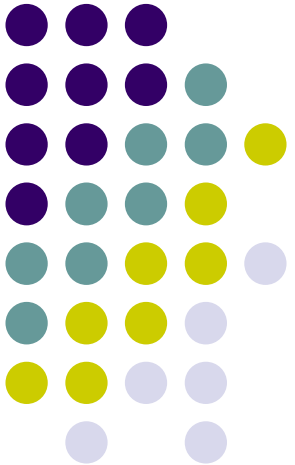
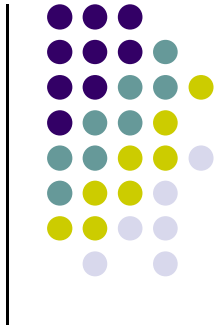
Análise de Projeto de Sistemas I

Módulo I

Processo de Software

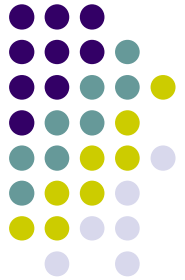


O que é software?



- Programas de computador e toda a documentação associada, como requisitos, modelos de design e manuais.
- Software produtos podem ser desenvolvidos para um cliente particular ou para o mercado em geral.
- Software produtos podem ser genéricos (Unix, Word etc) ou customizados (desenvolvidos so medida).
- Novos softwares podem ser criados através do desenvolvimento de novas aplicações, configurando-se sistemas genéricos ou reusando software existente.

Características do Software

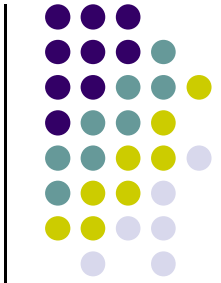


- O software é desenvolvido. Ele não é manufaturado no sentido clássico
  - Qualidade
  - Pessoas
  - Custo
- O software não sofre “desgaste”
- A maioria do software é construído sob medida, em vez de ser montado a partir de componentes pré existentes.

O que é Engenharia de Software?

- A economia de todas as nações desenvolvidas são dependentes de software. Mais e mais sistemas são controlados por software.
- Engenharia de Software é uma disciplina da engenharia que está envolvida com todos os aspectos da produção de software.
- Engenheiros de software devem adotar um enfoque sistemático e organizado para os seus trabalhos e usar ferramentas e técnicas apropriadas, dependendo do problema a ser resolvido, das restrições para o desenvolvimento e dos recursos disponíveis.

O que é um processo de software?

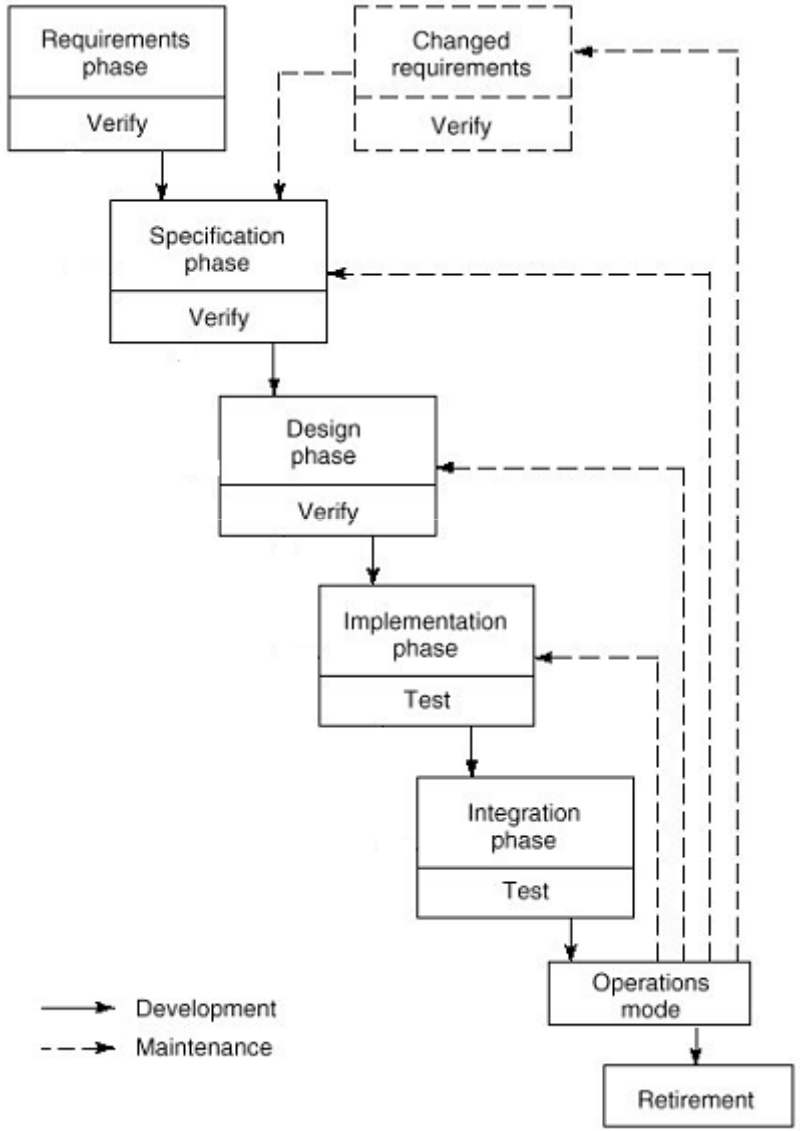


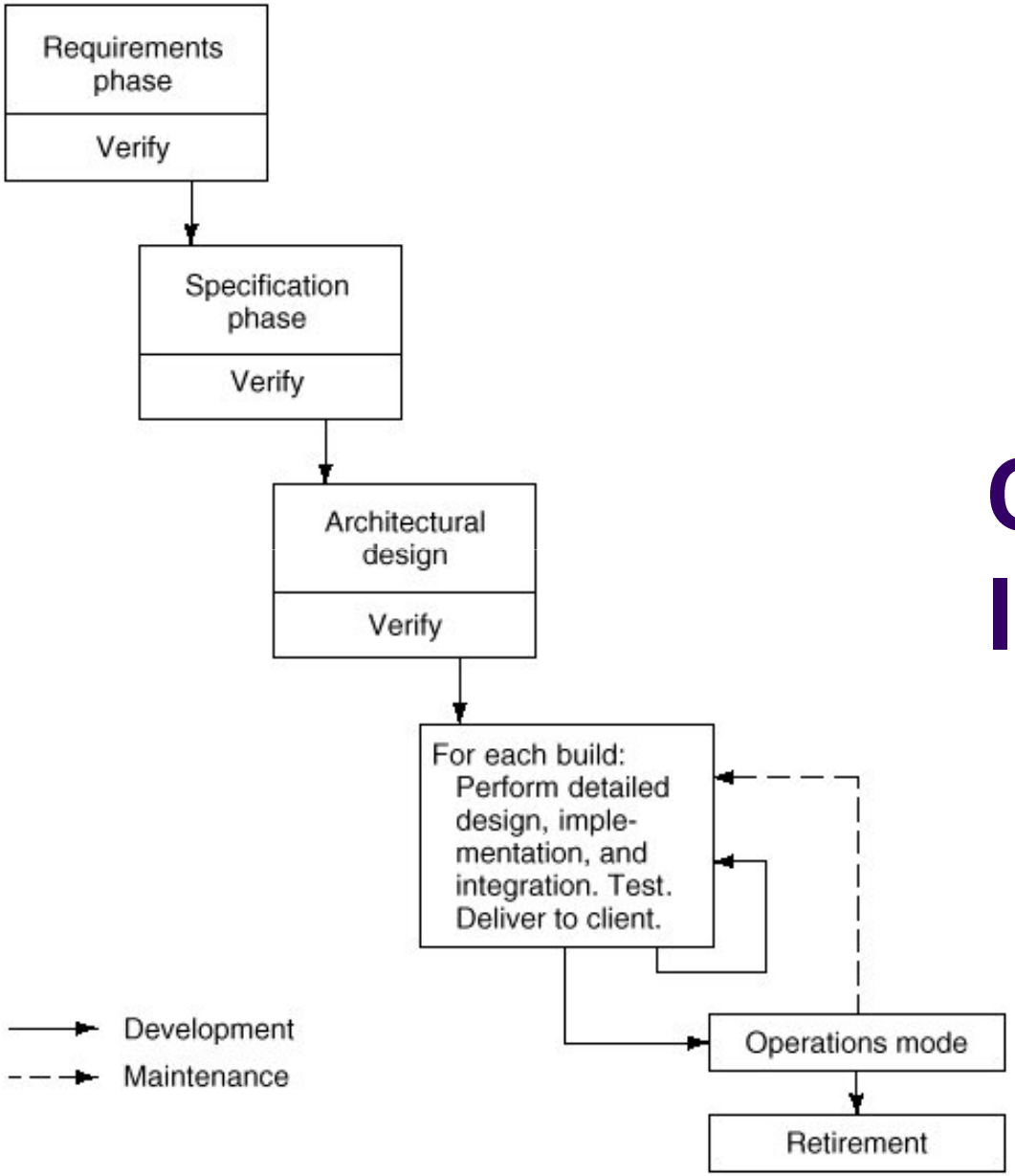
- Um conjunto de atividades cujo objetivo é o desenvolvimento ou evolução do software.
- Atividades genéricas em todos os processos de software são:
  - especificação - o que o sistema deve fazer e quais as restrições de desenvolvimento.
  - desenvolvimento - produção do sistema de software.
  - validação - certificar se o software é o que o cliente quer.
  - evolução - mudar o software em resposta a demandas diversas.

O que é um modelo de processo de software?

- Uma simplificada representação de um processo de software, apresentada de uma perspectiva específica.
- Exemplos de perspectivas de processos são:
  - workflow - sequência de atividades.
  - fluxo de dados - fluxo de informação.
  - papel/ação - quem faz o quê.
- Modelos de processo genéricos:
  - cascata.
  - iterativo.
  - baseado em componente

Ciclo de Vida em Cascata



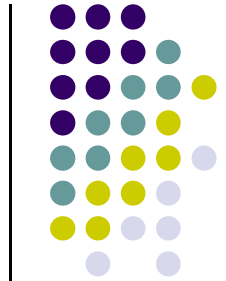


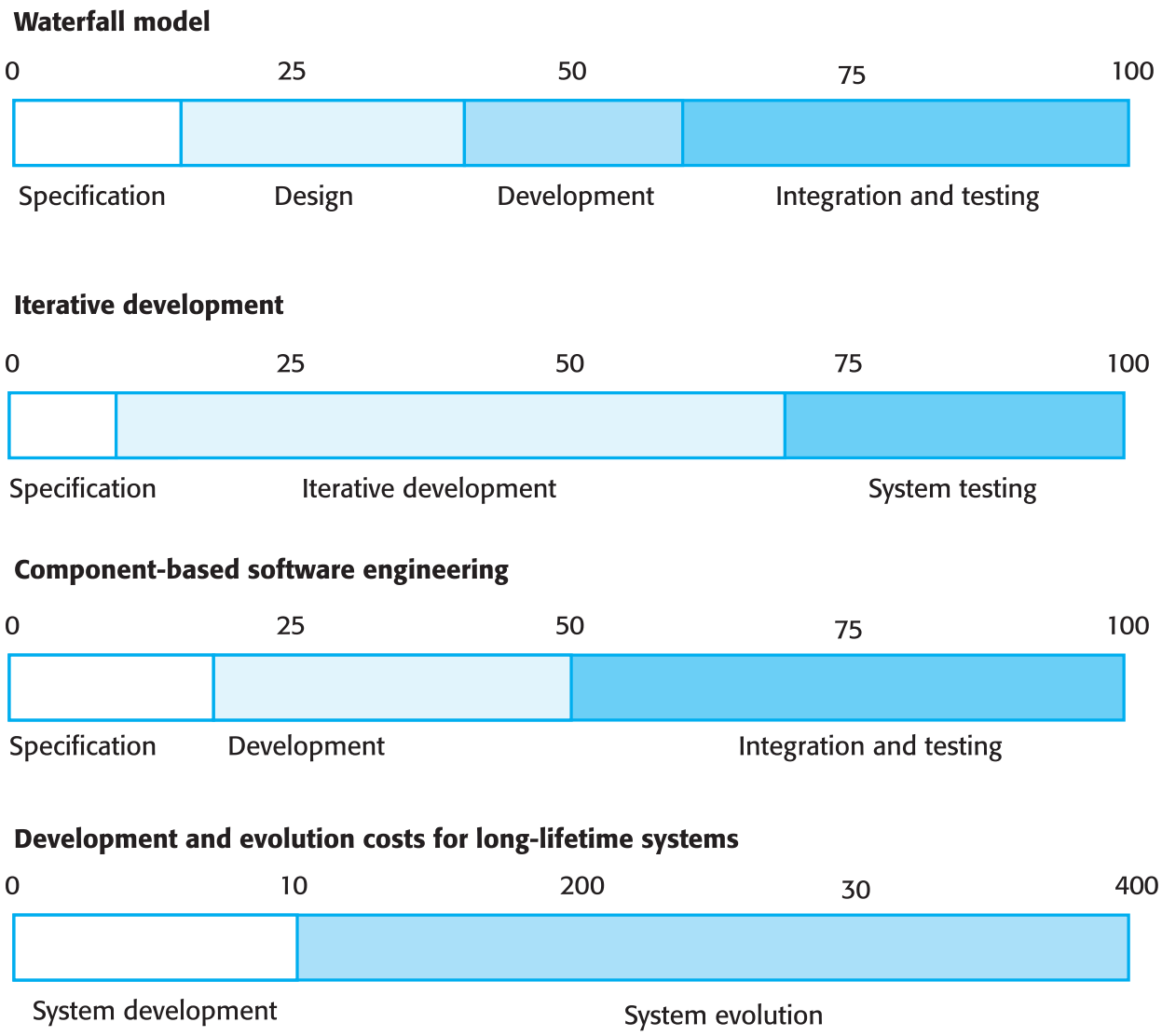
Ciclo de Vida Incremental

Quais são os custos da Engenharia de Software?

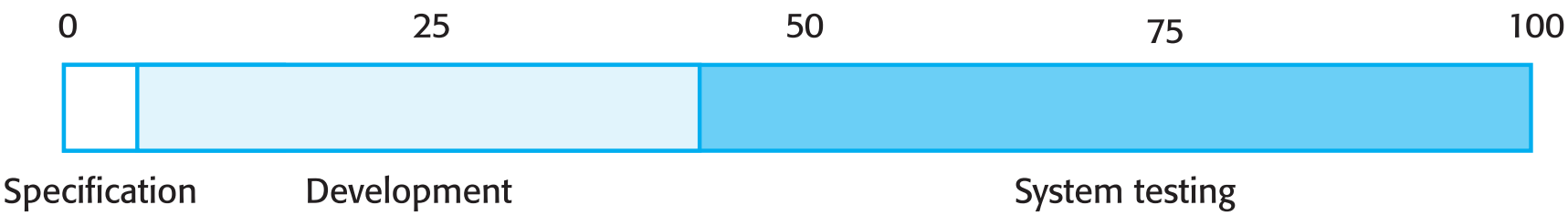
- Aproximadamente, 60% dos custos são custos de desenvolvimento, 40% são custos de testes. Para um software padrão, os custos de evolução sempre excedem os custos de desenvolvimento.
- Custos variam dependendo do tipo de sistema desenvolvido e dos atributos gerais do sistema como performance e confiabilidade.
- A distribuição dos custos depende do modelo de desenvolvimento que é usado.
- Os custos do software geralmente predominam sobre os custos dos sistemas computacionais.
- Os softwares custam mais para manter do que para desenvolver.

Distribuição de Custo por Atividade





Custo de Desenvolvimento do Produto



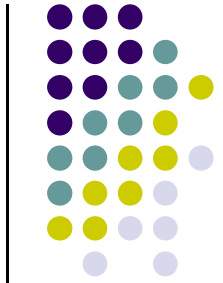
O que é CASE?

- *Computer-Aided Software Engineering*.
- SIstemas de softwares projetados para proporcionar suporte automático para as atividades dos processos de software.
- Sistemas CASE são geralmente usados para o suporte de metodologias.
  - *Upper-CASE* são ferramentas que suportam as atividades iniciais de requisitos e projeto.
  - *Lower-CASE* são ferramentas que suportam atividades subsequentes, como programação, teste e depuração.

Quais são os atributos de um bom software?

- O software deve entregar a funcionalidade e a performance requeridas ao usuário e deve ser manutenível, confiável e aceitável.
  - Manutenibilidade - o software deve evoluir para atender às mudanças de necessidade.
  - Dependabilidade - o software deve ser justificadamente digno de confiança, disponível, seguro, executar sem falhas.
  - Eficiência - o software não deve usar recursos do sistema desnecessariamente.
  - Aceitabilidade - o software deve ser aceito pelos usuários para os quais o mesmo foi projetado. Isto significa que ele deve ser compreensível, ter boa usabilidade e ser compatível com outros sistemas.

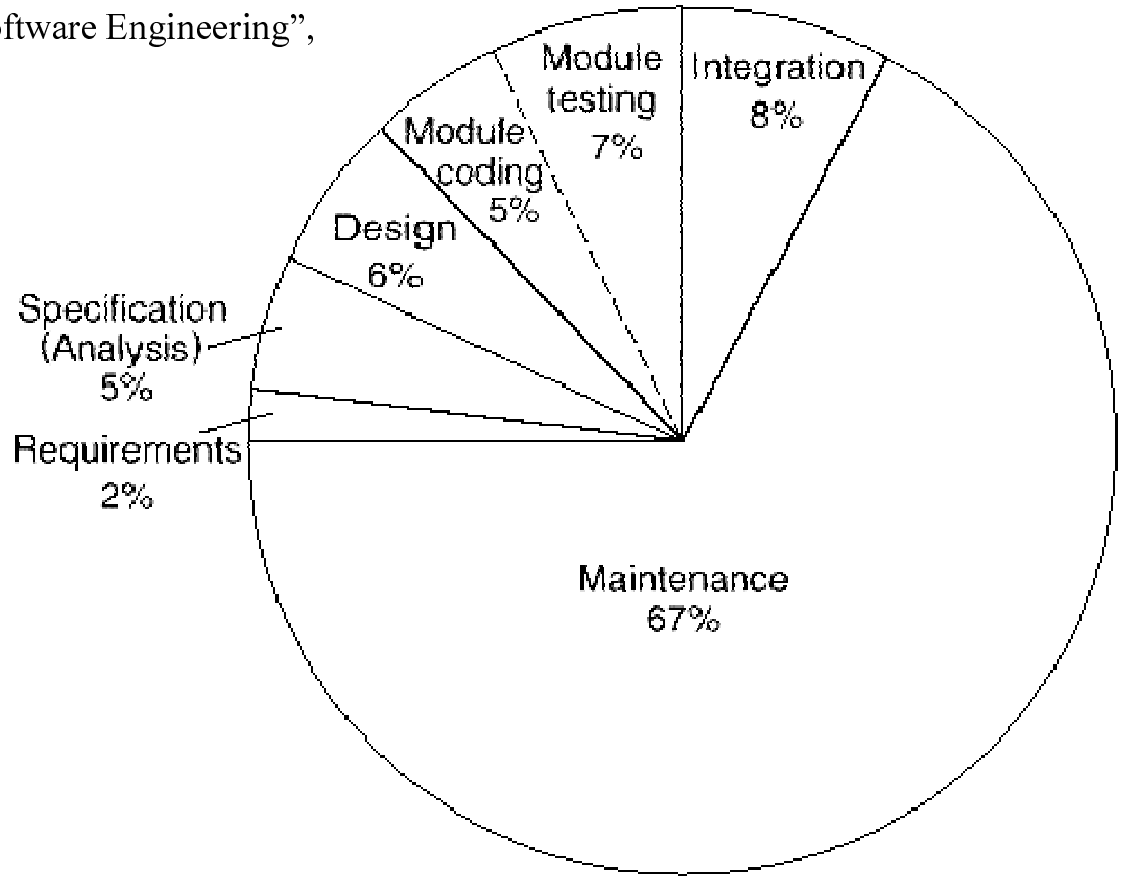
Manutenção de Software



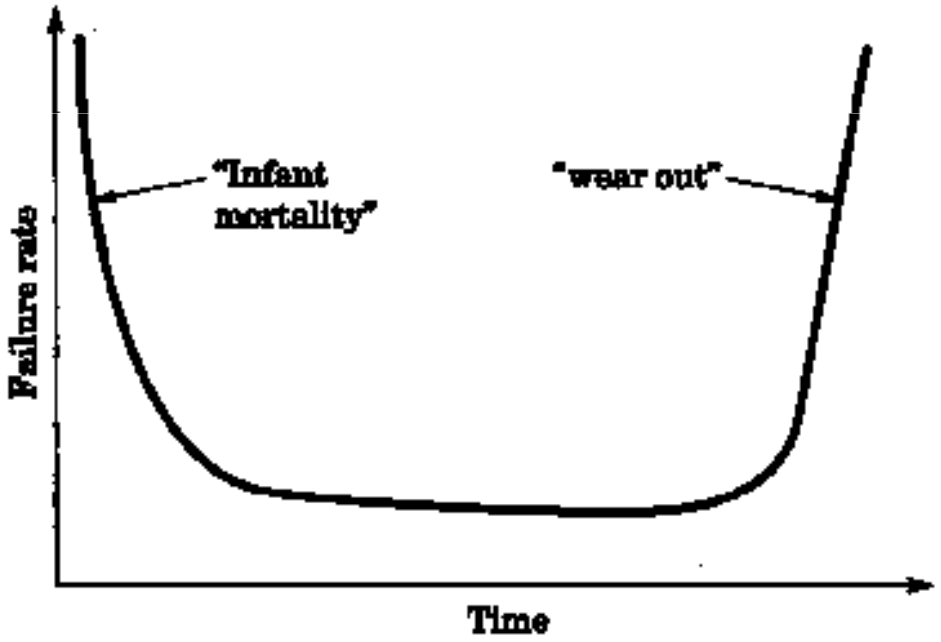
- Corretiva (erros no software)
- Evolutiva (novas características, nova interface etc)
- Preventiva (ex.: bug ano 2000)
- Adaptativas (novas necessidade, mudanças legais etc)

A Importância da Manutenção

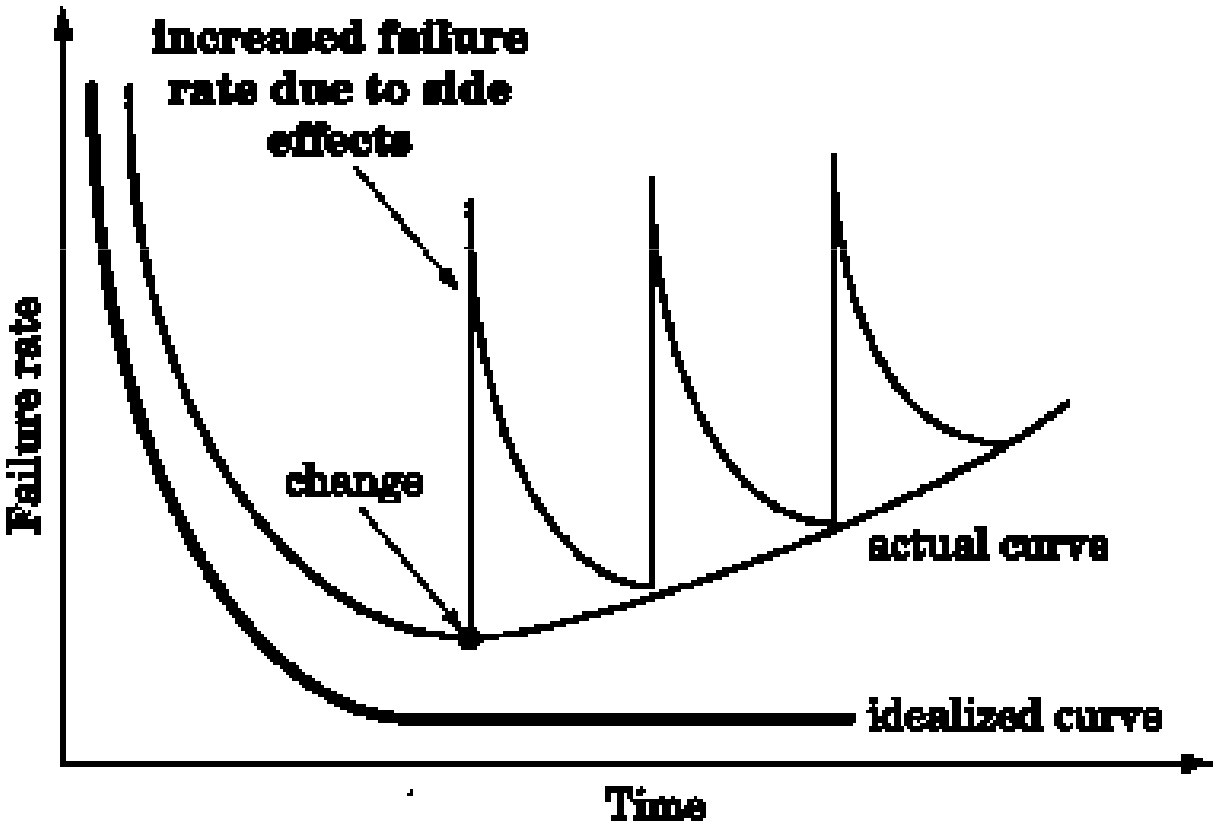
Stephen Schach,
"Classical and Object-oriented Software Engineering",
Mc Graw Hill, 1999

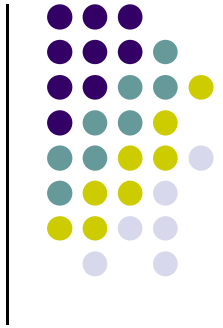


Curva de Erros do Hardware



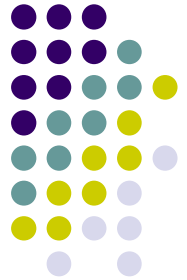
Curva de Erros do Software





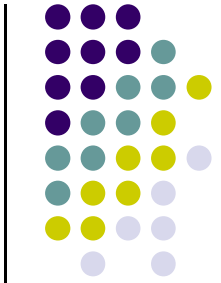
PROCESSO DE SOFTWARE

Processo de Software



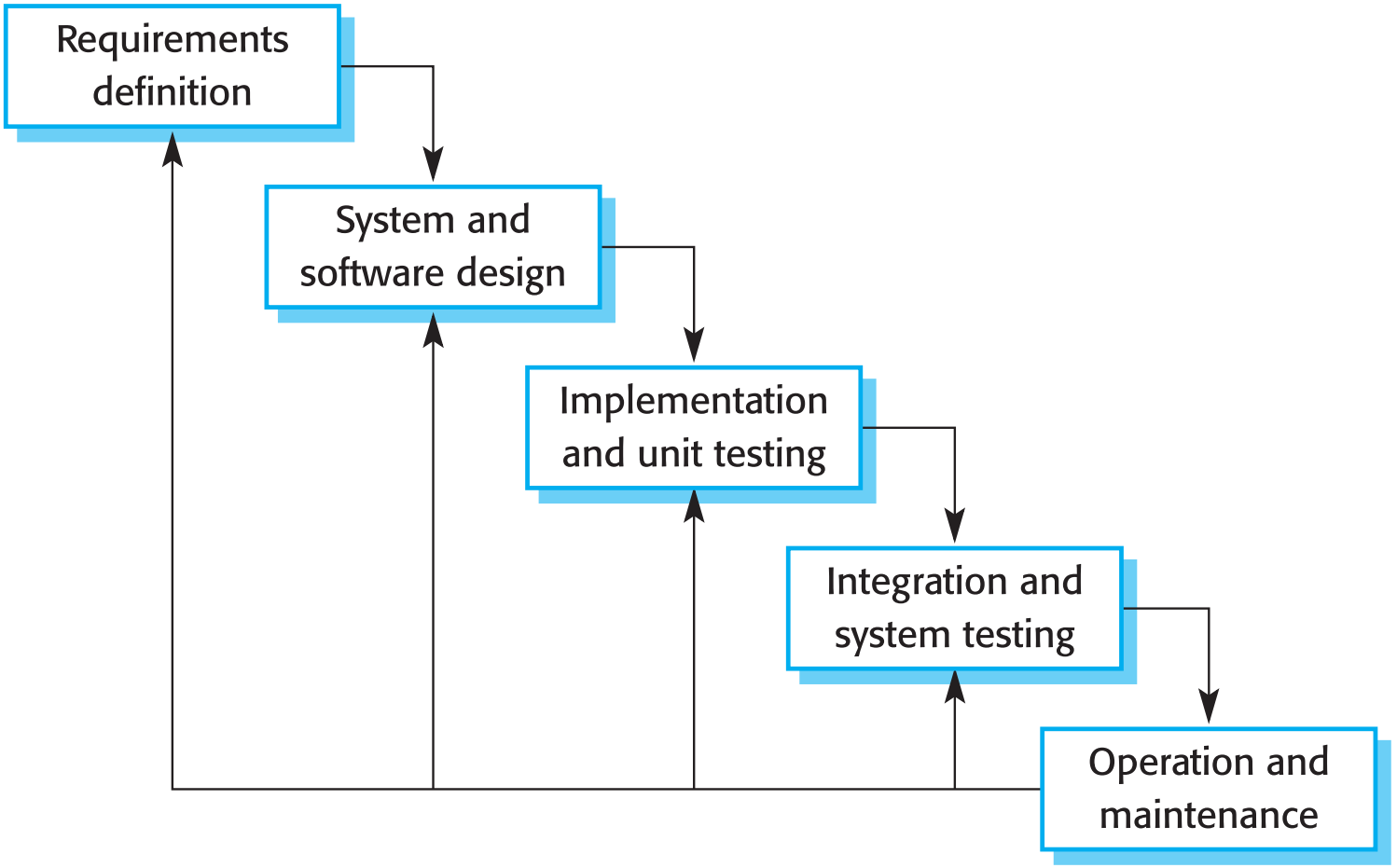
- Um conjunto estruturado de atividades requerido para desenvolver um sistema de software.
  - Especificação
  - Projeto
  - Validação
  - Evolução
- Um modelo de processo de software é uma representação abstrata de um processo. Ele representa a descrição de um processo a partir de uma perspectiva específica.

Modelos Genéricos de Processos de Software



- Modelo Cascata
  - Fases separadas e distintas de especificação e desenvolvimento.
- Desenvolvimento Evolutivo
  - Especificação, Desenvolvimento e Validação são intercalados.
- Engenharia de Software Baseada em Componentes.
  - O sistema é montado a partir de componentes existentes ou desenvolvidos.
- Há muitas variações desses modelos, como por exemplo, o Desenvolvimento Formal, onde um processo similar ao cascata é usado, mas a especificação é uma especificação formal que é refinada por vários estágios até um design implementável.

Modelo em Cascata

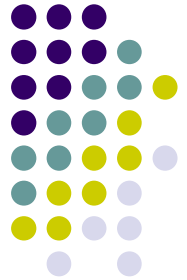


Sommerville, Software Engineering, 7th edition.

Fases do Modelo em Cascata

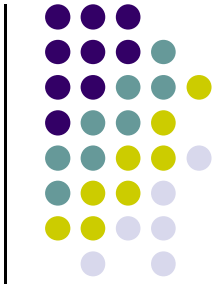
- Análise e Definição de Requisitos
- Projeto do Software e do Sistema
- Implementação e Teste de Unidade
- Integração e Teste de Sistema
- Operação e Manutenção
- O principal inconveniente do modelo em cascata é a dificuldade em acomodar mudanças depois que o processo está em execução. Toda fase deve ser completada antes de iniciar a próxima.

Modelo Cascata - Problemas



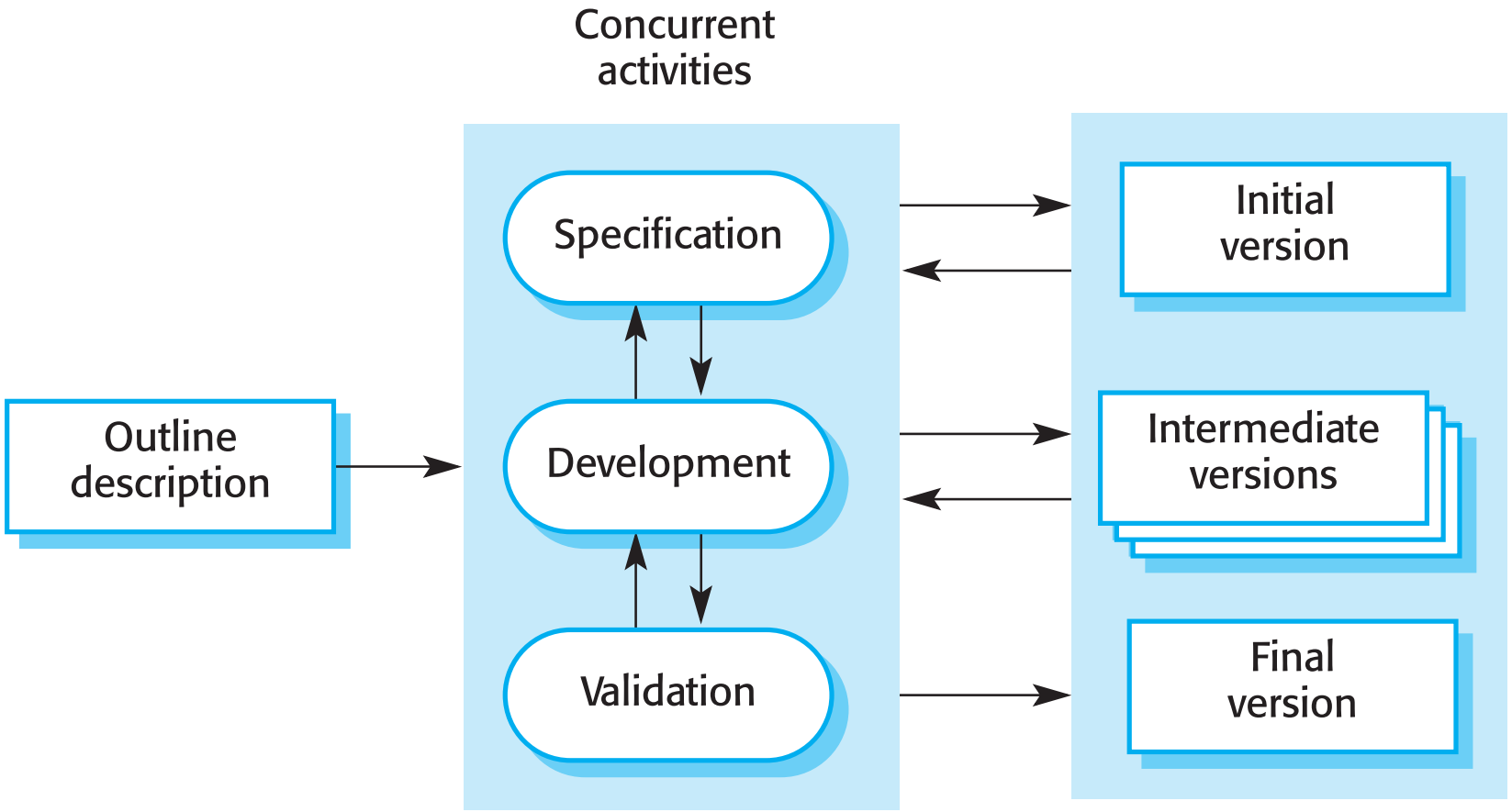
- O particionamento inflexível do projeto em estágios distintos torna difícil responder a mudanças nos requisitos.
- Então este modelo só deve ser usado quando os requisitos são bem compreendidos e mudanças serão moderadamente limitadas durante o processo de design.
- Poucos sistemas de negócio possuem requisitos estáveis.
- O modelo de cascata é mais usado em grande projetos de engenharia de sistemas, onde um sistema é desenvolvido em vários locais.

Desenvolvimento Evolutivo



- Desenvolvimento exploratório
  - O objetivo é trabalhar com o cliente e evoluir para um sistema final a partir de um esboço inicial de especificação. Deve começar com requisitos bem compreendidos e adicionar novas características à medida em que forem propostas pelo cliente.
- Prototipação descartável
  - O objetivo é compreender os requisitos do sistema. Deve começar com baixo entendimento dos requisitos para tornar claro o que é realmente necessário.

Desenvolvimento Evolutivo (2)



Sommerville, Software Engineering, 7th edition.

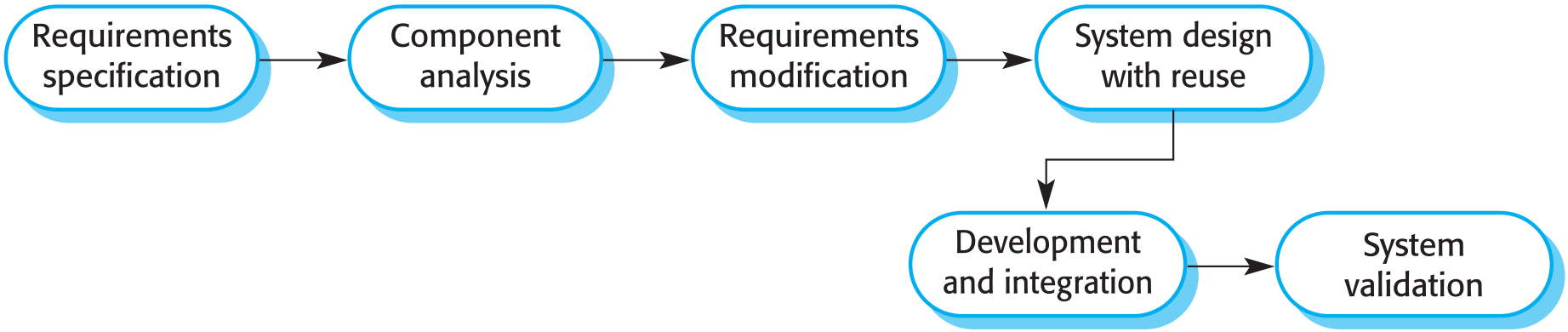
Desenvolvimento Evolutivo (3)

- Problemas
  - Falta de visibilidade do processo.
  - Os sistemas são geralmente pouco estruturados.
  - Habilidades especiais podem ser requeridas (como em linguagens para prototipação rápida).
- Aplicabilidade
  - Para sistemas interativos de pequeno e médio porte.
  - Para partes de grandes sistemas (por exemplo, para a IHC).
  - Para sistemas de ciclo de vida pequeno.

Engenharia de Software Baseada em Componentes

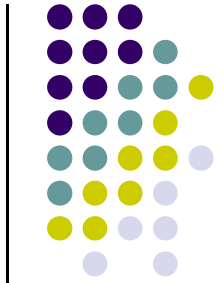
- Baseado em reuso sistemático, em que sistemas são integrados a partir de componentes existentes ou COTS (Commercial-Off-The-Shelf).
- Estágios do processo
  - Análise de componente.
  - Modificação de requisitos.
  - Design do sistema com reuso.
  - Desenvolvimento e Integração.
- Este enfoque está se tornando crescentemente usado, na medida em que padrões de componentes têm emergido.

Desenvolvimento Orientado ao Reuso



Sommerville, Software Engineering, 7th edition.

Iteração

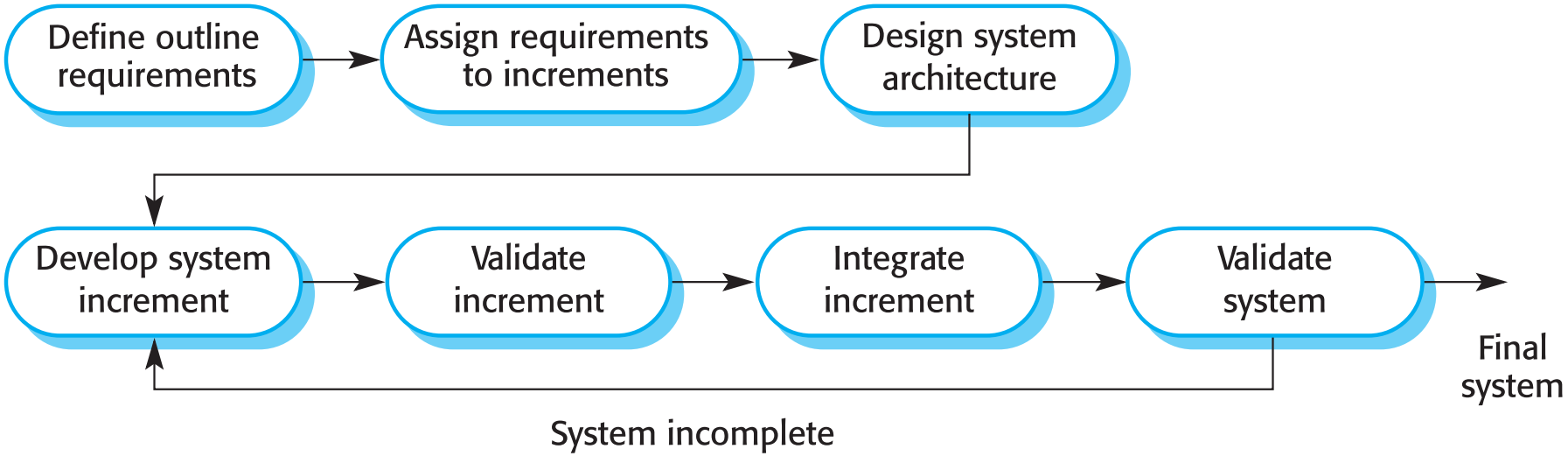


- Os requisitos de um sistema sempre evoluem ao longo de um processo de tal forma que a iteração, onde os estágios iniciais são re-trabalhados, é sempre parte do processo para grandes sistemas.
- A iteração pode ser aplicada para qualquer um dos modelos de processos genéricos.
- Dois enfoques relacionados são
  - Ciclo Incremental
  - Desenvolvimento Espiral

Desenvolvimento Incremental

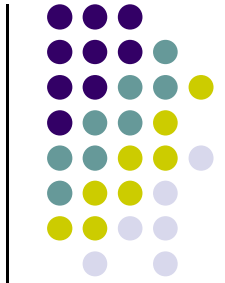
- Em vez de entregar o sistema de uma só vez, o desenvolvimento é particionado em incrementos, com cada incremento entregando parte da funcionalidade requerida.
- Os requisitos são priorizados e os mais importantes são incluídos nos incrementos iniciais.
- Uma vez que o desenvolvimento de um incremento é iniciado os requisitos são congelados, embora os requisitos para incrementos posteriores podem continuar a evoluir.

Desenvolvimento Incremental (2)



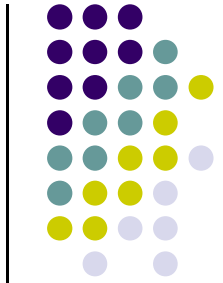
Sommerville, Software Engineering, 7th edition.

Vantagens do Desenvolvimento Incremental



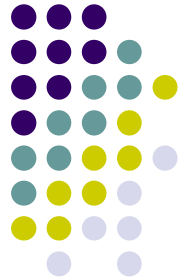
- Necessidades do cliente podem ser entregues em cada incremento, de tal forma que funcionalidades do sistema estão disponíveis cedo - feedback.
- Incrementos iniciais atuam como protótipos para auxiliar a elicitação dos requisitos para os incrementos futuros.
- Riscos mais baixos de fracasso do projeto.
- Os serviços mais prioritários tendem a ser mais testados.

Extreme Programming



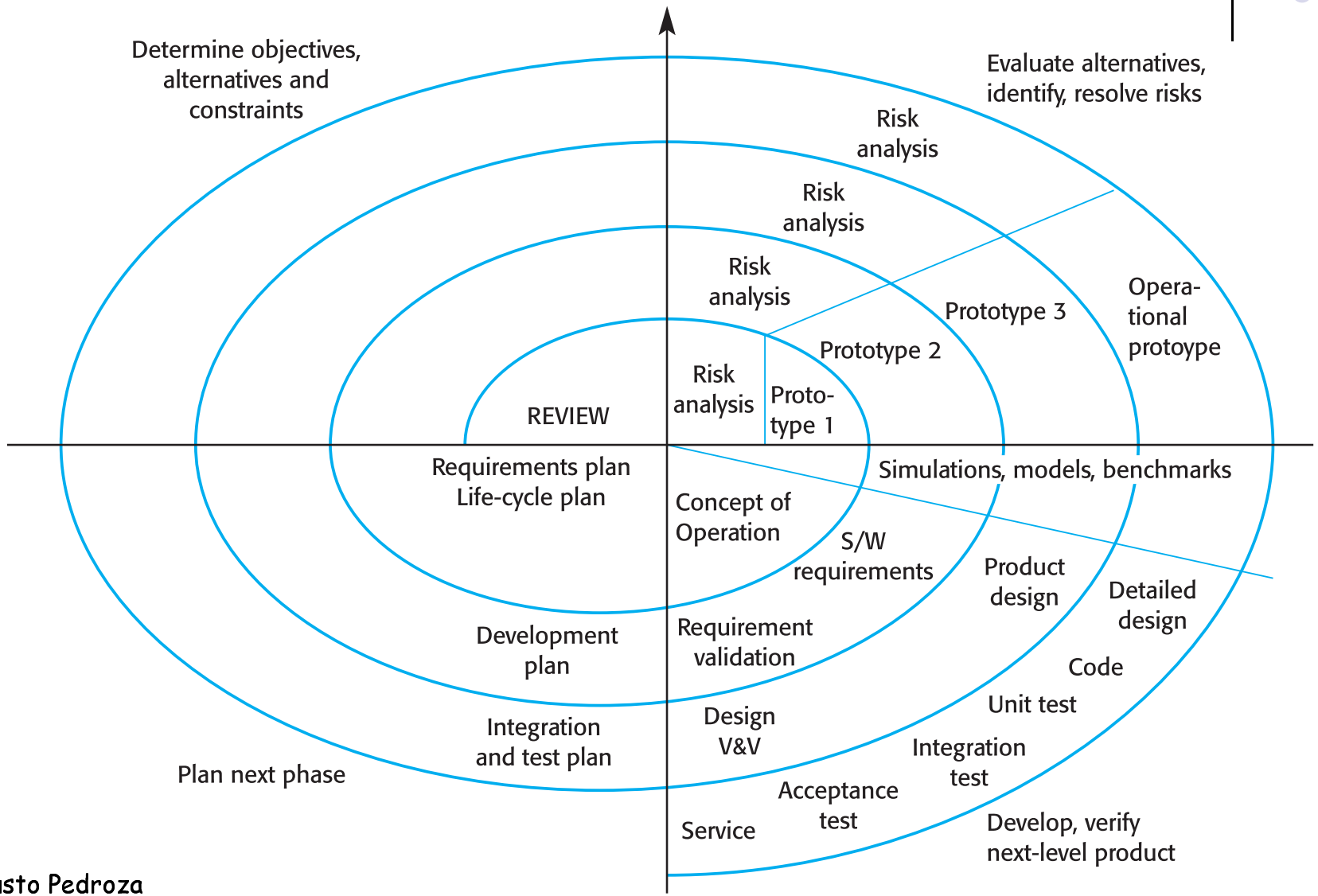
- Um enfoque de desenvolvimento baseado em entregas de incrementos muito pequenos de funcionalidade.
- Baseia-se em melhoria constante de código, no envolvimento do usuário no time de desenvolvimento e na programação em pares.
- Será abordado em detalhes após o estudo detalhado da disciplina de design.

Desenvolvimento Espiral

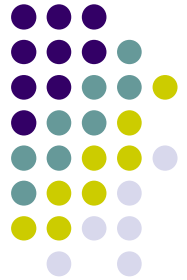


- O processo é representado como uma espiral, em vez de uma seqüência de atividades com realimentação.
- Cada loop na espiral representa uma fase no processo.
- Não há fases específicas como especificação ou design – loops na espiral são escolhidos dependendo do que for requerido.
- Riscos são explicitamente avaliados e resolvidos através do processo.

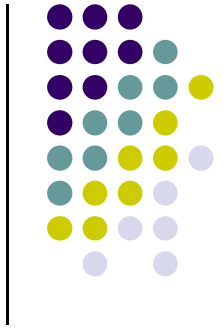
O Modelo Espiral



Setores do Modelo Espiral

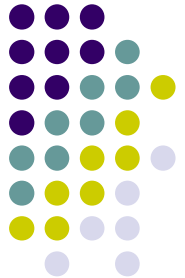


- Configuração do objetivo
  - Objetivos específicos para a fase são identificados.
- Avaliação e mitigação de riscos
  - Riscos são avaliados e atividades são destacadas reduzir os principais riscos.
- Desenvolvimento e Validação
  - Um modelo de desenvolvimento para o sistema é escolhido, o qual pode ser qualquer um dos modelos genéricos.
- Planejamento
  - O projeto é revisado e a próxima fase da espiral é planejada.



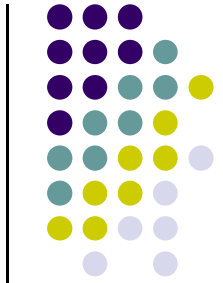
DISCIPLINAS

Atividades do Processo



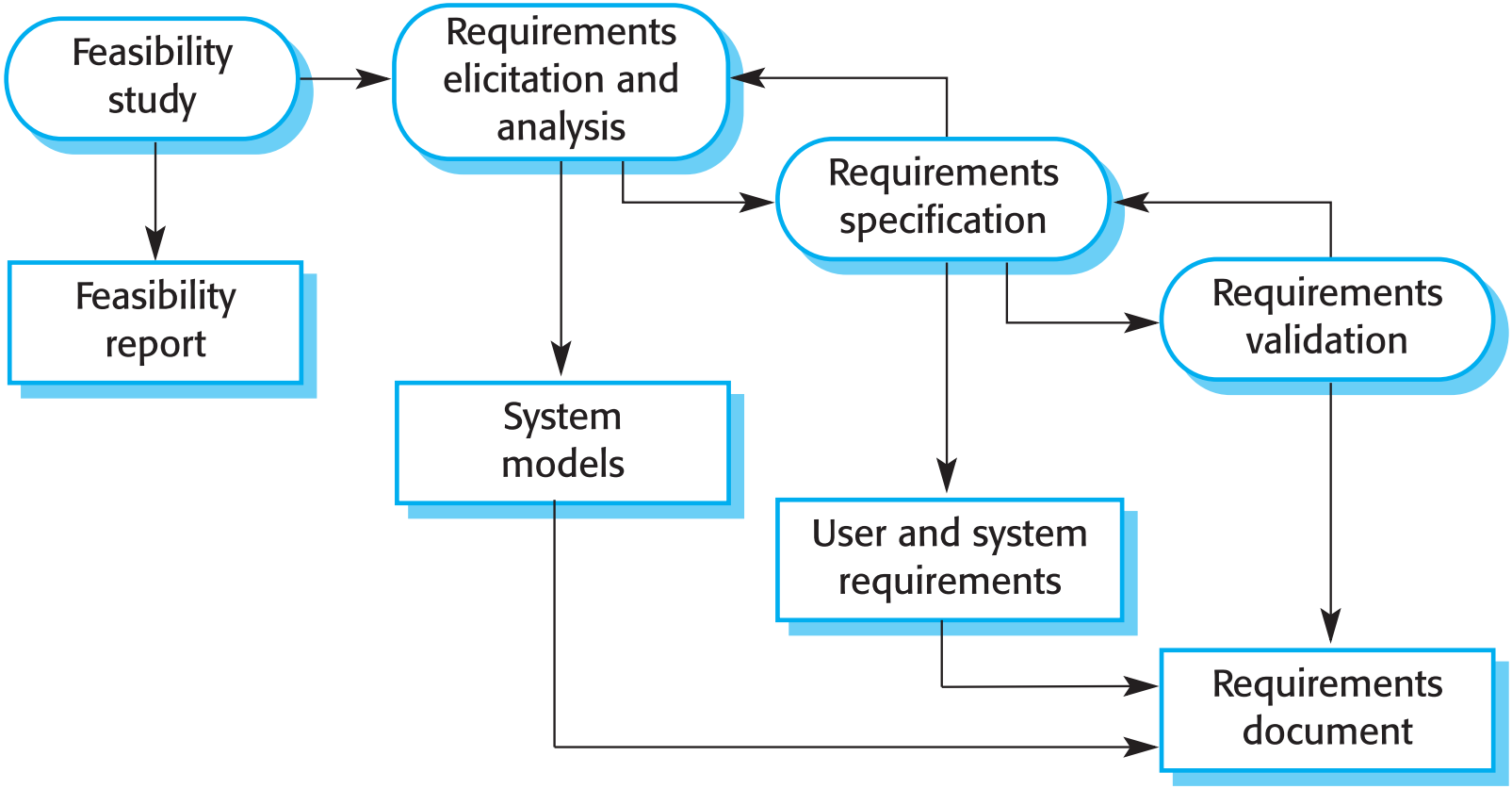
- Especificação de Software.
- Projeto e Implementação de Software.
- Validação do Software.
- Evolução do Software.

Especificação do Software



- O processo de estabelecer quais serviços são requeridos e as restrições sobre a operação e desenvolvimento do sistema
- Processo de engenharia de requisitos
  - Estudo de viabilidade.
  - Elucidação e análise de requisitos.
  - Especificação de requisitos.
  - Validação de requisitos.

Processo da Engenharia de Requisitos

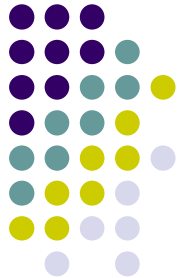


Sommerville, Software Engineering, 7th edition.

Projeto e Implementação do Software

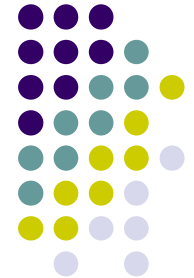
- O processo de converter a especificação do sistema num sistema executável.
- Projeto de software
  - Projetar uma estrutura de software que realiza a especificação.
- Implementação
  - Traduzir esta estrutura num programa executável.
- As atividades de projeto e implementação estão intimamente relacionadas e podem ser sobrepostas.

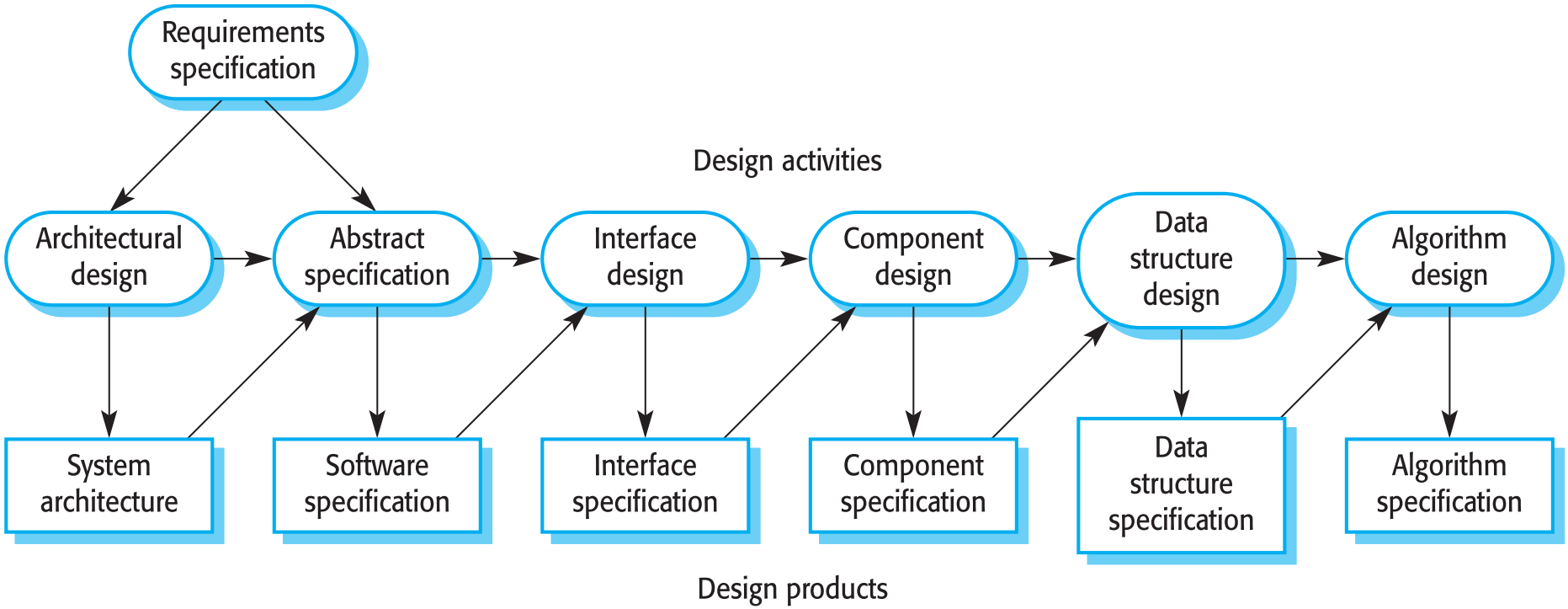
Atividades do Processo de Projeto (Design)



- Projeto arquitetural.
- Especificação abstrata.
- Projeto de interface.
- Projeto de componente.
- Projeto da estrutura de dados.
- Projeto de algoritmo.

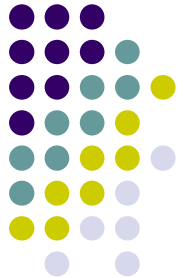
Atividades do Processo de Projeto





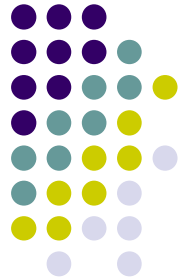
Sommerville, Software Engineering, 7th edition.

Métodos Estruturados



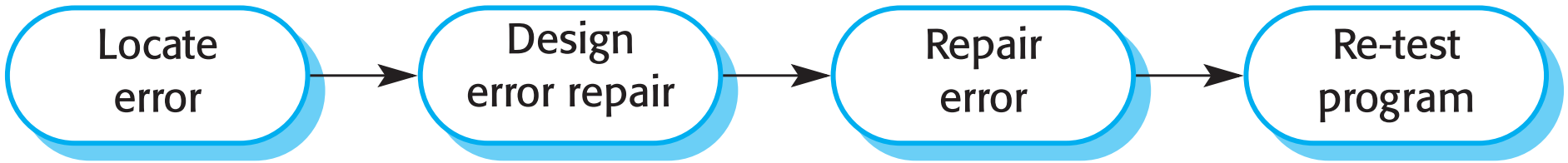
- Abordagem sistemática para desenvolver um projeto de software.
- O projeto é normalmente documentado como um conjunto de modelos gráficos.
- Modelos possíveis
  - Modelo de objetos.
  - Modelo de seqüência.
  - Modelo de transição de estado.
  - Modelo de fluxo de dados.
  - Modelo estrutural.

Programação e Depuração

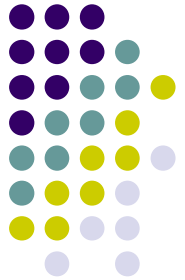


- Traduzir um projeto para um programa e remover erros do programa.
- A programação é uma atividade pessoal – não existe processo geral de programação.
- Programadores executam alguns testes no programa para descobrir falhas no programa e os remove no processo de depuração.

O Processo de Depuração

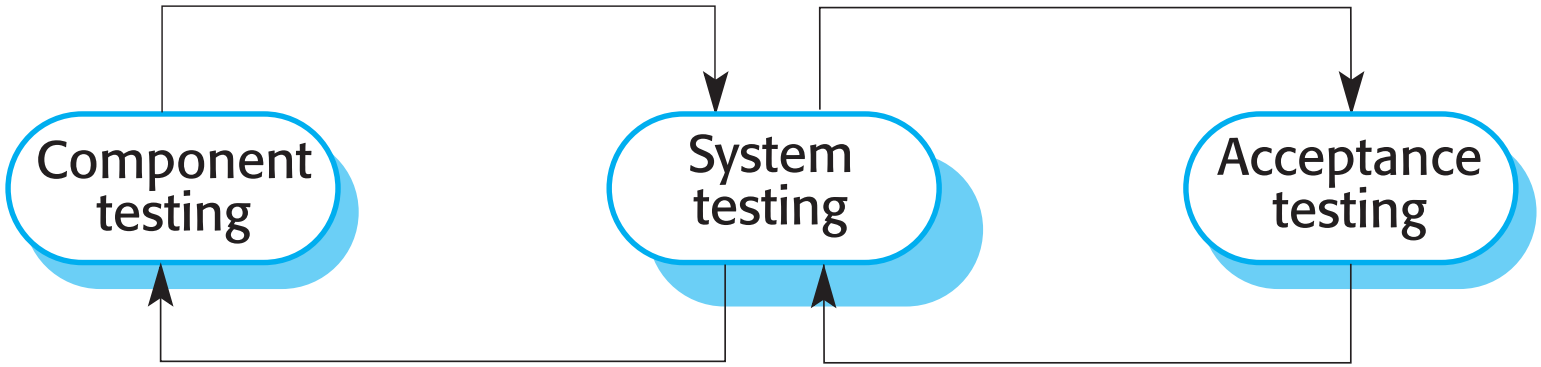


Validação do Software

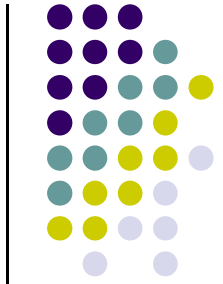


- A verificação e a validação (Ver e Val – V e V) pretendem mostrar que um sistema conforma-se com a sua especificação e atende aos requisitos do sistema – agrega valor ao cliente.
- Envolvem atividades de verificação e revisão de processos e testes do sistema.
- Os testes envolvem a execução do sistema com casos de testes que são derivadas da especificação dos dados reais a serem processadas pelo sistema.

O Processo de Teste

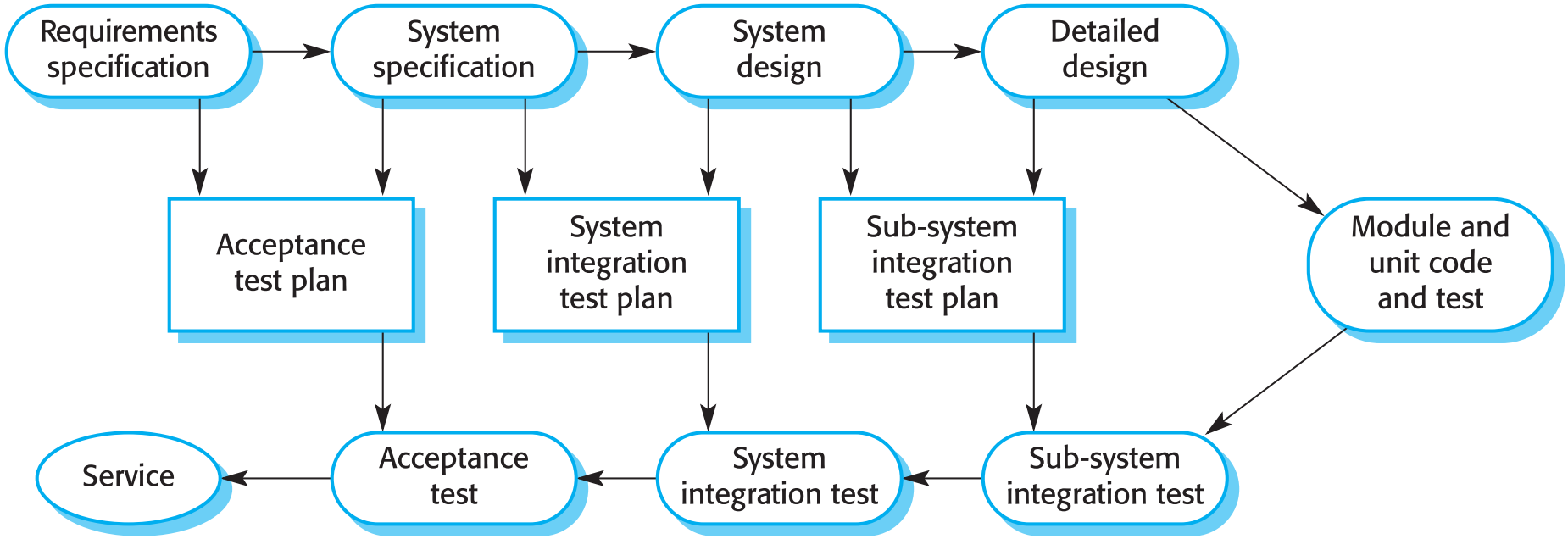


Estágios do Testes



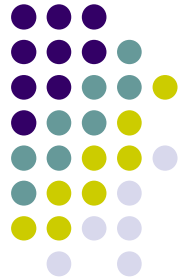
- Testes de unidade ou componente
  - Componentes individuais são testados independentemente.
  - Componentes podem ser funções ou objetos ou agrupamentos coerentes dessas entidades.
- Teste de sistema
  - Testar o sistema como um todo. O teste das propriedades emergentes é particularmente importante.
- Teste de aceitação
  - Teste com os dados do cliente para verificar se o sistema atende às necessidades do cliente.

Fases do Teste



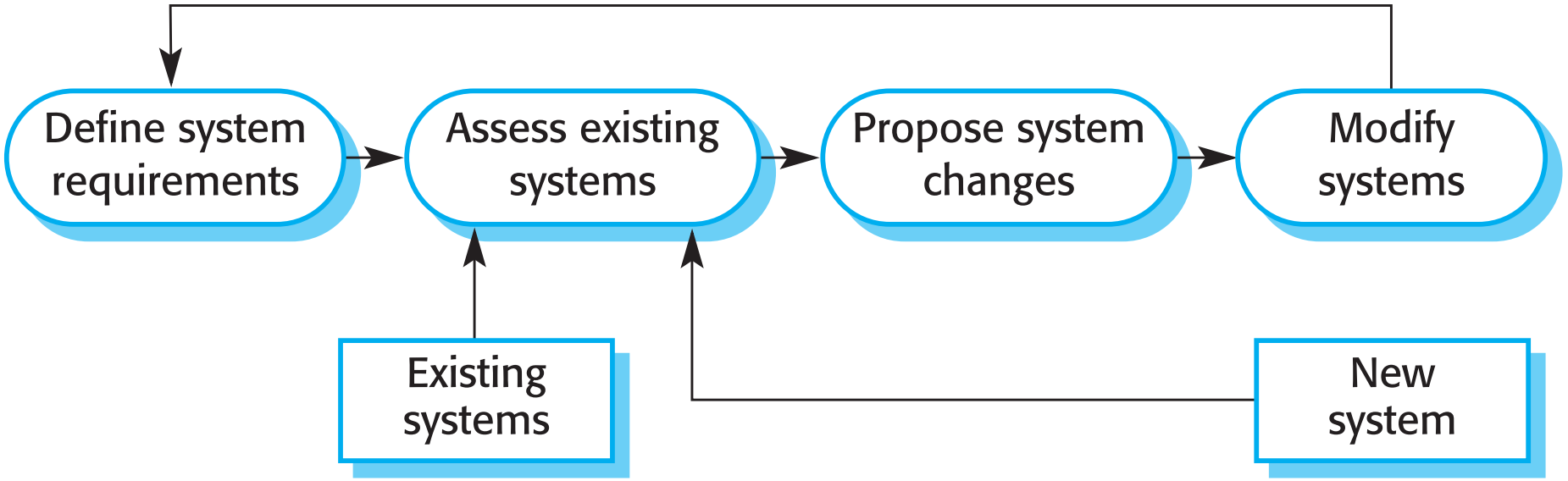
Sommerville, Software Engineering, 7th edition.

Evolução do Software

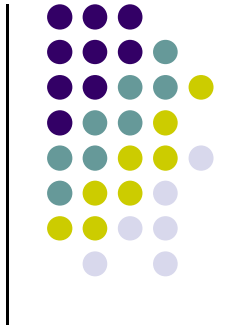


- O software é inerentemente flexível e pode mudar (tendência à conformidade).
- Como os requisitos mudam de acordo com as circunstâncias do negócio, o software que sustenta o negócio deve evoluir e mudar.
- Embora tenha existido uma demarcação clara entre o desenvolvimento e evolução (manutenção) isto está se tornando cada vez mais irrelevante, ao passo que cada vez há menos sistemas completamente novos.

Evolução do Sistema



Sommerville, Software Engineering, 7th edition.

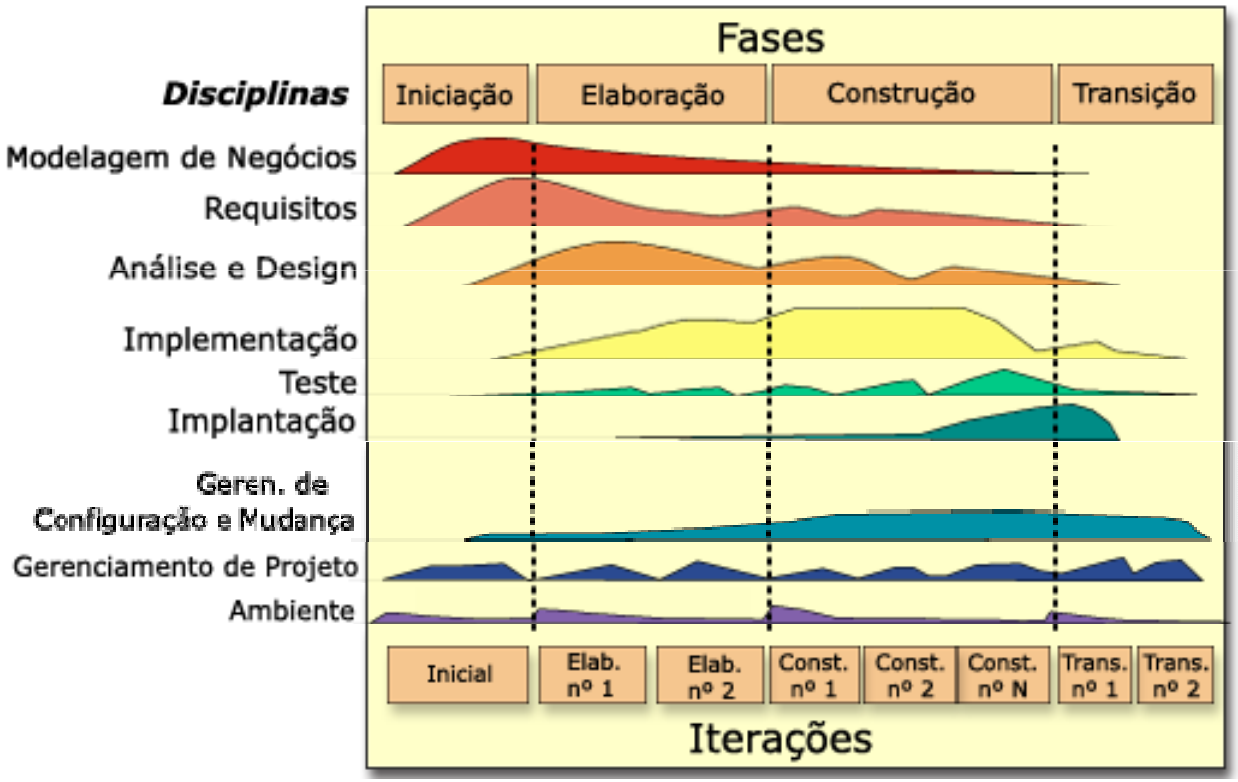


O RUP

Rational Unified Process

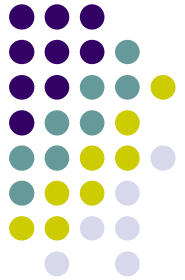
- Um modelo moderno de processo baseado no trabalho com UML e processos associados.
- Normalmente descrito a partir de três perspectivas.
  - Uma perspectiva dinâmica que mostra fases sobre tempo.
  - Uma perspectiva estática que mostra as atividades do processo.
  - Uma perspectiva prática que sugere boas práticas.

O Modelo de Fases do RUP



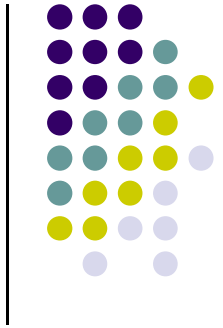
RUP Tutorial – Rational - IBM.

Pilares Básicos do UP



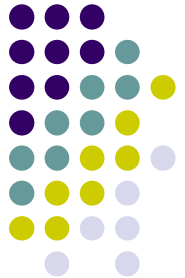
- Fases: Iniciação, Colaboração, Construção e Transição.
- Dirigido por casos de uso.
- Centrado em uma solução de arquitetura.
- Iterativo e Incremental.

Iniciação



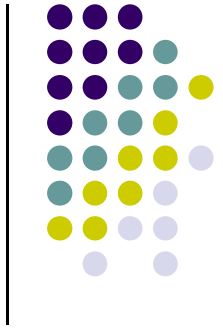
- Determinar o escopo do projeto (limites, do projeto, teto de custo, tempo etc)
- Levantamento dos riscos
- Criar o Business Case (Plano de negócio)
  - Quanto custará?
  - Qual o retorno?
- Levantamento inicial de requisitos (funcionais e não funcionais)
- Proposta de uma arquitetura candidata

Iniciação (2)



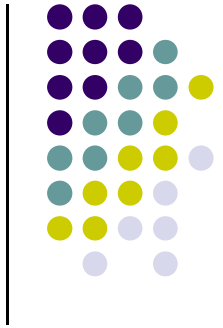
- Preparar o ambiente para o projeto
  - avaliando o projeto e a organização,
  - selecionando ferramentas e decidindo quais partes do processo devem ser melhoradas.

Elaboração



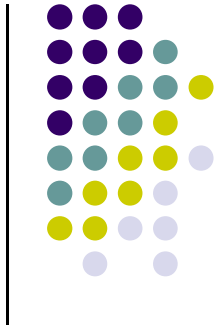
- Definir, validar e criar a *Baseline* da arquitetura (prova de conceito arquitetural).
- Análise dos requisitos restantes, empacotando os funcionais em casos de uso.
- Refinar a Visão, com base nas informações novas obtidas durante a fase, estabelecendo uma compreensão sólida dos casos de uso mais críticos que conduzem as decisões de arquitetura e planejamento.

Elaboração (2)



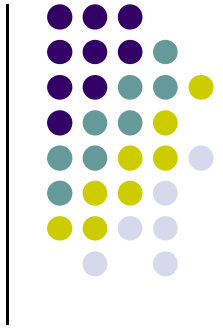
- Refinar os riscos.
- Planejamento da etapa de Construção (criar planos de iteração detalhados e *baselines*).
- Refinar a estratégia de desenvolvimento e posicionar o ambiente de desenvolvimento, incluindo o processo, as ferramentas e o suporte de automatização necessários para dar assistência à equipe de construção.

Elaboração (3)



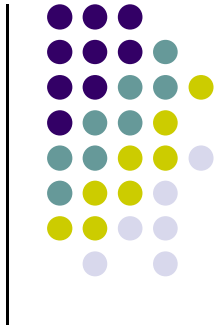
- Refinar a arquitetura e selecionar componentes. Os componentes potenciais são avaliados e as decisões de fazer/comprar/reutilizar são bem compreendidas para determinar o custo da fase de construção e programar com confiança.
- Os componentes de arquitetura selecionados são integrados e avaliados em comparação com os cenários básicos. As lições aprendidas dessas atividades podem resultar em um novo design da arquitetura, levando em consideração designs alternativos ou reconsiderando os requisitos.

Construção



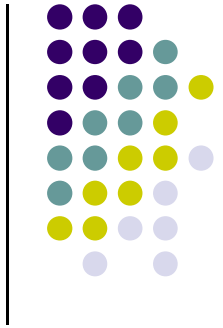
- Desenvolver de modo iterativo e incremental um produto completo que esteja pronto para a transição para a sua comunidade de usuários.
- Concluir a análise, o design, o desenvolvimento e o teste de todas as funcionalidades necessárias
- Isso implica descrever os casos de uso restantes e outros requisitos, incrementar o design, concluir a implementação e testar o software.

Construção (2)



- Normalmente há componentes que podem ser desenvolvidos um independente do outro, permitindo o paralelismo natural entre as equipes (se os recursos permitirem).
- O paralelismo pode acelerar bastante as atividades de desenvolvimento, mas aumenta a complexidade do gerenciamento de recursos e da sincronização dos fluxos de trabalho.
- Uma arquitetura sofisticada será essencial para atingir um paralelismo significativo.

Transição



- Executar planos de implantação
- Finalizar o material de suporte para o usuário final
- Testar o produto liberado no local do desenvolvimento
- Criar um release do produto
- Obter feedback do usuário
- Ajustar o produto com base em feedback
- Disponibilizar o produto para os usuários finais

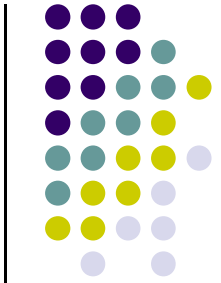
Atividades de cada iteração

- Os workflows e modelos do RUP variam conforme o esquema abaixo:
  - Modelo de Negócio (modelo de casos de uso de negócio)
  - Requisitos (modelo de casos de uso sistêmicos);
  - Análise (modelo de análise);
  - Projeto (modelo de projeto e modelo de implantação);
  - Implementação (modelo de implementação);
  - Teste (modelo de teste).
  - Implantação, Gerência de Configuração e Mudança, Gerência de Projeto e Ambiente.

RUP Dirigido por casos de uso

- Caso de uso: seqüência de ações que o sistema efetuará para oferecer alguns resultados de valor a um ator (um tipo de usuário de um sistema).
- Pergunte: o que um sistema pode fazer por seus atores? Para quê um ator usa um sistema? As respostas virão em formatos de casos de uso.

RUP Dirigido por casos de uso (2)



- Um caso de uso, portanto, é um caso de uso de um sistema! Ou:
  - uma funcionalidade fornecida por um sistema a um ator;
  - uma forma de usar um sistema.
- Exemplo: Seja o sistema de Biblioteca da Unifor. Os alunos (atores) usam o sistema para renovar empréstimos de livros.
- Então, para que os alunos usam o sistema (em quais casos ele usam o sistema)?
- Se o usam para renovar empréstimo, este é um caso de uso do sistema.
- Se o usam para fazer reserva, então este é outro caso de uso do sistema.

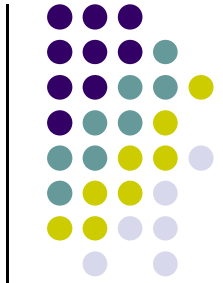
RUP Dirigido por casos de uso (3)

- Objetivos do levantamento de requisitos:
  - encontrar os verdadeiros requisitos (agregam valor aos usuários – necessidades, não desejos);
  - representá-los em uma forma sistematizada e intuitiva aos stakeholders.
- Objetivos da Análise
  - especificar mais detalhadamente os requisitos, ajudando na sua compreensão (dos UCs);
  - criar um modelo primário e conceitual dos principais elementos do sistema, focado em refinar as suas interações;
  - o modelo criado é muitas vezes transiente - será transformado em modelo de design.

Desenvolvimento dirigido por UC (4)

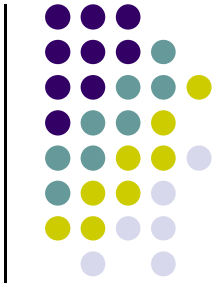
- O Modelo de projeto
  - Modelo de classes refinado, com as responsabilidades reportadas nos UCs sendo alocadas para classes específicas e relacionamentos múltiplos (associações, generalizações, dependências e realizações).
  - Uso de tecnologias na especificação do modelo (orbs, frameworks, tookits).
  - Agrupamento de classes em subsistemas/componentes.
  - O modelo de deploy define a organização física do sistema.

Desenvolvimento dirigido por UC (5)



- O modelo de implementação reflete o empacotamento das funcionalidades em componetes e suas interações.
- O modelo de teste é uma coleção de casos de teste.
- Cada caso de teste define um conjunto de entradas, condições de execução e resultados.
- A maioria dos casos de teste pode ser derivada diretamente dos casos de uso - outros dos requisitos suplementares.

Por que fazer casos de Uso?



- Na fase de iniciação, casos de uso de alto nível são desenvolvidos para auxiliar a definir o escopo e o contexto, bem como auxiliar no planejamento e nas estimativas do projeto como um todo.
- O que deve ser incluído no projeto e o que pertence a outro projeto?
- O que pode-se realmente ser construído com os atuais orçamento e cronograma?

Por que fazer casos de Uso? (2)

- Na fase de Elaboração, desenvolve-se casos de uso mais detalhados.
- Esse detalhamento contribui para a análise de risco e o *Baseline* da arquitetura.
- Usados no planejamento das iterações da etapa de Construção e para dirigir todo o processo de desenvolvimento, uma vez que a maioria das atividades iniciam a partir dos casos de uso.

Por que fazer casos de Uso? (3)

- Na fase de Construção, os casos de uso são os pontos de partida para desenvolver os planos de teste e para o projeto (design).
- O detalhamento de alguns casos de uso pode acontecer como parte da análise de cada iteração.
- Os casos de uso fornecem os requisitos a serem satisfeitos em cada iteração.

Por que fazer casos de Uso? (4)

- Na fase de Transição, os casos de uso são usados para desenvolver os manuais do usuário e o material de treinamento.

UP - Centrado na Arquitetura

- Uma solução de arquitetura de software é uma visão comum da organização do sistema, na qual todos os envolvidos concordam ou aceitam.
- A busca da solução sobre como o sistema estará organizado recebe tem como insumos principais os casos de uso críticos e os requisitos não funcionais.
- O Documento de Arquitetura de Software fornece uma visão geral de arquitetura abrangente do sistema, usando diversas visões de arquitetura para descrever diferentes aspectos do sistema.
- A finalidade é identificar e documentar as decisões mais significantes sobre a arquitetura de software para orientar o desenvolvimento, manutenção e a implantação deste sistema.

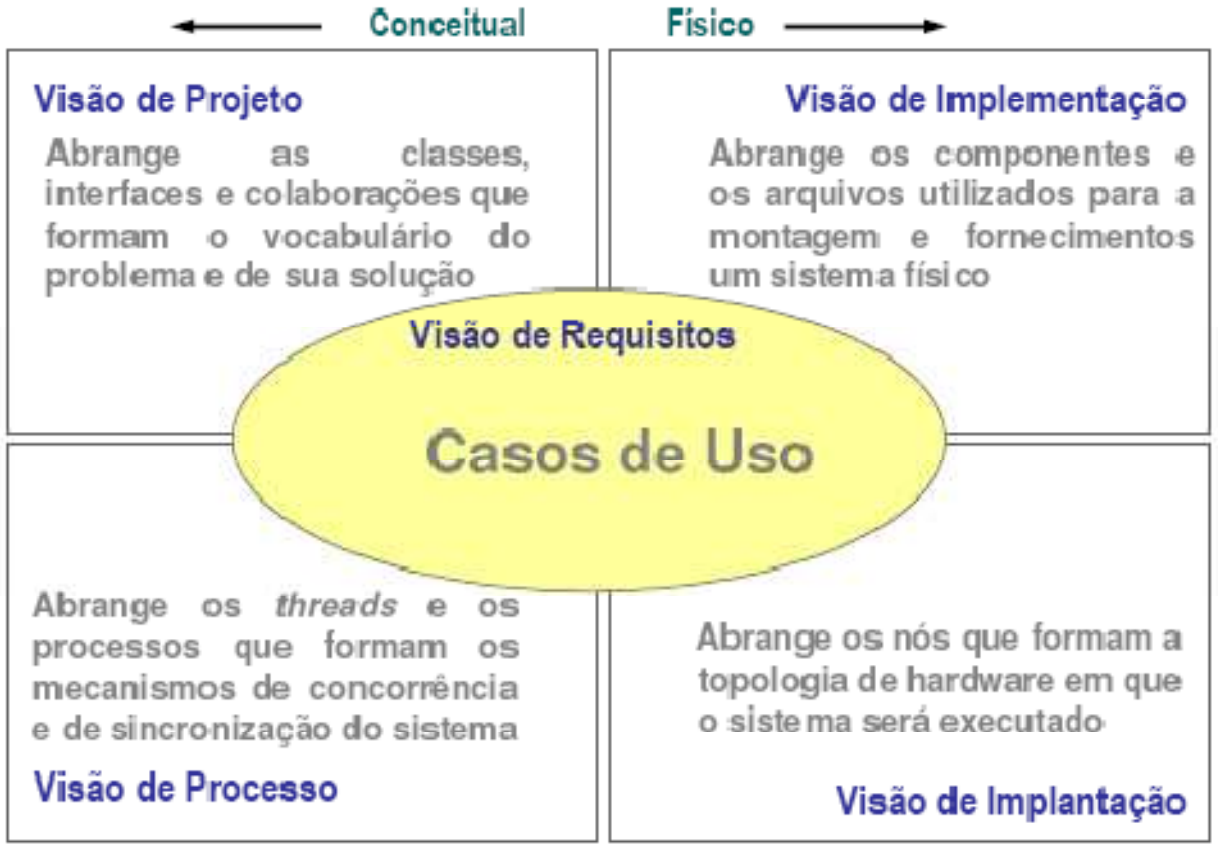
UP - Centrado na Arquitetura (2)

- O público alvo do documento de arquitetura de software é, principalmente, o conjunto de papéis desempenhados pelos integrantes das disciplinas de Análise e Projeto, Implementação e Implantação, mas poderá ser referenciado por qualquer integrante do projeto.
- Os conceitos e modelos ali definidos, se respeitados, garantirão consistência e um nível de homogeneidade do sistema que permitirá sua fácil evolução, entendimento e escala.
- A arquitetura de um sistema diz respeito a sua estrutura organizacional, incluindo a decomposição em partes, a inter-relação entre essas partes, além dos mecanismos e princípios que devem guiar o projeto de um sistema.

UP - Centrado na Arquitetura (3)

- Desta forma, pode-se definir a arquitetura como a captura dos aspectos estratégicos da estrutura em alto nível de um sistema.
- A arquitetura, conforme o padrão de modelagem descrito pela UML, pode ser analisada através de 5 divisões (4 + 1) de escopo chamadas de visões.
- A motivação principal para se usar diferentes visões para a arquitetura é a de reduzir a complexidade como um todo. Cada uma destas visões representa um aspecto específico da arquitetura.
- a UML centraliza seu enfoque no tratamento de casos de uso, visão que permeia as outras visões.

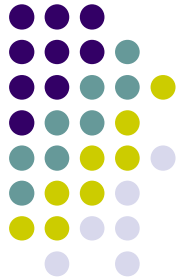
UP - Centrado na Arquitetura (4)



UP - Centrado na Arquitetura (5)

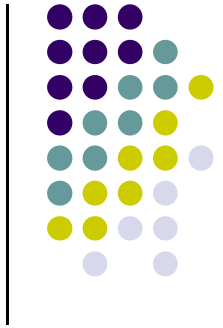
- A organização da arquitetura a qual nos referimos, portanto, engloba decisões sobre:
  - Os elementos estruturais que comporão o sistema e as suas interfaces, junto com o comportamento esperado e exigido de cada elemento, a partir das colaborações necessárias entre as partes;
  - A composição progressiva, em subsistemas, dos elementos estruturais e comportamentais da solução;
  - O estilo arquitetural – visão 4 + 1 da arquitetura;
  - Aspectos gerais da solução, relacionados ao atendimento dos principais requisitos não funcionais da solução;

Iterativo e Incremental



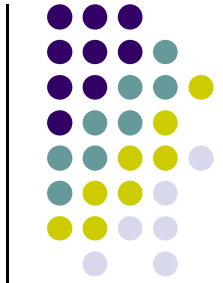
- Divide a construção do software em pequenas iterações (1-3 meses).
- A progressão linear da cascata é irreal.
- Realimentação.
- Simplifica o desenvolvimento - pequenas partes.
- Aumenta a motivação.
- Melhor para provisionar.

Boas Práticas RUP



- Desenvolver iterativamente.
- Gerenciar requisitos.
- Usar Arquiteturas baseadas em componentes.
- Modelar o software visualmente.
- Verificar a qualidade do software.
- Gerenciar as mudanças.

Pontos Principais

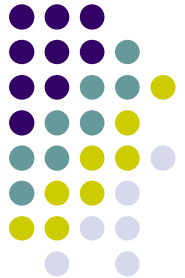


- Processos de software são as atividades envolvidas na produção e evolução de um sistema de software.
- Modelos de processo de software são representações abstratas desses processos.
- Atividades gerais são especificação, projeto e implementação, validação e evolução.
- Modelos genéricos de processos descrevem a organização do processo de software – por exemplo, cascata, evolutivo, baseado em componentes.
- Modelos de processos iterativos descrevem o processo de software com um ciclo de atividades.

Pontos Principais (2)

- A engenharia de requisitos é o processo de desenvolver uma especificação de software.
- Os processos de projeto e implementação transformam a especificação em um programa executável.
- A validação envolve a verificação de que o sistema atende à sua especificação e às necessidades do usuário.
- A evolução se preocupa com a modificação do sistema depois que o mesmo está em uso.
- O RUP é um modelos genérico de processo que separa atividades de fases.

Referências



- SOMMERVILLE, Ian. Software Enginnering (update). 9th Edition, Addison Wesley, 2010.
- JACOBSON, Ivar; BOOCH, Grady; RUMBAUGH, James; “Unified Software Development Process”. Addison Wesley, 1999.
- ARLOW, Jim; NEUSTADT, Ila; “UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design” , second edition. Addison Wesley, 2005.
- KRUCHTEN, Philippe. “The Rational Unified Process: An Introduction”, Third Edition. Addison Wesley, 2003.