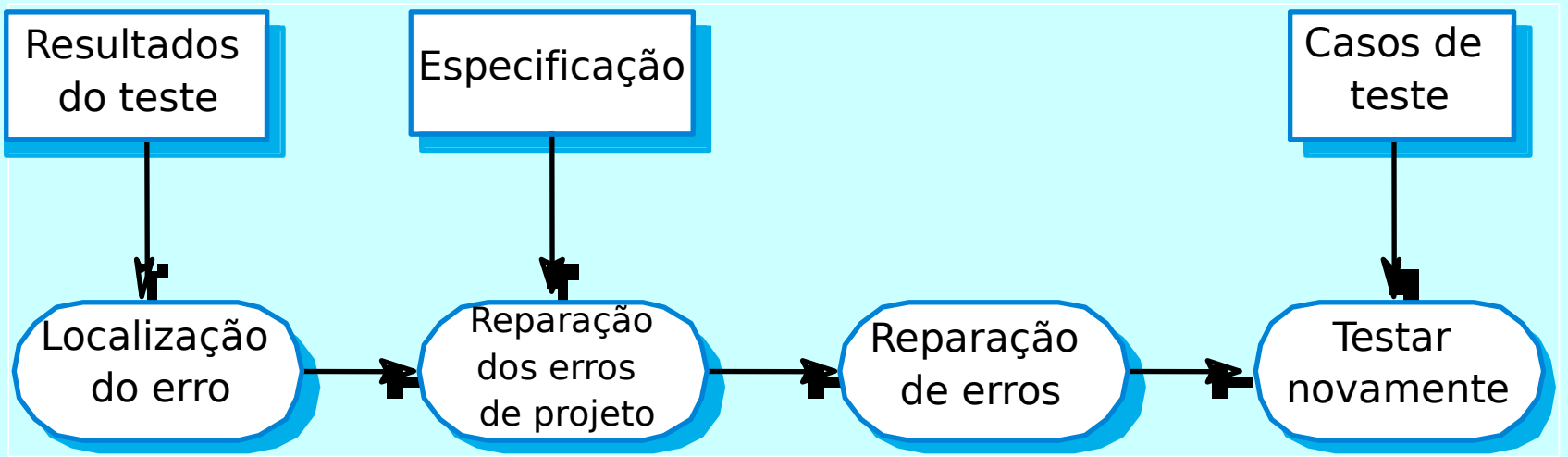
Tipos de Teste

- Teste de validação
  - Tem a finalidade de demonstrar que o software atende aos seus requisitos.
  - Este teste tem sucesso quando consegue mostrar que os requisitos foram implementados corretamente.

Testes e Depuração

- São atividades distintas.
- Verificação e validação são voltados para a identificação dos defeitos do programa.
- Depuração (*debugging*) tem como objetivo a localização e a reparação dos erros.
- A depuração envolve formular uma hipótese sobre o comportamento do programa e então testar esta hipótese para encontrar o erro do sistema.

Processo de Depuração



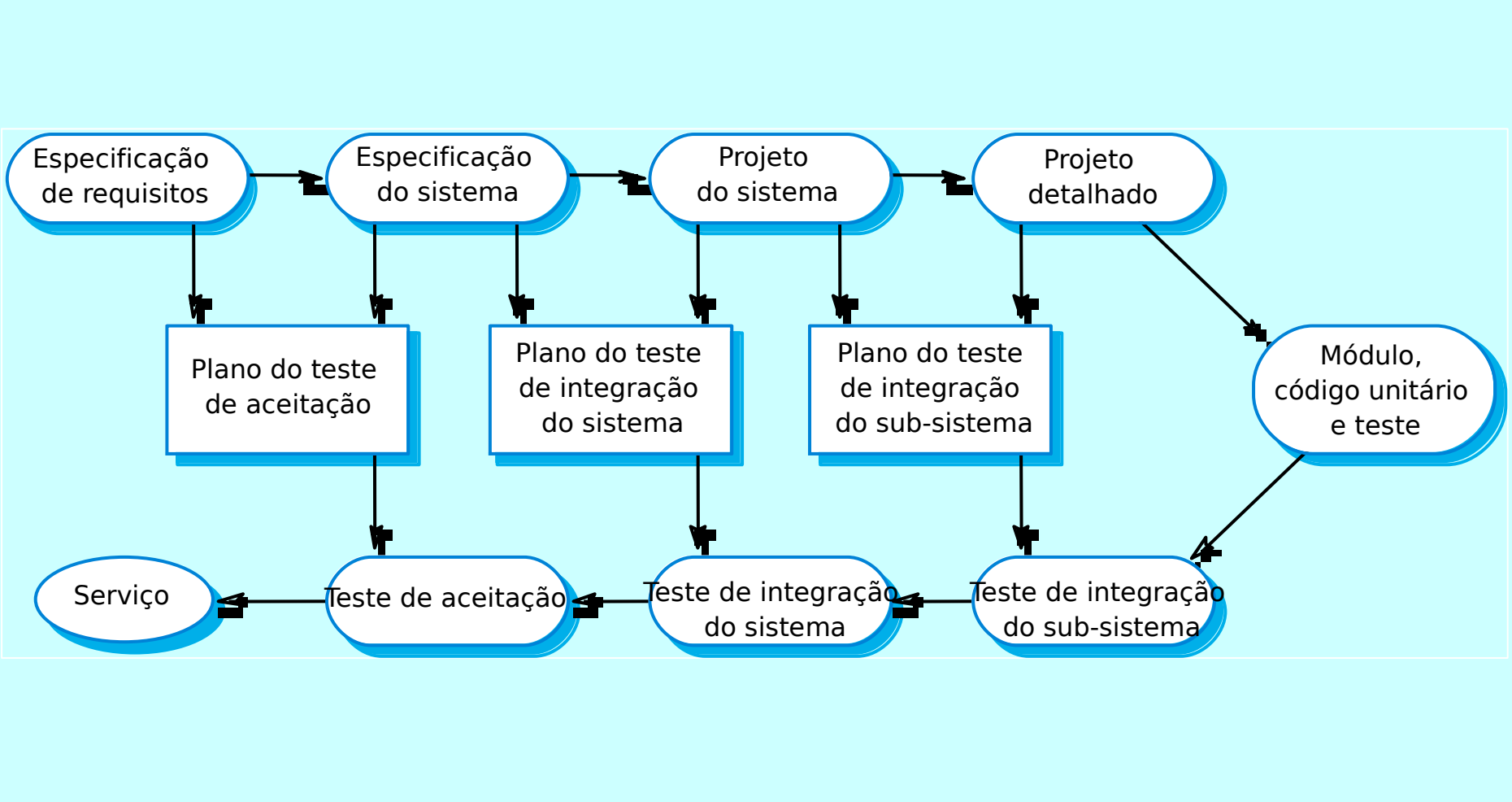
Planejamento de Verificação e Validação

- O processo de verificação e validação é caro e depende do tipo de sistema.
- Metade do custo pode ser alocado para esta fase.
- O planejamento é necessário para a realização deste processo com sucesso.
- O planejamento deve começar no início do processo.

Planejamento de Verificação e Validação

- O planejamento deve balancear os testes estáticos e dinâmicos.
- O planejamento do teste consiste em definir os padrões do processo de teste, além de descrever os produtos de teste.

Modelo-V


Especificação de requisitos
Especificação do sistema
Projeto do sistema
Projeto detalhado
Plano do teste de aceitação
Plano do teste de integração do sistema
Plano do teste de integração do sub-sistema
Módulo, código unitário e teste
Serviço
Teste de aceitação
Teste de integração do sistema
Teste de integração do sub-sistema

Estrutura de um Plano de Teste de Software

- O processo de teste
  - Descrição das principais fases do processo de teste.
- Rastreamento dos requisitos
  - Os usuários são os maiores interessados de que o sistema esteja de acordo com os requisitos e o teste deve ser planejado para que todos os requisitos sejam individualmente testados.

Estrutura de um Plano de Teste de Software

- Itens testados
  - Os produtos do processo de software a serem testados devem ser especificados.
- Agenda de testes
  - Os recursos para realização dos testes devem ser agendados.

Estrutura de um Plano de Teste de Software

- Procedimentos para gravação do teste
  - Não basta executar os testes. É necessário que os resultados sejam sistematicamente gravados. É importante, inclusive, para uma auditoria.
- Requisitos de hardware e software
  - Relação das ferramentas necessárias e estimativa da configuração de hardware exigida.

Estrutura de um Plano de Teste de Software

- Restrições
  - As restrições afetam o processo de teste, tais como escassez de pessoal, e devem ser consideradas.

Inspeções de Software

- Estas atividades envolvem pessoas que irão examinar os fontes com o objetivo de descobrir anomalias e defeitos.
- As inspeções não exigem a execução de um sistema para que possa ser utilizado antes da implementação.
- Eles podem ser aplicados para qualquer representação do sistema (requisitos, projetos, configuração de dados, testes de dados, etc).
- A técnica é útil para descobrir erros nos programas.

Sucesso na Inspeção

- Muitos defeitos diferentes podem ser descobertos numa simples inspeção.
- Num teste, um defeito pode mascarar muitos outros defeitos.
- A repetição do domínio e o conhecimento da programação são tipos de erros comuns.

Inspeções e Testes

- As inspeções e testes são técnicas complementares e não excludentes.
- Ambas podem ser utilizadas durante o processo de validação e verificação.
- As inspeções podem ser utilizadas para checar a especificação, mas não para checar os reais requisitos dos usuários.
- As inspeções não podem checar características não-funcionais (performance, usabilidade, etc).

Inspeções do Programa

- Abordagem formalizada para revisões de documentos.
- Voltada para a detecção de defeitos (não correção).
- Os defeitos podem ser erros lógicos, anomalias no código e que podem indicar uma condição erronêa.

Pré-condições das Inspeções

- A especificação precisa deve estar disponível.
- Os membros das equipes devem estar familiarizados com as padronizações da organização.
- O código-fonte e outras representações do sistema devem estar disponíveis.
- Uma lista de erros deve ser preparada.
- O gerenciamento deve aceitar que a inspeção aumentará os custos no início do processo.

Procedimento das Inspeções

- Uma visão geral da inspeção deve ser apresentada para a equipe.
- Código-fonte e documentos associados devem ser distribuídos para inspeção da equipe.
- As inspeções e os erros descobertos devem anotados.
- Modificações são feitas para reparar os erros descobertos.
- A re-inspeção pode ou não ser necessária.

Pessoas na Inspeção

- Autor
  - O programa ou o projetista responsável para a produção do programa ou documento. Responsável por corrigir os defeitos descobertos durante o processo de inspeção.
- Inspetor
  - Responsável por encontrar erros, omissões e inconsistências nos programas e documentações.
- Leitor
  - Apresentar o código ou documentos no encontro das inspeções.

Pessoas na Inspeção

- Escritor
  - Registra os resultados da atividade de inspeção.
- Líder
  - Gerencia o processo e facilita a inspeção.
- Moderador
  - Responsável pela melhoria do processo de inspeção, atualização do checklist e desenvolvimento dos padrões.

Checklist da Inspeção

- Deve ser elaborado um checklist dos erros mais comuns para orientar a inspeção.
- Os erros do checklist são dependentes da linguagem de programação e refletem as características de erros que estão relacionados com a linguagem.

Taxas de Inspeção

- De acordo com Sommerville:
  - A inspeção é um processo caro.
  - É possível realizar 500 inspeções/hora durante a inspeção geral.
  - São realizadas 90/125 declarações/hora.
  - A inspeção de 500 linhas custa o esforço de 40 homens/hora.

Verificação e Métodos Formais

- Métodos formais podem ser utilizados quando uma especificação matemática do sistema é produzida.
- É normalmente a última técnica de verificação estática a ser utilizada.
- Eles envolvem detalhada análise matemática da especificação e podem desenvolver argumentos formais para verificar se o programa está em conformidade com a especificação matemática.

Desenvolvimento Limpo de Software

- A filosofia do método '*cleanroom*' é evitar o defeito, assim não será preciso removê-lo.
- Este processo de software é baseado em:
  - Desenvolvimento incremental.
  - Especificação formal.
  - Verificação estática utilizando argumentos corretos.
  - Testes estatísticos para determinar a confiabilidade do programa.

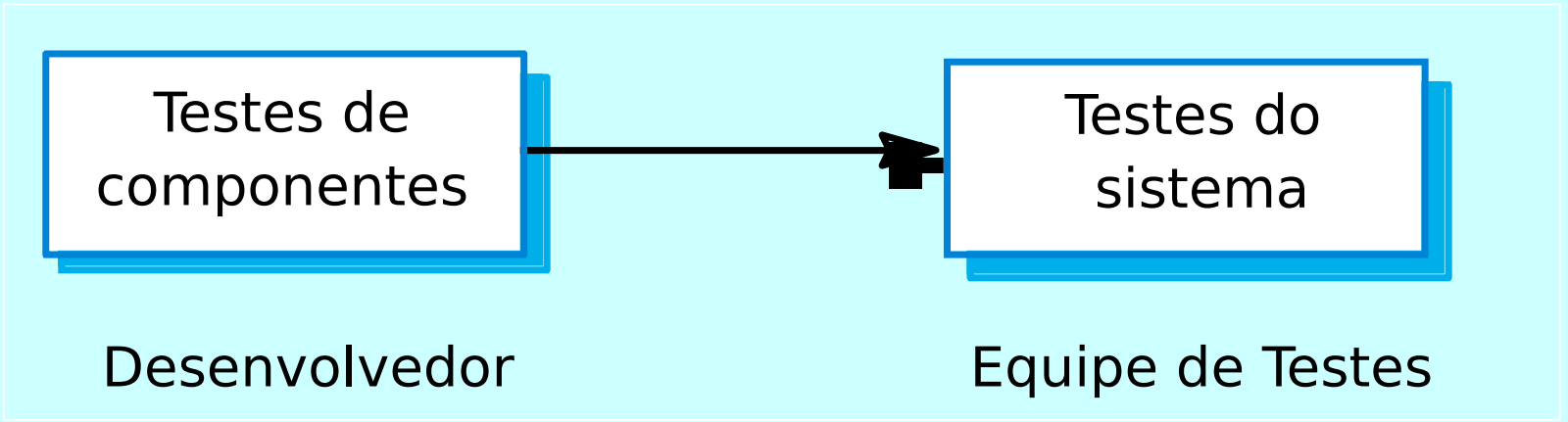
Processo de Teste

- Teste de componentes
  - Teste dos componentes de programas individualmente.
  - A responsabilidade normalmente fica a cargo do desenvolvedor.
  - Os testes são derivados da experiência do desenvolvedor.

Processo de Teste

- Teste do sistema
  - Teste dos grupos de componentes integrados para criar um sistema ou um sub-sistema.
  - A responsabilidade normalmente fica a cargo da equipe de testes.
  - Os testes são baseados na especificação do sistema.

Fases do Processo de Teste



Teste de Defeitos

- O objetivo é descobrir os defeitos dos programas.
- Um teste é realizado com sucesso quando é possível perceber um comportamento anômalo.
- Os testes mostram a presença de defeitos, mas não conseguem afirmar a ausência.

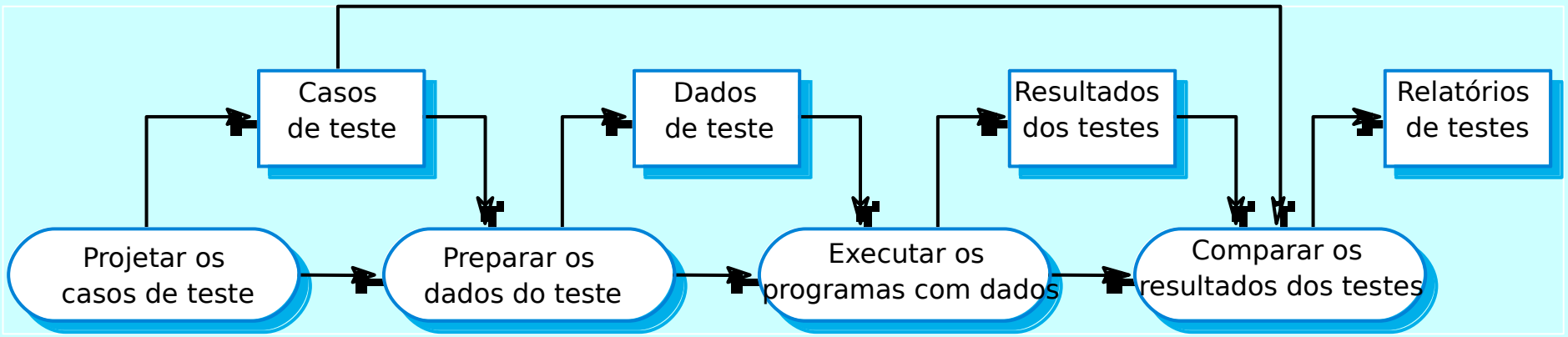
Objetivos do Processo de Testes

- Testes de validação
  - Têm o objetivo de demonstrar ao desenvolvedor e ao cliente que o programa está de acordo com os requisitos.
  - Um teste com sucesso mostra que o sistema opera comforme foi projetado.

Objetivos do Processo de Testes

- Testes de defeitos
  - Tem o objetivo descobrir as falhas ou defeitos no software onde o seu comportamento é incorreto ou não está de acordo com a sua especificação.
  - Um teste com sucesso é aquele que faz o sistema executar erros e, assim, expor os defeitos.

Processo de Teste de Software



Políticas de Testes

- Somente a execução exaustiva de testes pode mostrar que um programa está livre de defeitos. Porém, o teste exaustivo é inviável.

Políticas de Testes

- As políticas de testes definem a abordagem a ser utilizada na seleção dos testes de sistema:
  - Todas as funções do menu devem ser testadas.
  - Combinações de funções acessadas através do mesmo menu devem ser testadas.
  - Se for necessária a entrada de dados, todas as funções devem ser testadas com dados corretos e dados incorretos.

Testes de Sistemas

- Envolve a integração de componentes para criar um sistema ou sub-sistema.
- Podem envolver testes incrementais para serem entregues ao cliente.

Testes de Sistemas

- Duas fases:
  - Teste de integração
    - A equipe de testes tem acesso ao código-fonte.
    - O sistema é testado para que os componentes possam ser integrados.
  - Teste de release
    - A equipe de teste testa o sistema completo para ser entregue como uma caixa-preta.

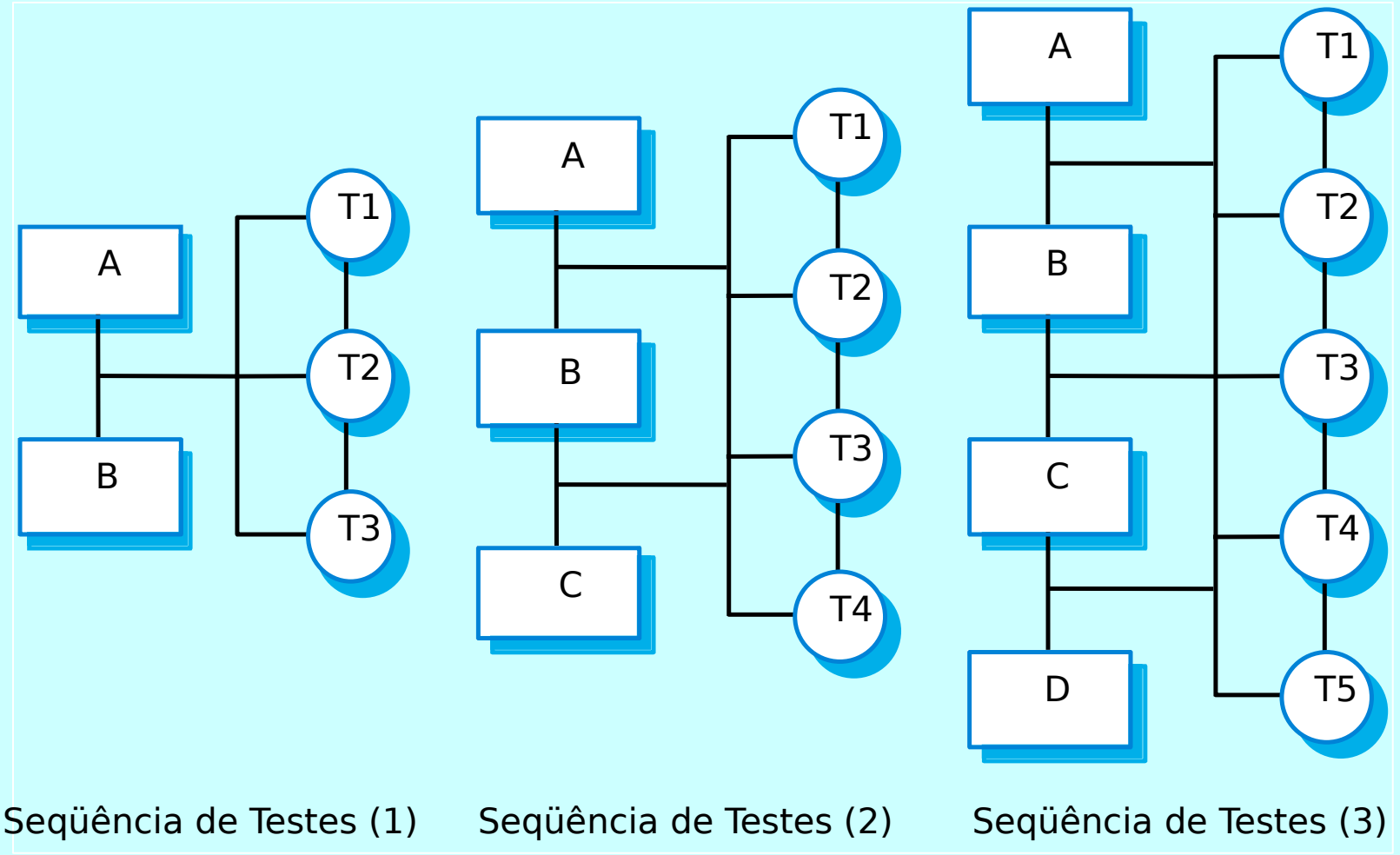
Teste de Integração

- Envolve a construção de um sistema a partir dos seus componentes e testes.
- Integração top-down
  - Desenvolve o esqueleto do sistema e povoa com os seus componentes.

Teste de Integração

- Integração *bottom-up*
  - Integra a infraestrutura dos componentes e então adiciona as suas funcionalidades.
- Para simplificar a localização de erros, os sistemas devem ser incrementalmente integrados.

Testes de Integração Incremental



Abordagens de Testes

- Validação da arquitetura
  - O teste de integração top-down é o mais indicado para a descoberta de erros na arquitetura do sistema.
- Demonstração do sistema
  - O teste de integração top-down permite uma demonstração limitada no estágio inicial de desenvolvimento.

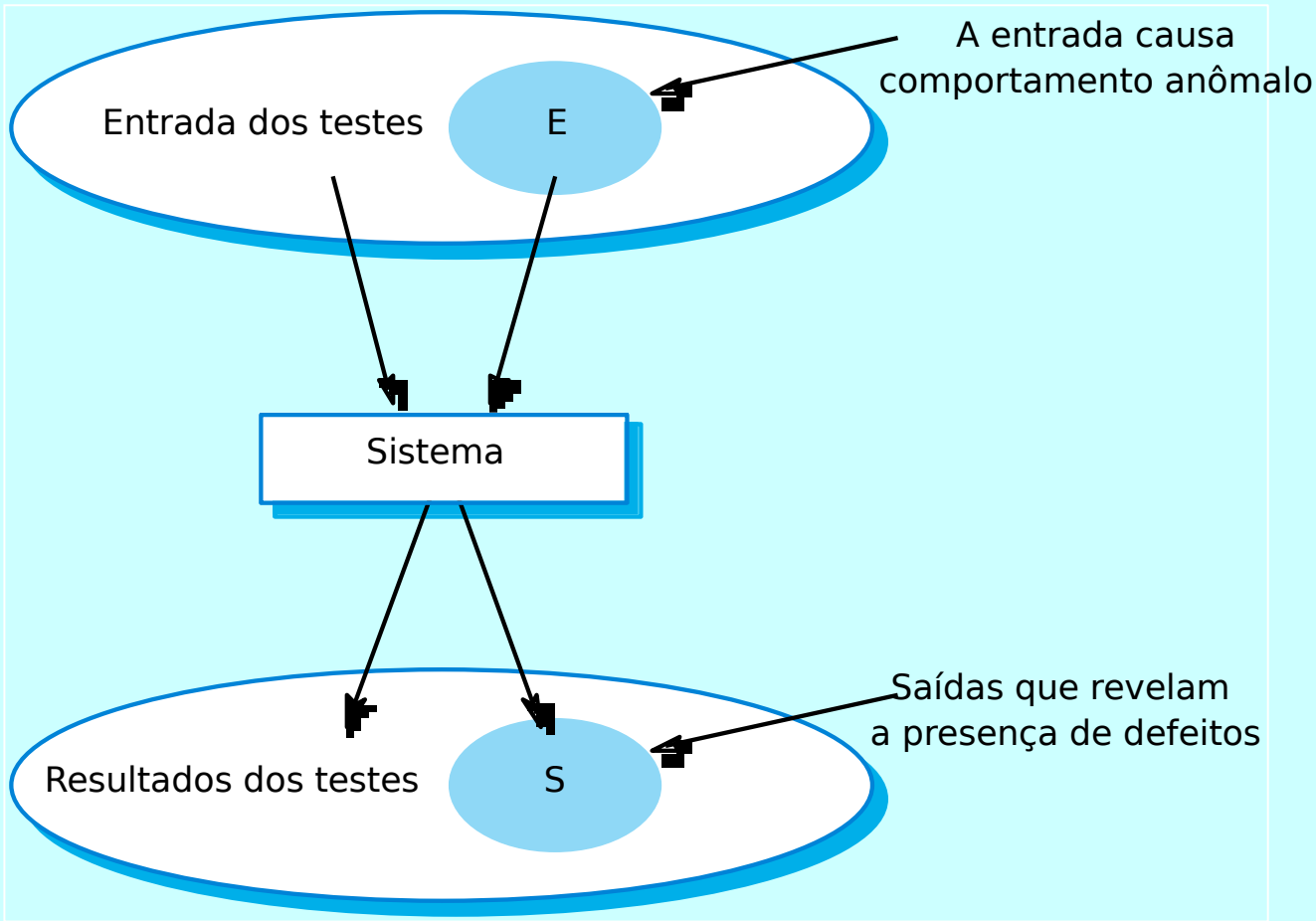
Abordagens de Testes

- Implementação do teste
  - É mais fácil com o teste de integração bottom-up.
- Observação dos testes
  - Problemas acontecem com as duas abordagens.
  - Código extra pode ser necessário para a observação dos testes.

Testes de Release

- É o processo de testar uma release de um sistema que será distribuída ao cliente.
- O objetivo é aumentar o grau de confiança do fornecedor que o software atende aos seus requisitos.
- Teste de release é normalmente um teste caixa-preta ou teste funcional
  - É baseado somente na especificação do sistema.
  - A equipe de testes não tem conhecimento sobre a implementação do sistema.

Teste Caixa-Preta



Roteiros de Testes

- Um roteiro para ajudar às equipes de testes conseguirem revelar os defeitos:
  - Escolher entradas que forçam o sistema a gerar todas as mensagens de erro.
  - Projetar entradas que causam overflow no buffer.
  - Repetir a mesma entrada ou série de entradas por diversas vezes.

Roteiros de Testes

- Um roteiro para ajudar às equipes de testes conseguirem revelar os defeitos:
  - Forçar que as saídas inválidas sejam geradas.
  - Força que a computação de resultados muito grandes ou muito pequenos.

Casos de Uso

- Os casos de uso podem servir de base para derivar os testes para um sistema.
- Eles ajudam a identificar as operações a serem testadas e ajudam a projetar os casos de testes necessários.
- A partir de um diagrama de seqüência, as entradas e saídas podem ser identificadas para os testes.

Execução dos Testes

- Parte do teste de release pode envolver testar as propriedades emergentes de um sistema, tais como performance e confiabilidade.
- A execução dos testes normalmente envolve o planejamento de uma série de testes onde a carga vai sendo incrementada até que a performance do sistema se torne inaceitável.

Teste de Stress

- Exercita o sistema, além de verificar a carga máxima suportada.
- Este tipo de teste checa a perda de dados ou serviços inaceitáveis.
- O teste de stress é particularmente relevante para sistemas distribuídos, pois podem exibir uma degradação severa com uma rede sobrecarregada.

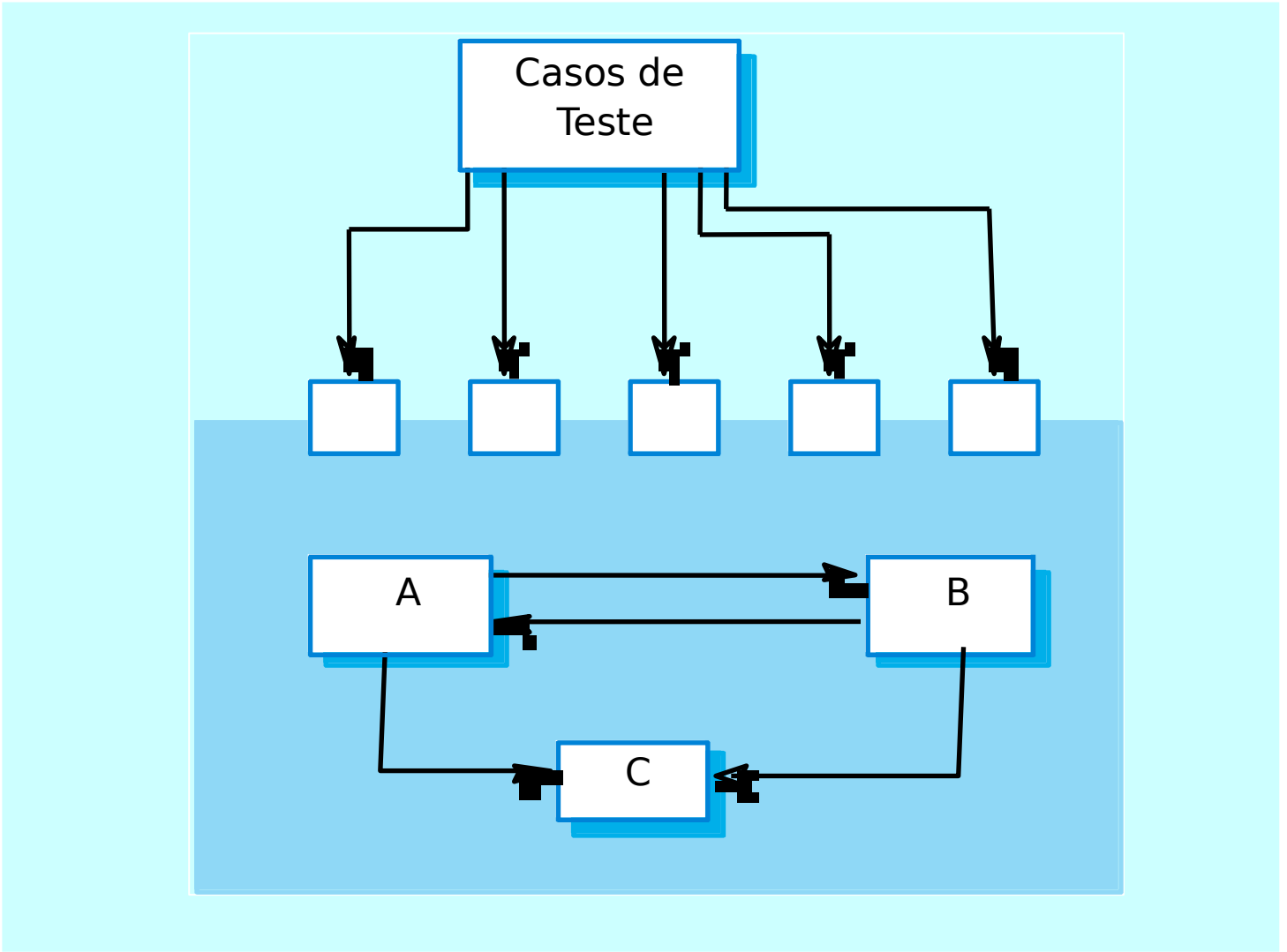
Teste de Componentes

- Teste de componentes ou unitários é o processo de testar individualmente os componentes de maneira isolada.
- Ele é um processo para testar defeitos.
- Os componentes podem ser:
  - Funções individuais ou métodos dentro de um objeto.
  - Classes com muitos atributos e métodos.
  - Composição de componentes com interfaces definidas usadas para acessar as suas funcionalidades.

Teste da Interface

- Tem como objetivo detectar as falhas devido aos erros da interface ou suposições errôneas sobre a interface.
- São importantes principalmente para desenvolvimento orientado-objetos como objetos são definidos por suas interfaces.

Teste de Interface



Tipos de Interface

- Parâmetros da interface
  - Os dados são passados de um procedimento para outro.
- Interfaces que possuem memória compartilhada
  - Bloco de memória é compartilhado entre os procedimentos e funções.

Tipos de Interface

- Interfaces procedurais
  - Sub-sistemas encapsulam um conjunto de procedimentos que podem ser chamados por outros sub-sistemas.
- Passagem de mensagens
  - Sub-sistemas requerem serviços de outros sub-sistemas.

Erros de Interfaces

- Problemas de mau uso
  - Ocorre quando uma chamada a um componente ativa um outro componente e faz um erro no seu uso. Por exemplo, a ordem dos parâmetros está errada.
- Dificuldades de entendimento
  - Ocorre quando uma chamada a um componente e este tem um comportamento diferente do esperado.
- Erros temporais
  - Quando, devido a uma falha temporal, o objeto acionado opera em diferentes velocidades.

Roteiro de Testes de Interface

- Projetar os testes para que os parâmetros para um procedimento.
- Testar a passagem de parâmetro com nulos.
- Projetar testes que podem causar a falha dos componentes.
- Utilizar teste de stress na passagem de mensagens do sistema.
- Em sistemas com memória compartilhada, varie a ordem em que os componentes são ativados.

Questionário

- Qual é a diferença entre validação e verificação de software?
- Quais são os principais objetivos da validação e verificação de software?
- É possível garantir que um sistema esteja totalmente livre de erros? Justifique a sua resposta.

Questionário

- No que consiste a depuração de software? Que erros comumente são encontrados nesta atividade?
- O que é um plano de teste?
- O que é *cleanroom*?
- Qual é a diferença entre teste caixa branca e teste caixa preta?

Re-engenharia

- Reorganização e modificação dos software já existentes para torná-los mais manuteníveis.
- Reestruturar ou reescrever parte ou todos de um sistema legado sem mudar as suas funcionalidades.
- Aplicável onde alguns, mas não todos, sub-sistemas de um grande sistema requerem frequente manutenção.

Re-engenharia

- A re-engenharia envolve adicionar esforços para tornar mais fácil a manutenção de sistemas. O sistema pode ser reestruturado ou redocumentado.

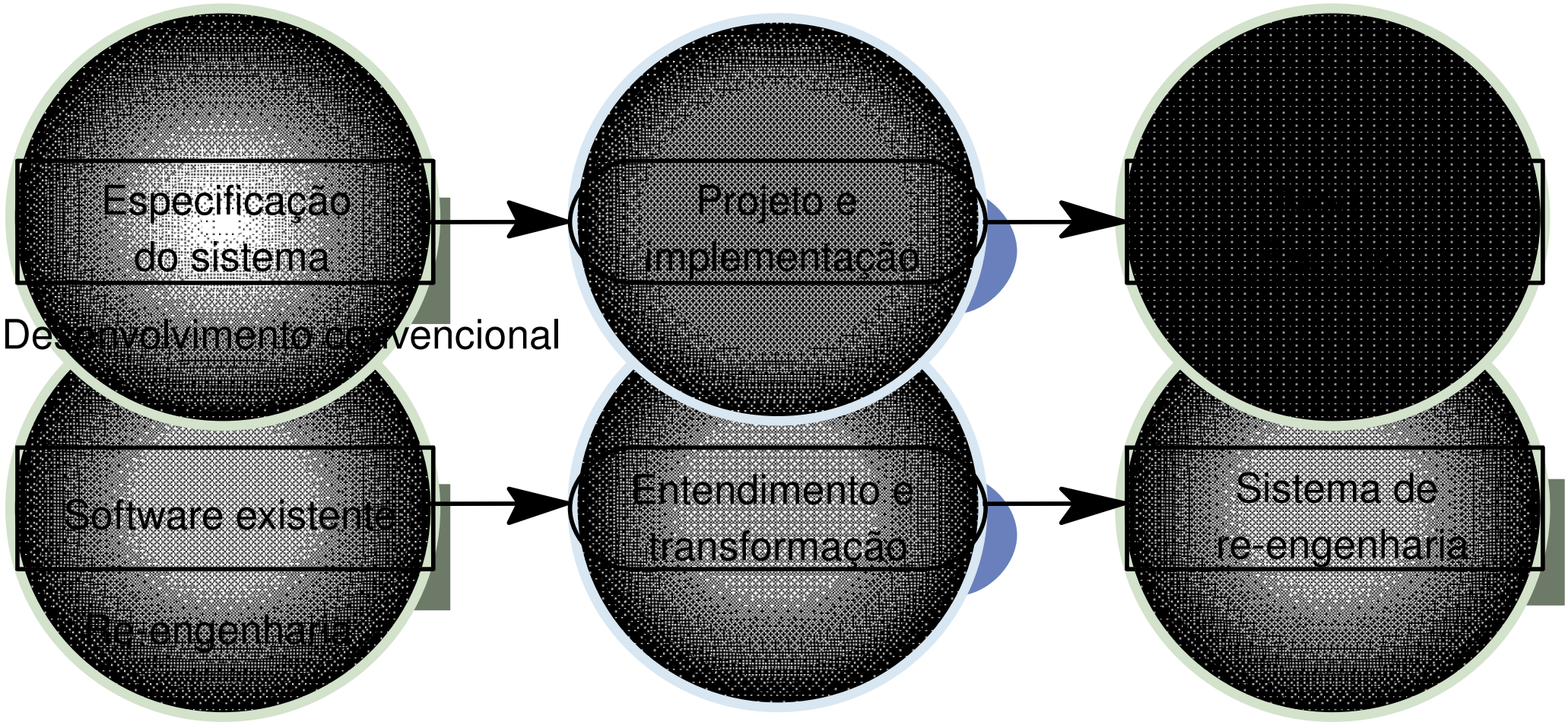
Quando usar a re-engenharia?

- Quando as mudanças do sistema se referem a uma parte do produto.
- Quando o hardware ou software se tornam obsoletos.
- Quando as ferramentas para realização da re-estrutura estão disponíveis.

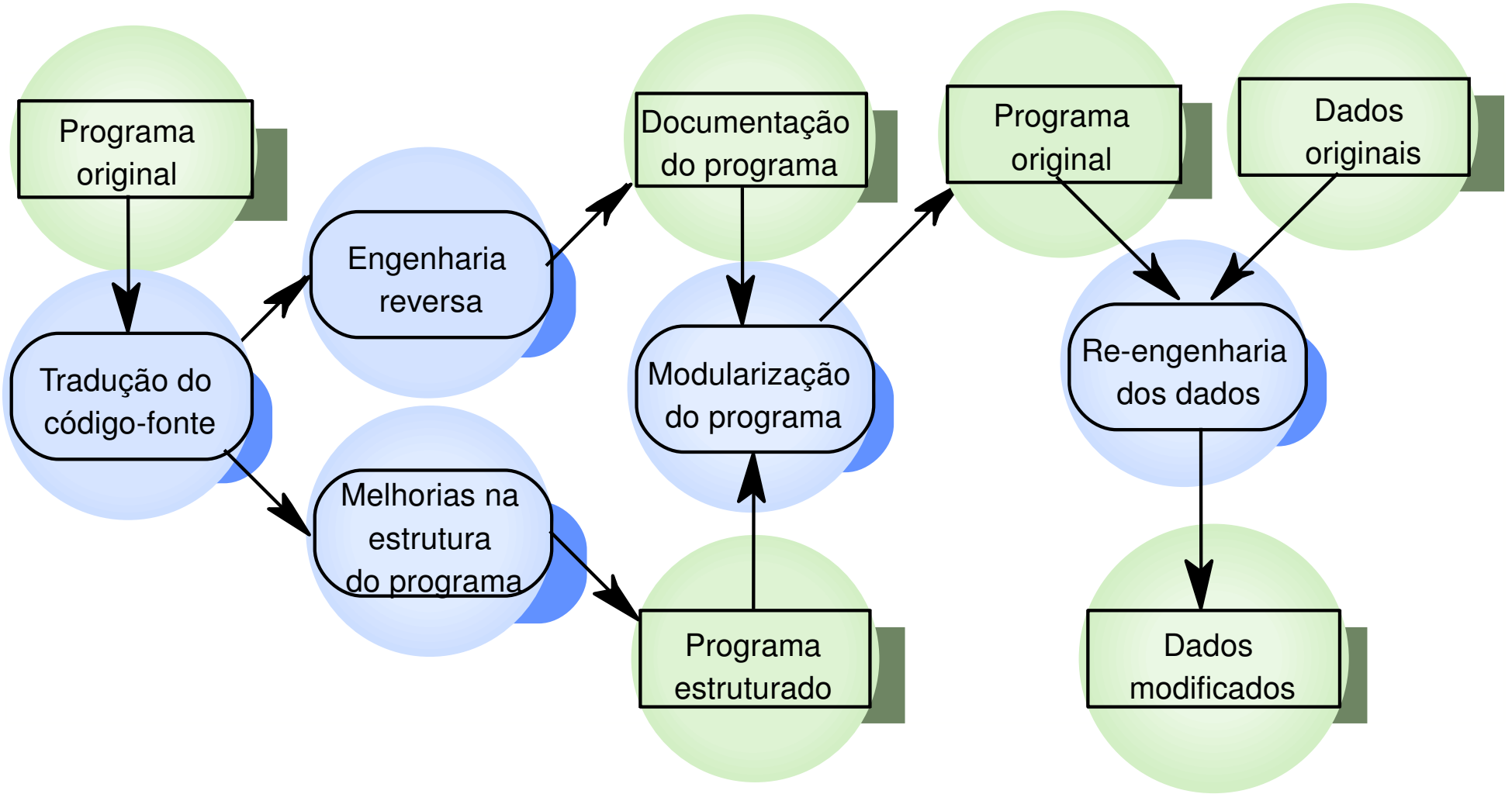
Vantagens

- Risco reduzido
  - Existe um alto risco no desenvolvimento de um novo software.
- Custo reduzido
  - O custo de re-engenharia é significativamente menor do que o custo de desenvolver um novo software.

Comparação


Especificação do sistema
Projeto e implementação
Desenvolvimento convencional
Software existente
Entendimento e transformação
Sistema de re-engenharia
Re-engenharia

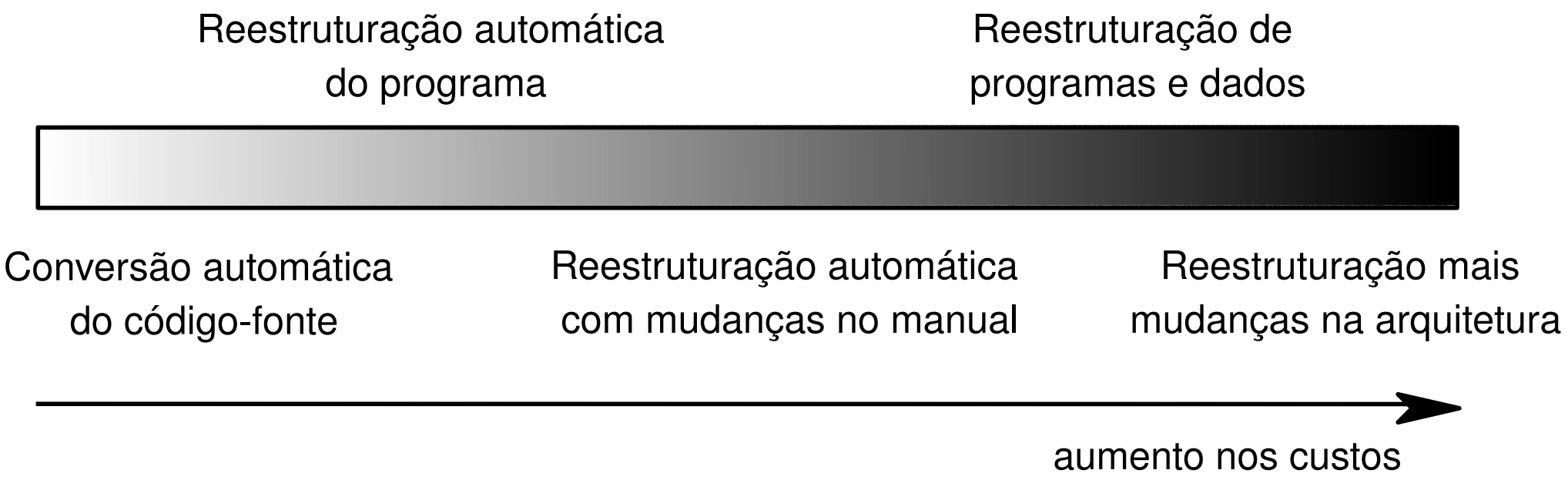
Processo de Re-engenharia



Fatores de custo

- A qualidade do software que passará pelo processo de re-engenharia.
- Suporte de ferramentas disponíveis para re-engenharia.
- A extensão dos dados que precisarão ser convertidos.
- A disponibilidade de especialistas para liderar o processo.

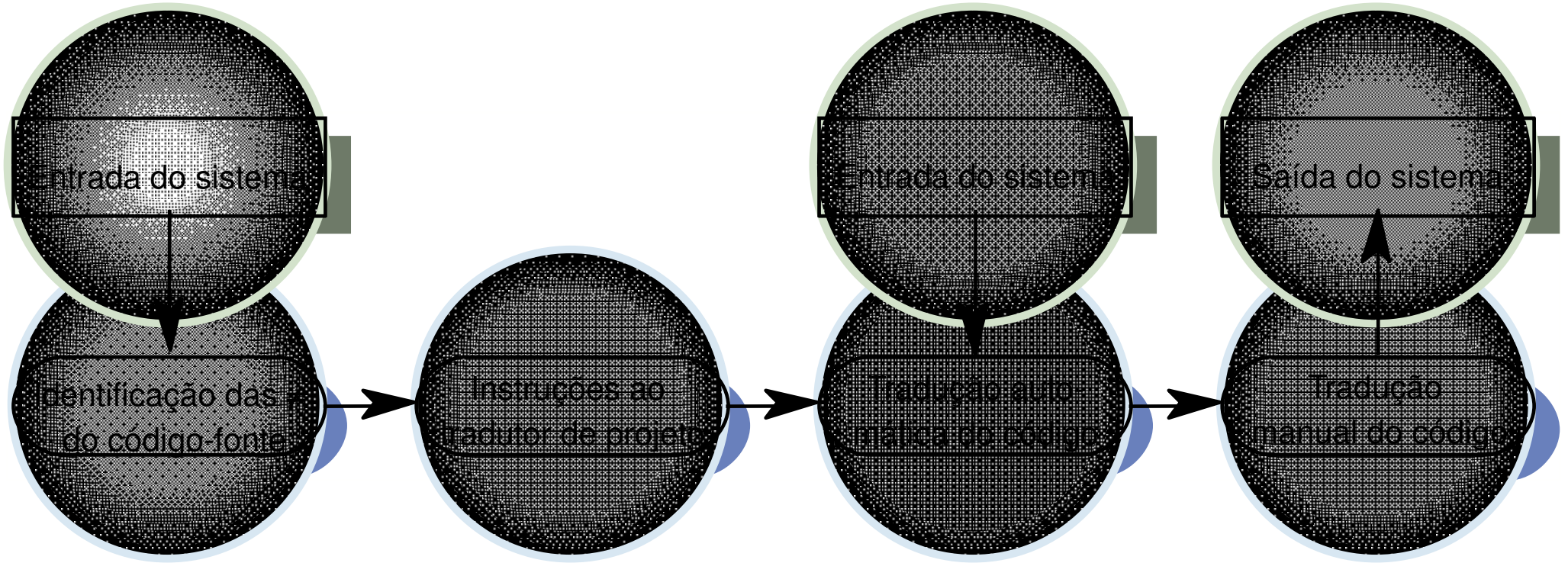
Avaliação de Custos



Tradução do Código-Fonte

- Conversão de uma linguagem de programação para outra.
- Isso pode ser em razão de:
  - Atualização da plataforma de hardware.
  - Pela escassez de profissionais especializados.
  - Por mudanças políticas na organização.
- Somente se realmente existir um tradutor disponível.

Processo de Tradução de Programas



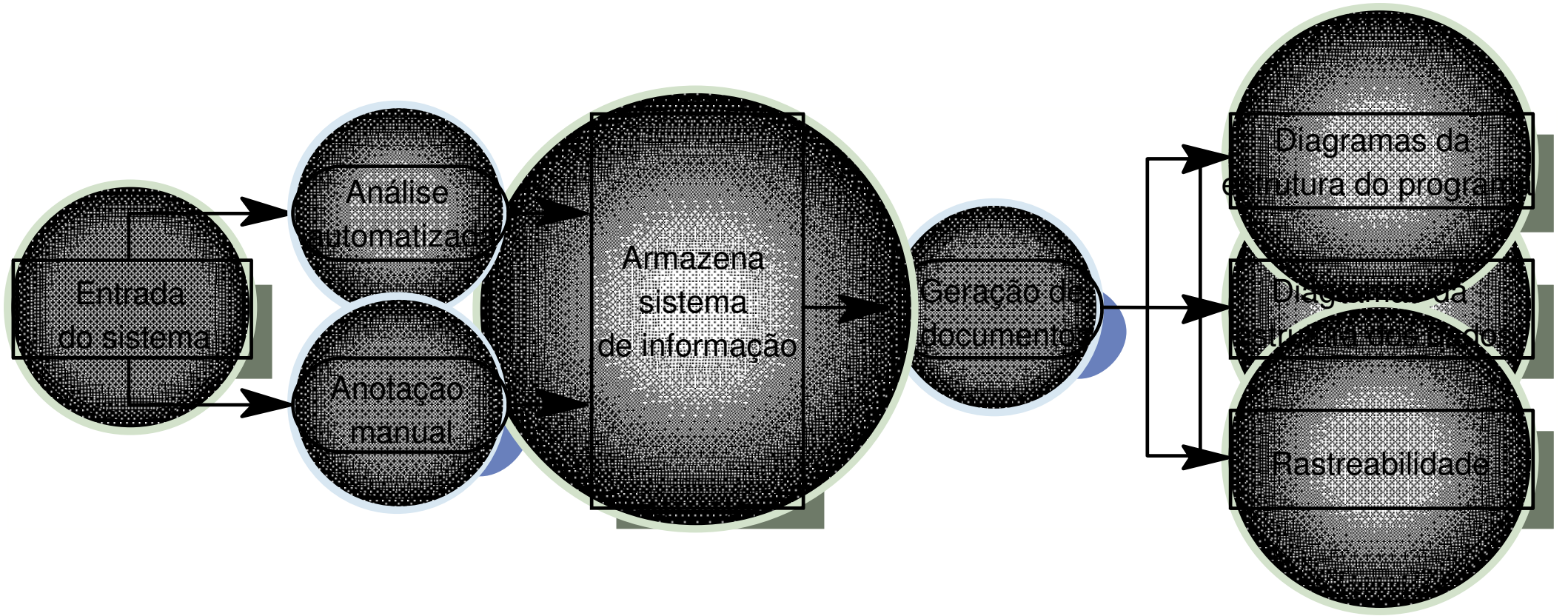
Engenharia Reversa

- Analisar o software com o objetivo de entender o seu projeto e especificação.
- Pode ser parte de um processo de re-engenharia, mas também pode ser utilizado para geração de uma nova especificação de um sistema para uma nova implementação.

Engenharia Reversa

- Construção do banco de dados do programa e gera a informação dele.
- Ferramentas para entendimento dos programas podem ser utilizadas neste processo.

Processo de Engenharia Reversa



Questionário

- Qual é a diferença entre re-engenharia e engenharia reversa?
- Comente duas vantagens no uso da re-engenharia.
- Explique como funciona o processo de engenharia reversa.

eXtreme Programming

- Metodologia ágil para equipes pequenas e médias e que irão desenvolver software com requisitos vagos ou que mudam com freqüência.
- A codificação é a atividade principal.
- Ênfase
  - menor em processos formais de desenvolvimento.
  - maior na disciplina rigorosa.

eXtreme Programming

- Manifesto ágil
  - Valorização:
    - Indivíduos e interação mais que processos e ferramentas.
    - Software em funcionamento mais que documentação abrangente.
    - Colaboração com o cliente mais que negociação de contratos.
    - Responder a mudanças mais que seguir um plano.

eXtreme Programming

- Manifesto ágil
  - Principais preocupações:
    - Entregar funcionalidades para o cliente de forma rápida.
    - Procuram usar o mínimo de documentação possível.
    - Realizam projetos simples sem se preocupar em antecipar as funcionalidades.

eXtreme Programming

- Idéia
  - Baseia-se na revisão permanente do código, testes freqüentes, participação do usuário final, refatoramento contínuo, integração contínua, planejamento, design e redesign a qualquer hora.

eXtreme Programming

- Quatro variáveis
  - 3 controláveis: escopo, qualidade e recursos.
  - 1 determinada pela dinâmica do desenvolvimento: tempo.

eXtreme Programming

- Princípios
  - Comunicação
  - Simplicidade
  - *Feedback*
  - Coragem

eXtreme Programming

- Comunicação
  - Várias práticas do XP promovem uma maior comunicação entre os membros da equipe.
  - A comunicação não é limitada por procedimentos formais. Usa-se o melhor meio possível, por exemplo:
    - Uma conversa ou reunião informal.
    - E-mail, bate-papo ou telefonema.
    - Diagramas.
    - O próprio código.
    - Estórias do usuário.

eXtreme Programming

- Comunicação
  - Preferência à comunicação mais ágil
    - Telefonema é melhor do que e-mail.
    - Presença física é melhor do que remota.
    - Código auto-explicativo é melhor do que documentação escrita.

eXtreme Programming

- Simplicidade
  - O XP incentiva as práticas que reduzem a complexidade do sistema.
  - A solução adotada deve ser sempre a mais simples para se alcançar os objetivos esperados:
    - Usar as tecnologias, *design*, algoritmos e técnicas mais simples que permitirão atender aos requisitos do usuário final.
    - *Design*, processo e código podem ser simplificados a qualquer momento.
    - Qualquer *design*, processo ou código criado pensando nas iterações futuras deve ser descartado.

eXtreme Programming

- *Feedback*
  - Várias práticas do XP garantem um rápido *feedback* sobre as várias etapas/fases do processo.
    - *Feedback* sobre qualidade do código (testes de unidade, programação em pares, posse coletiva).
    - *Feedback* sobre estado do desenvolvimento (estórias do usuário final, integração contínua, jogo do planejamento).

eXtreme Programming

- *Feedback*
  - Permite maior agilidade
    - Erros detectados e corrigidos imediatamente.
    - Requisitos e prazos reavaliados mais cedo.
    - Facilita a tomada de decisões.
    - Permite estimativas mais precisas.
    - Maior segurança e menos riscos para investidores.

eXtreme Programming

- Coragem
  - Testes, integração contínua, programação em pares e outras práticas do XP aumentam a confiança do programador e ajudam-no a ter coragem para:
    - Melhorar o *design* de código que está funcionando para torná-lo mais simples.
    - Jogar fora o código desnecessário.
    - Investir tempo no desenvolvimento de testes.

eXtreme Programming

- Coragem
  - Mexer no *design* em estágio avançado do projeto.
  - Pedir ajudar aos que sabem mais.
  - Dizer ao cliente que um requisito não vai ser implementado no prazo prometido.
  - Abandonar processos formais e fazer *design* e documentação em forma de código.

Práticas XP

- A equipe
  - Todos em um projeto XP são parte de uma equipe.
  - Esta equipe deve incluir um representante do cliente, que:
    - estabelece os requerimentos do projeto.
    - define as prioridades.
    - controla o rumo do projeto.

Práticas XP

- A equipe
  - O representante é usuário final que conhece o domínio do problema e suas necessidades.

Práticas XP

- A equipe
  - Outros papéis assumidos pelos integrantes da equipe:
    - programadores.
    - testadores (que ajudam o cliente com testes de aceitação).
    - analistas (que ajudam o cliente a definir requerimentos).
    - gerente (garante os recursos necessários).
    - *coach* (orienta a equipe, controla a aplicação do XP).
    - *tracker* (coleta métricas).

Práticas XP

- Jogo de planejamento (*planning game*)
  - Prática XP na qual se define
    - estimativas de prazo para cada tarefa.
    - as prioridades: quais as tarefas mais importantes.

Práticas XP

- Jogo de planejamento (*planning game*)
  - Dois passos chave:
    - Planejamento de um *release*
      - Cliente propõe funcionalidade desejadas (estórias).
      - Programadores avaliam a dificuldade de implementá-las.
    - Planejamento de um iteração (de duas semanas)
      - Cliente define as funcionalidades prioritárias para a iteração.
      - Programadores as quebram em tarefas e avaliam o seu custo.

Práticas XP

- Jogo de planejamento (*planning game*)
  - Ótimo *feedback* para que o cliente possa dirigir o projeto
    - É possível ter uma idéia clara do avanço do projeto.
    - Clareza reduz riscos, aumenta a chance de sucesso.

Práticas XP

- Testes de aceitação
  - No jogo de planejamento, o usuário-cliente elabora estórias que descrevem cada funcionalidade desejada. Programador as implementa
    - Cada estória deve ser entendida suficientemente bem para que os programadores possam estimar sua dificuldade.
    - Cada estória deve ser testável.

Práticas XP

- Testes de aceitação
  - Testes de aceitação são elaborados pelo cliente
    - São testes automáticos.
    - Quando rodarem com sucesso, funcionalidade foi implementada.
    - Devem ser rodados novamente em cada iteração futura.
    - Oferecem *feedback*: pode-se saber, a qualquer momento, qual a porcentagem do sistema já foi implementada e quanto falta.

Práticas XP

- Pequenos lançamentos (*small releases*)
  - Disponibiliza, a cada iteração, software 100% funcional
    - Benefícios do desenvolvimento disponíveis imediatamente.
    - Menor risco (se o projeto não terminar, parte existe e funciona).
    - Cliente pode medir com precisão do quanto já foi feito.
    - *Feedback* do cliente permitirá que problemas sejam detectados cedo e facilita a comunicação entre o cliente e o desenvolvimento.

Práticas XP

- Pequenos lançamentos (*small releases*)
  - Cada lançamento possui funcionalidades prioritárias
    - Valores de negócio implementados foram escolhidos pelo cliente.
  - Lançamento pode ser destinado a:
    - usuário-cliente (que pode testá-lo, avaliá-lo e oferecer *feedback*).
    - usuário-final (sempre que possível).

Práticas XP

- Pequenos lançamentos (*small releases*)
  - *Design* simples e integração contínua são práticas essenciais para viabilizar pequenos lançamentos freqüentes.

Práticas XP

- Design simples
  - O *design* está presente em todas as etapas do XP
    - O projeto começa simples e se mantém simples através de testes e refinamento do *design* (refatoramento).
  - Todos buscamos *design* simples e claro. Em XP, isto é levado a níveis extremos.
    - Não permitimos que se implemente nenhuma função adicional que não será usada na atual iteração.

Práticas XP

- *Design* simples
  - Implementação ideal é aquela que:
    - Roda todos os testes.
    - Expressa todas as idéias que você deseja expressar.
    - Não contém código duplicado.
    - Tem o mínimo de classes e métodos.
  - O que não é necessário AGORA não deve ser implementado.

Práticas XP

- Programação em pares
  - Todo o desenvolvimento em XP é feito em pares
    - Um computador, um teclado e dois programadores.
    - Um piloto e um co-piloto.
    - Papéis são alternados freqüentemente.
    - Pares são trocados periodicamente.

Práticas XP

- Programação em pares
  - Benefícios
    - Melhor qualidade do *design*, código e testes.
    - Revisão constante do código.
    - Nivelamento da equipe.
    - Maior comunicação.

Práticas XP

- Testes
  - O seu desenvolvimento é guiado por testes
  - "*Test first, then code*"
    - Os testes puxam o desenvolvimento.
    - Os programadores XP escrevem testes primeiro, escrevem código e rodam testes para validar o código escrito.
    - Cada unidade de código só tem valor se seu teste funcionar 100%.

Práticas XP

- Testes
  - "*Test first, then code*"
    - Todos os testes são executados automaticamente, o tempo todo.
    - Testes são a documentação executável do sistema.

Práticas XP

- Testes
  - Testes dão maior segurança: coragem para mudar
    - Que adianta a OO isolar a interface da implementação se o programador tem medo de mudar a implementação?
    - Código testado é mais confiável.
    - Código testado pode ser alterado sem medo.

Práticas XP

- Refinamento do *design* (refatoramento)
  - Não existe uma etapa isolada de design em XP
    - O código é o *design*.
  - O *design* é melhorado continuamente através de refatoramento.

Práticas XP

- Refinamento do *design* (refatoramento)
  - O refatoramento é um processo formal realizado através de etapas reversíveis
    - Passos de refatoramento melhoram, incrementalmente, a estrutura do código, sem alterar sua função.
    - Existência prévia de testes é essencial (elimina o medo de que o sistema irá deixar de funcionar por causa da mudança).

Práticas XP

- Integração contínua
  - Projetos XP mantêm o sistema integrado o tempo todo
    - Integração de todo o sistema pode ocorrer várias vezes ao dia.
    - Todos os testes (unidade e integração) devem ser executados.
  - Integração contínua reduz o tempo passado no inferno da integração.

Práticas XP

- Integração contínua
  - Benefícios
    - Expõe o estado atual do desenvolvimento (viabiliza lançamentos pequenos e freqüentes).
    - Estimula *design* simples, tarefas curtas e agilidade.
    - Oferece *feedback* sobre todo o sistema.
    - Permite encontrar problemas de *design* rapidamente.

Práticas XP

- Posse coletiva
  - Em um projeto XP, qualquer dupla de programadores pode melhorar o sistema a qualquer momento.
  - Todo o código em XP pertence a um único dono: a equipe
    - Todo o código recebe a atenção de todos os participantes resultando em uma maior comunicação.
    - Maior qualidade (menos duplicação, maior coesão).
    - Menos riscos e menos dependência de indivíduos.

Práticas XP

- Posse coletiva
  - Todos compartilham a responsabilidade pelas alterações.
  - Testes e integração contínua são essenciais e dão segurança aos desenvolvedores.
  - Programação em pares reduz o risco de danos.

Práticas XP

- Padrões de codificação
  - O código escrito em projetos XP segue um padrão de codificação, definido pela equipe:
    - Padrão para nomes de métodos, classes e variáveis.
    - Organização do código.
  - Todo o código parece que foi escrito por um único indivíduo, competente e organizado.

Práticas XP

- Padrões de codificação
  - Código com estrutura familiar facilita e estimula
    - Posse coletiva.
    - Comunicação mais eficiente.
    - Simplicidade.
    - Programação em pares.
    - Refinamento do *design.*

Práticas XP

- Metáfora
  - Equipes XP mantêm uma visão compartilhada do funcionamento do sistema.
    - Pode ser uma analogia com algum outro sistema que facilite a comunicação entre os membros da equipe e cliente.
  - Facilita a escolha dos nomes de métodos, classes, campos de dados etc
    - Serve de base para estabelecimento de padrões de codificação.

Práticas XP

- Ritmo saudável
  - Projetos XP estão na arena para ganhar
    - Entregar software da melhor qualidade.
    - Obter a maior produtividade dos programadores.
    - Obter a satisfação do cliente.

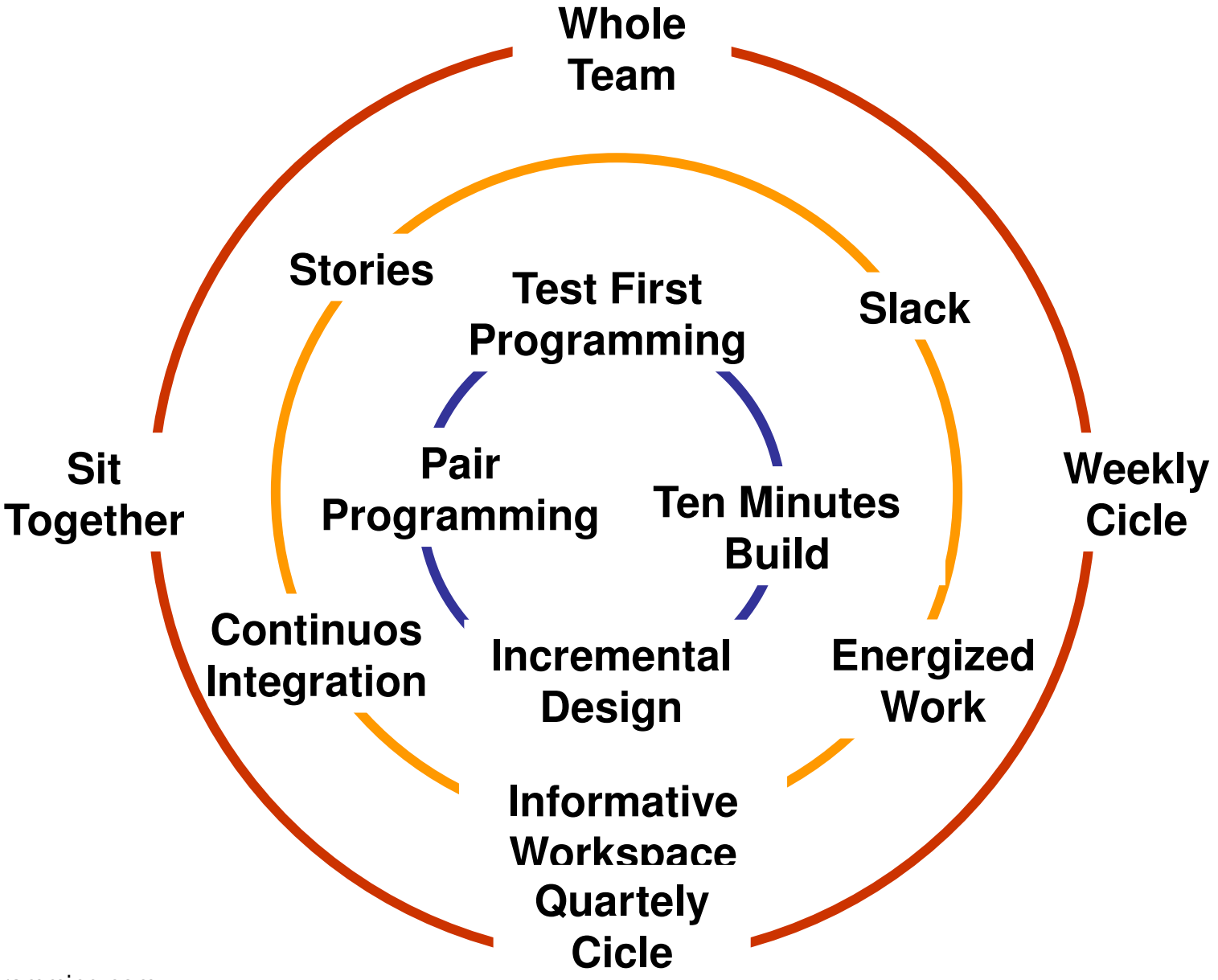
Práticas XP

- Ritmo saudável
  - Projetos com cronogramas apertados que sugam todas as energias dos programadores não são projetos XP
    - Semanas de 80 horas levam à baixa produtividade.
    - Produtividade baixa leva a código ruim, relaxamento da disciplina, dificulta comunicação, aumenta a irritação e o *stress* da equipe.
    - Tempo ganho será perdido depois.

Práticas XP

- Ritmo saudável
  - Projeto deve ter ritmo sustentável por prazos longos.
    - Eventuais horas extras são aceitáveis quando produtividade é maximizada no longo prazo.

Práticas em XP



Adaptado de xprogramming.com

Dificuldades em XP

- Vencer barreiras culturais
  - Deixar alguém mexer no seu código.
  - Trabalhar em pares
    - Ter coragem de admitir que não sabe.
    - Pedir ajuda.

Dificuldades em XP

- Vencer barreiras culturais
  - Vencer hábitos antigos
    - Manter as coisas simples.
    - Jogar fora código desnecessário.
    - Escrever testes antes de codificar.
    - Refatorar com freqüência (vencer o medo).

Quando não usar XP

- Equipes grandes e espalhadas geograficamente
  - Comunicação é um valor fundamental do XP.
  - Não é fácil garantir o nível de comunicação requerido em projetos XP em grandes equipes.
- Situações onde não se tem controle sobre o código
  - Código legado que não pode ser modificado.

Quando não usar XP

- Situações onde o *feedback* é demorado
  - *compile-link-build-test* que leva 24 horas.
  - Testes são muito difíceis, arriscados e que levam muito tempo.
  - Programadores espalhados em ambientes físicos distantes e sem comunicação eficiente.

Conclusões

- *eXtreme Programming* (XP) é uma metodologia de desenvolvimento de software baseada nos valores simplicidade, comunicação, *feedback* e coragem.
- Para implementar XP não é preciso usar diagramas ou processos formais. É preciso fazer uma equipe se unir em torno de algumas práticas simples, obter *feedback* suficiente e ajustar as práticas para a sua situação particular.

Conclusões

- XP pode ser implementada aos poucos, porém a maior parte das práticas são essenciais.
- Nem todos os projetos são bons candidatos a usar uma metodologia ágil como XP.
- XP é mais adequado a equipes pequenas e médias.

Questionário

- O que você entende por metodologia ágil?
- Qual é a principal atividade em XP?
- Quais são os valores da metodologia XP?
- O que é refatoramento?
- Como funciona a prática de programação em duplas em XP?
- No que consiste as práticas na metodologia XP?
- A metodologia XP se aplica melhor às pequenas ou grandes equipes? Justifique a sua resposta.

Referência Bibliográfica

- [1] Carvalho, Ariadne M.B. & Chiossi, Thelma C. dos Santos. ***Introdução à Engenharia de Software***. Editora da Unicamp, 2001.
- [2] Jalote, Pankaj. ***Integrated Approach To Software Engineering***. 2 ed. Springer-Verlag, 1997.
- [3] Pressman, Roger S. ***Software engineering: a practioner's approach***. 5 ed. McGraw Hill, 2001.
- [4] Sommerville, Ian. ***Software Engineering***. 6 ed. Addison Wesley, 2001.