



Desenvolvimento de Sistemas Software

Aula Teórica 2: Métodos de Desenvolvimentos



Engenharia de Software

- Engenharia

*“The creative application of scientific principles to **design** or **develop** structures, machines, apparatus, or manufacturing processes, or works utilizing them singly or in combination; or to **construct** or **operate** the same with full cognizance of their design; or to forecast their behavior under specific operating conditions; all as respects an intended function, economics of operation or safety to life and property” (Engineers' Council for Professional Development)*

- Engenharia de Software

*“Aplicação de um método/processo **sistemático, disciplinado e quantificável** à concepção, desenvolvimento, operação e manutenção de software” (IEEE CS)*

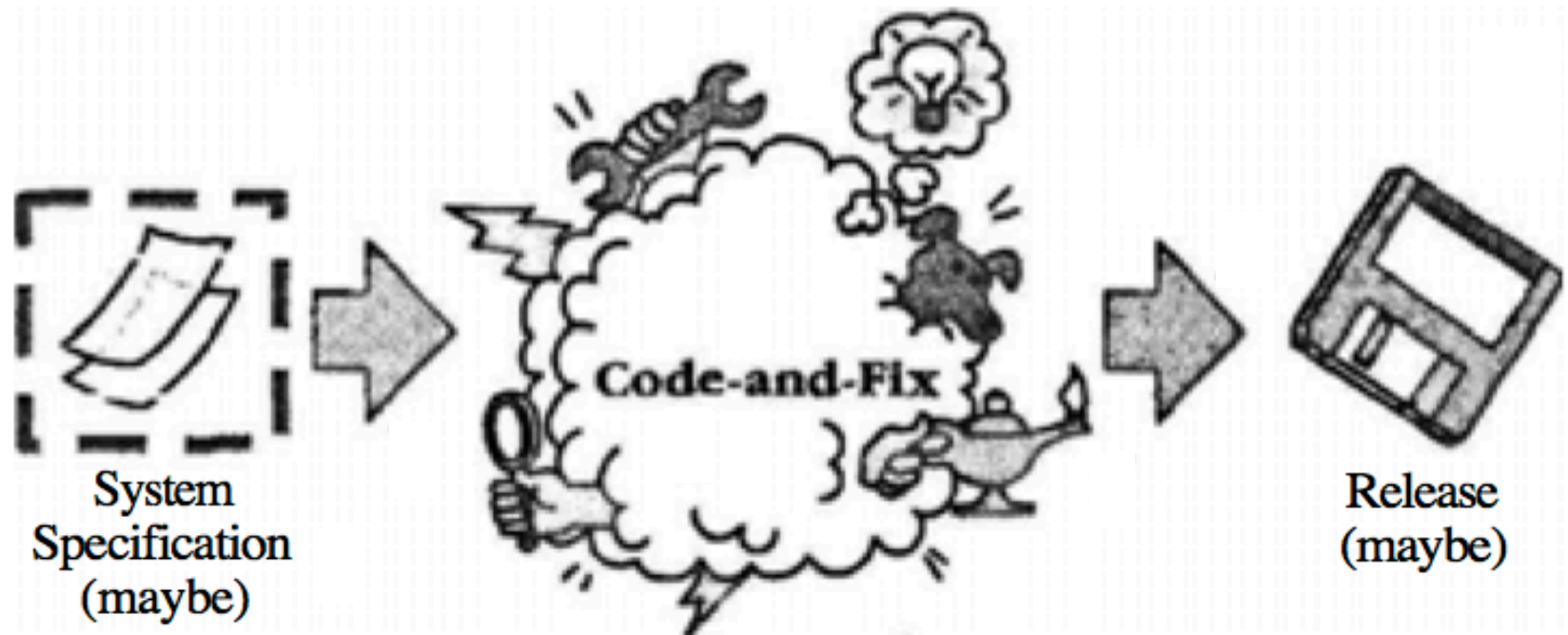


Processo de Desenvolvimento de Software

- Define como se estrutura o desenvolvimento de software
- Identifica as fases de desenvolvimento e como se passa de umas para outras
 - Quem faz o quê
 - Quando é feito
 - e Durante quanto tempo



“Code and fix”

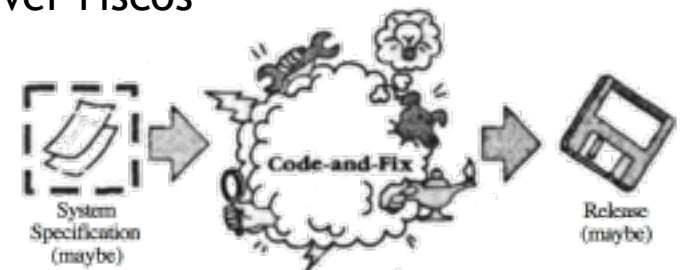


(McConnell, 1996)



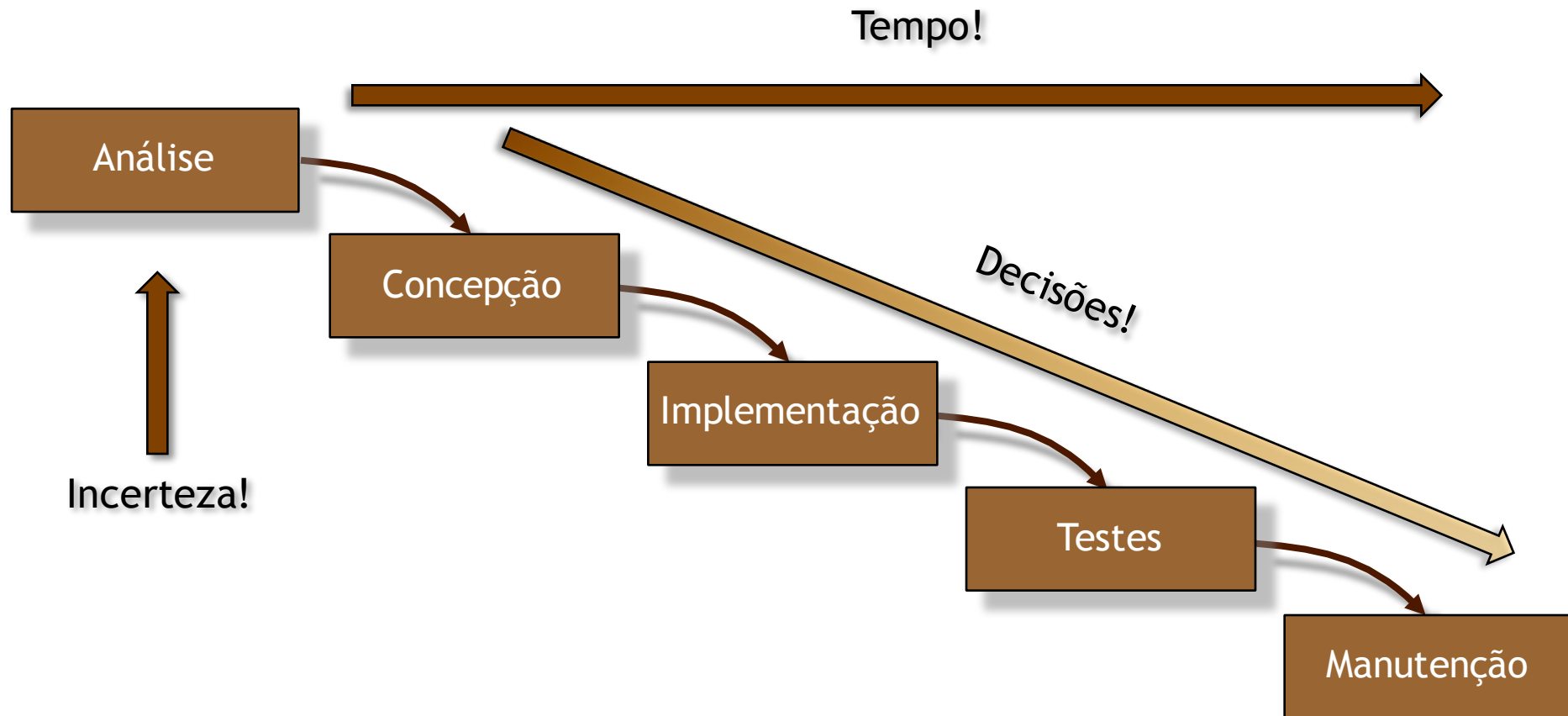
“Code and fix”

- Raramente útil, mas muito utilizado
- Não tanto uma estratégia deliberada, mais o resultado de falta de planeamento e tempo
 - Sem muita análise, começa-se a produzir código de imediato
 - Eventualmente fazem-se testes e os inevitáveis erros têm que ser resolvidos
- “*Vantagens*”
 - Produção de código é imediata
 - Não requer muita competência - qualquer programador pode utilizar
- Problemas
 - Não existe forma de avaliar progresso
 - Não existe forma de avaliar qualidade e/ou de prever riscos
- Apenas viável para pequenos projectos
 - provas de conceito ou *demos*





Modelo em Cascata (Waterfall)

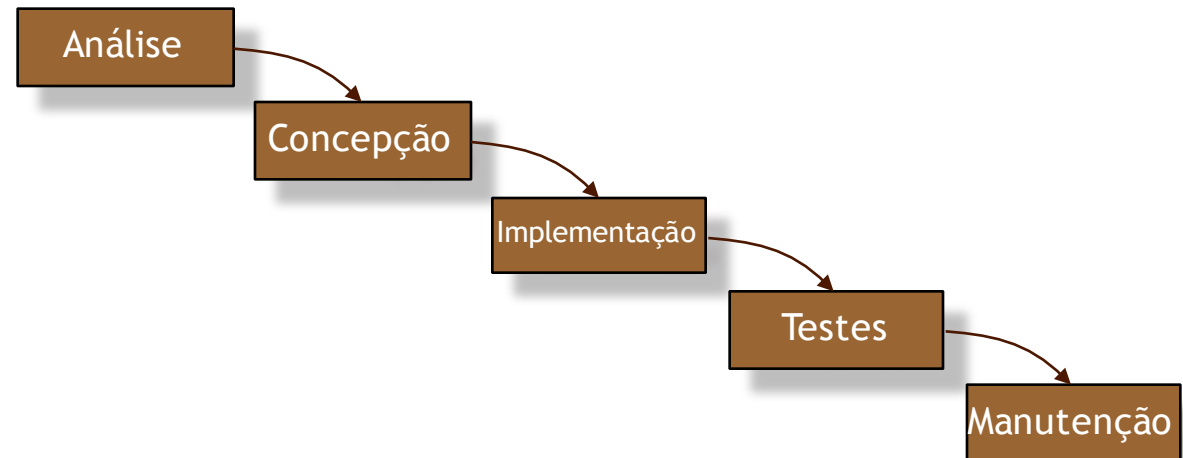


- Define uma série de etapas executadas sequencialmente.



Modelo em Cascata

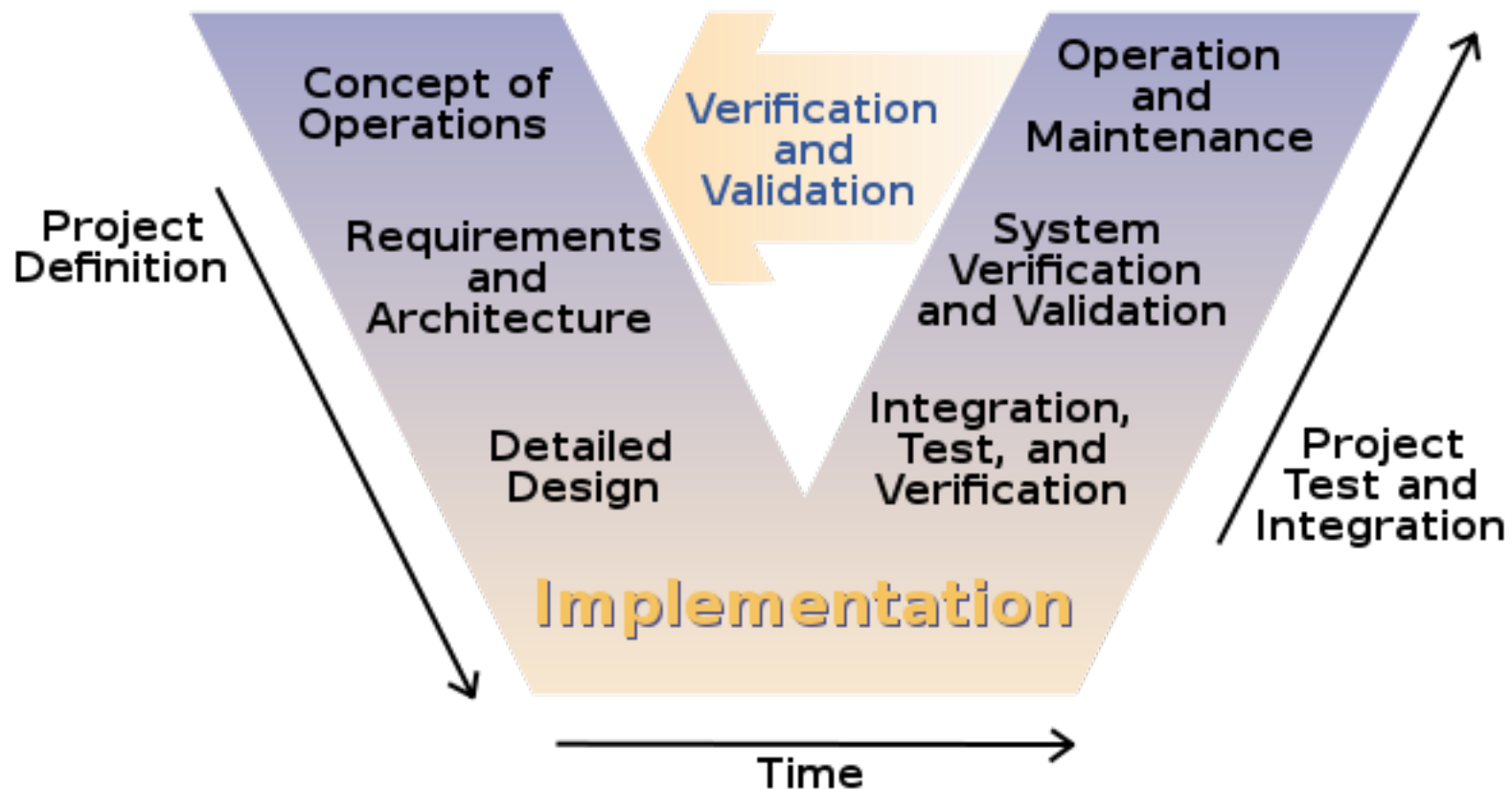
- Define uma série de etapas executadas sequencialmente.
- Assume que é sempre possível tomar as decisões mais correctas
 - como prever situações de alto risco mais à frente?
 - como favorecer reutilização?



- Irrealista!



Modelo em V



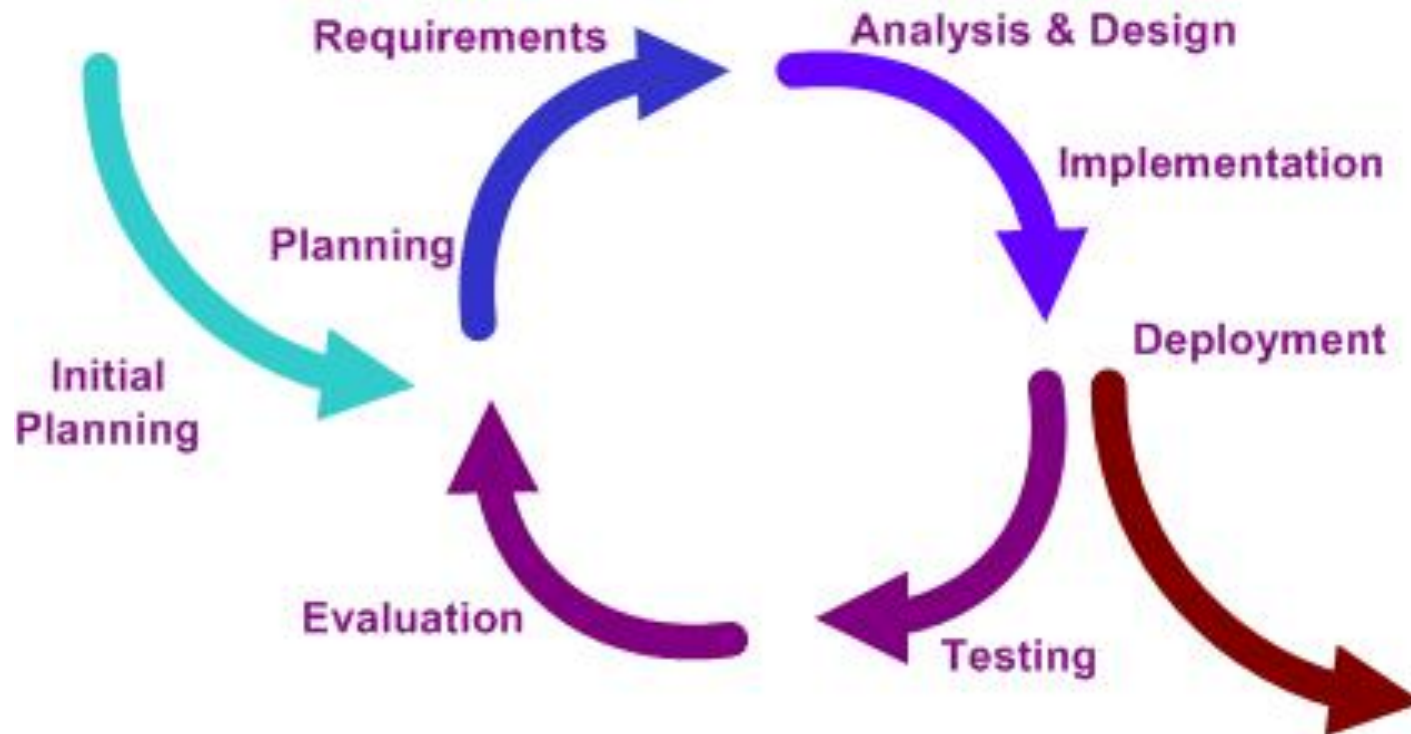


Modelo em V

- Uma extensão do Modelo em Cascata
- Torna explícita a relação entre as fases de desenvolvimento e as fases de teste
- Verificação e validação continuam a aparecer após implementação
 - Validação - Estamos a construir o systema certo?
(are we building the right system?)
 - Verificação - Estamos a construir bem o sistema?
(are we building the system right?)
- Popular na área do hardware
- Problemas com rigidez da abordagem continuam



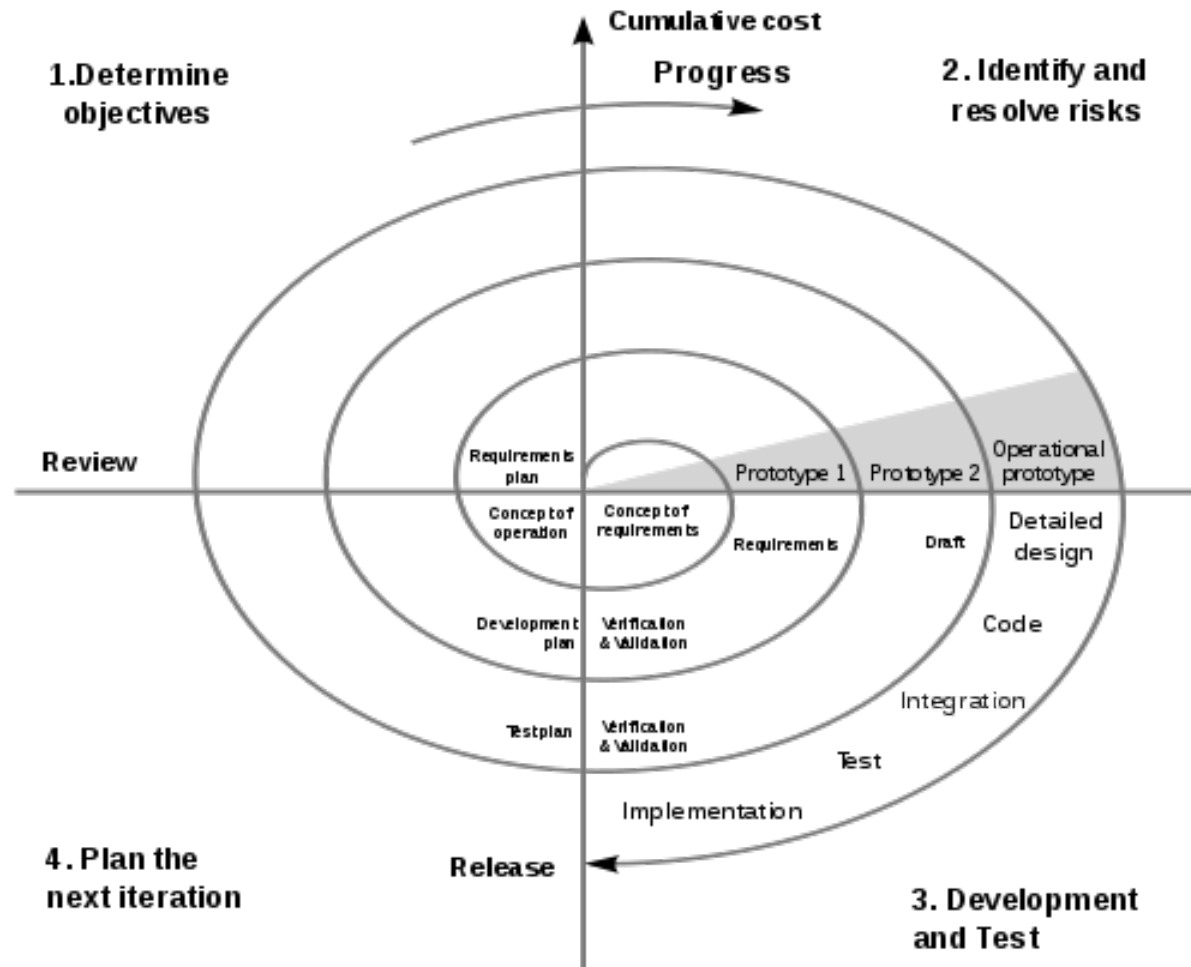
Desenvolvimento Iterativo e Incremental





Modelo em Espiral

(Boehm, 1986)





Modelo em Espiral

- Combina características do Modelo em Cascata com uma abordagem de prototipagem.
- Mais indicado para projectos de larga escala, dispendiosos e/ou complexos
- Foca-se na necessidade de iterar para controlar riscos
 1. Definição dos requisitos com o maior detalhe possível
 2. Concepção de um primeiro “desenho” do sistema.
 3. Um primeiro protótipo é construído a partir do passo anterior
 4. Desenvolvimento de um segundo protótipo a partir do primeiro (este passo é iterado até o cliente estar satisfeito):
 - ① avaliação do protótipo anterior (pontos fortes e fracos, riscos)
 - ② definição de requisitos para o novo protótipo
 - ③ planeamento e conceção do novo protótipo
 - ④ construção e teste do novo protótipo
 5. O produto final é construído a partir do protótipo final e, depois de avaliado e testado, entra em produção.
- O cliente pode optar por abortar o projecto se considerar o risco de desenvolver o produto demasiado alto.
 - custo de desenvolvimento, custo de operação, qualquer outro factor relevante para a obtenção de um produto satisfatório



Modelo em Espiral - prós e contras

- Vantagens
 - maior capacidade de lidar com incerteza, maior controlo de risco
 - promove a inclusão de objectivos de qualidade no processo de desenvolvimento
 - permite demonstração/exploração via protótipos do sistema desde cedo
 - flexibilidade da abordagem - no limite pode tornar-se no modelo em cascata!
- Problemas
 - Dependência da qualidade da avaliação de risco
 - Análise de risco requer competências muito específicas
 - Custo pode disparar (análise de risco)
 - Processo ainda rígido
 - Iterações de 6 meses a 2 anos
 - Funciona melhor para projectos grandes

Problemas...

- Processos muito rígidos e burocráticos!





Agile Manifesto (Beck et al. 2001)

Ao desenvolver e ao ajudar outros a desenvolver software, temos vindo a descobrir melhores formas de o fazer. Através deste processo começámos a valorizar:

Indivíduos e interacções mais do que processos e ferramentas

Software funcional mais do que documentação abrangente

Colaboração com o cliente mais do que negociação contratual

Responder à mudança mais do que seguir um plano

Ou seja, apesar de reconhecermos valor nos itens à direita, valorizamos mais os itens à esquerda.



Os Doze Princípios do Manifesto Ágil

1. A nossa maior prioridade é, desde as primeiras etapas do projecto, satisfazer o cliente através da entrega rápida e contínua de software com valor.
2. Aceitar alterações de requisitos, mesmo numa fase tardia do ciclo de desenvolvimento. Os processos ágeis potenciam a mudança em benefício da vantagem competitiva do cliente.
3. Fornecer frequentemente software funcional. Os períodos de entrega devem ser de poucas semanas a poucos meses, dando preferência a períodos mais curtos.
4. O cliente e a equipa de desenvolvimento devem trabalhar juntos, diariamente, durante o decorrer do projecto.
5. Desenvolver projectos com base em indivíduos motivados, dando-lhes o ambiente e o apoio de que necessitam, confiando que irão cumprir os objectivos.
6. O método mais eficiente e eficaz de passar informação para e dentro de uma equipa de desenvolvimento é através de **conversa pessoal e directa**.
7. A principal **medida de progresso** é a entrega de **software funcional**.
8. Os processos ágeis promovem o **desenvolvimento sustentável**. Os promotores, a equipa e os utilizadores deverão ser capazes de manter, indefinidamente, um ritmo constante.
9. A atenção permanente à **excelência técnica** e um bom desenho da solução aumentam a agilidade.
10. A **simplicidade** - a arte de maximizar a quantidade de trabalho que não é feito - é essencial.
11. As melhores arquitecturas, requisitos e desenhos surgem de **equipas auto-organizadas**.
12. A **equipa reflecte regularmente** sobre o modo de se tornar mais eficaz, fazendo os ajustes e adaptações necessárias.



Agile Methods - eXtreme Programming (XP)

- **Fine scale feedback**
 - Pair programming
 - Planning game
 - Test driven development
 - Whole team
- **Continuous process**
 - Continuous integration
 - Design improvement
 - Small releases
- **Shared understanding**
 - Coding standard
 - Collective code ownership
 - Simple design
 - System metaphor
- **Programmer welfare**
 - Sustainable pace



XP - prós e contras

- Vantagens
 - Focado no código
 - Código consistente e elegível
 - Focado nos testes
 - Software disponível incrementalmente
 - Focado na solução mais simples
 - Forte envolvimento do cliente
- Dificuldades
 - Exige disciplina, nem sempre bem aceite
 - Preço e duração do projecto?
 - Dificuldades com projectos ou equipas grandes
 - Qualidade não é medida nem planeada (característica emergente?)
 - Testes podem perder a '*big picture*'
 - Muita duplicação de código / muito *refactoring*



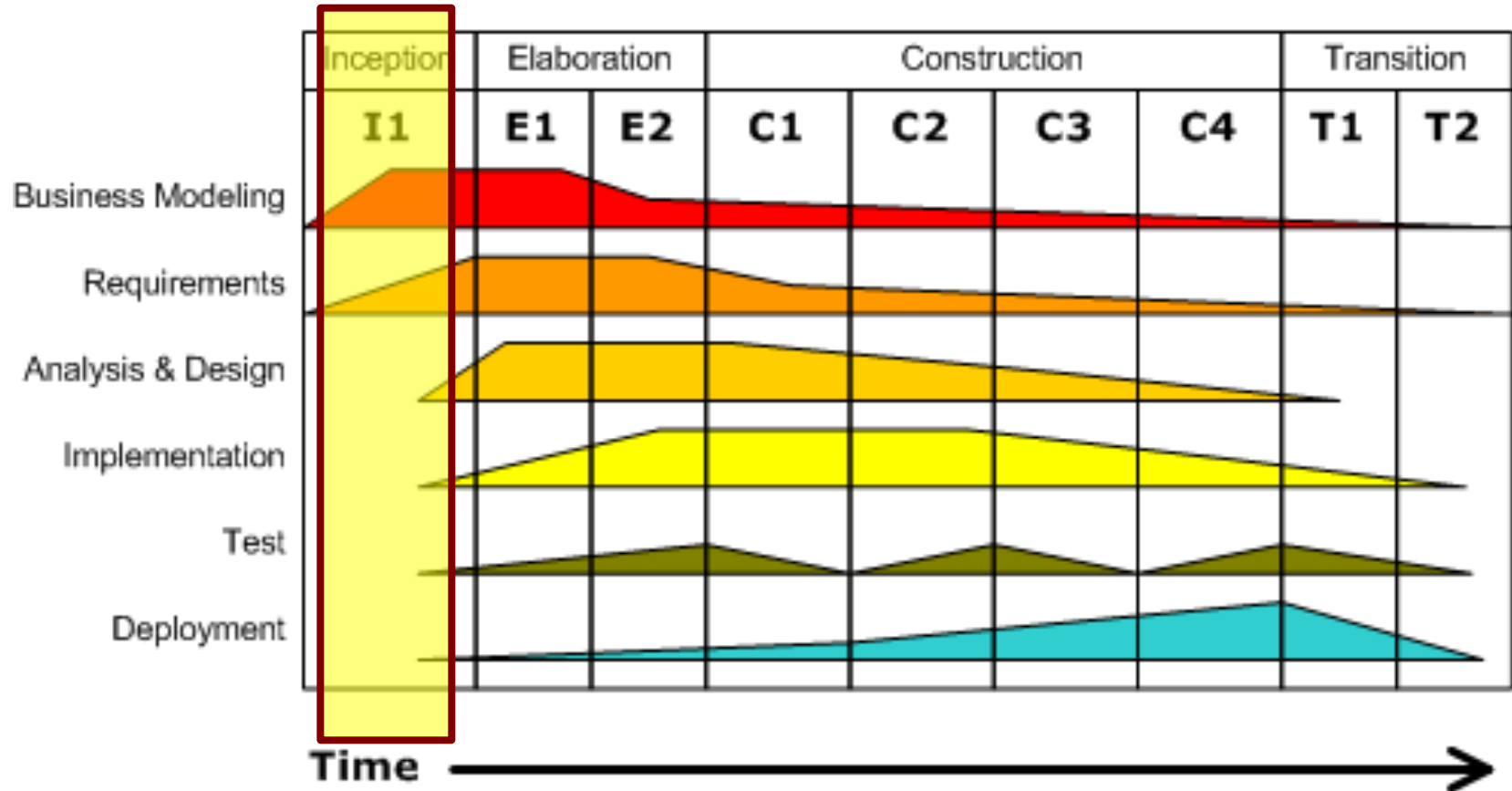


Abordagem em DSS?

- Processo iterativo e incremental
 - *releases* frequentes com progressivamente mais funcionalidade
- Focado nos requisitos (funcionais) do cliente
 - requisitos *guiam* o desenvolvimento do sistema
- Com fases bem definidas
- Baseado na construção de modelos
 - permite planear qualidade e controlar riscos
 - facilita manutenção e suporte a alterações de requisitos
 - permite definir um “contrato” prévio entre equipa de desenvolvimento e cliente

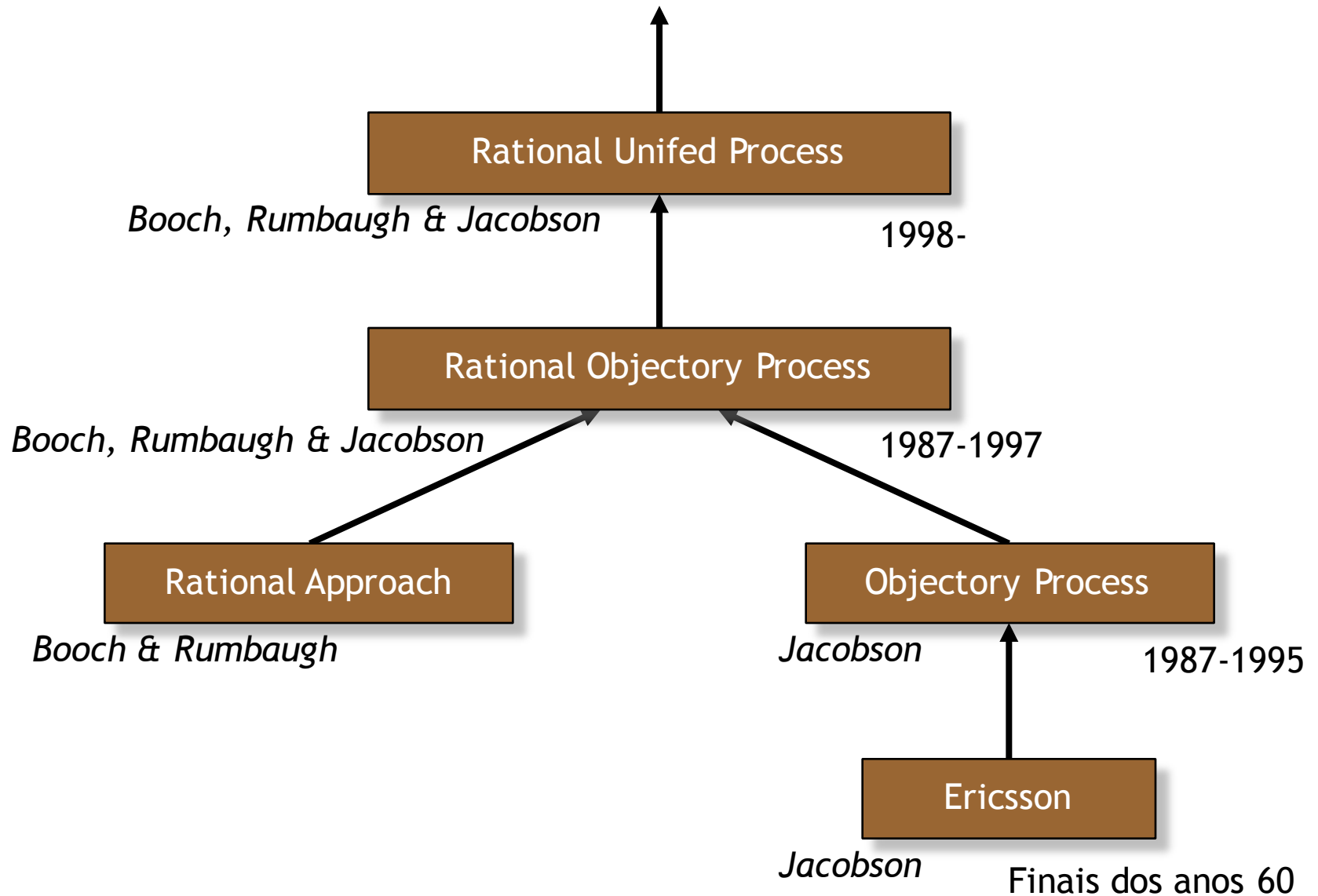


(Rational) Unified Process





Breve História do Unified Process





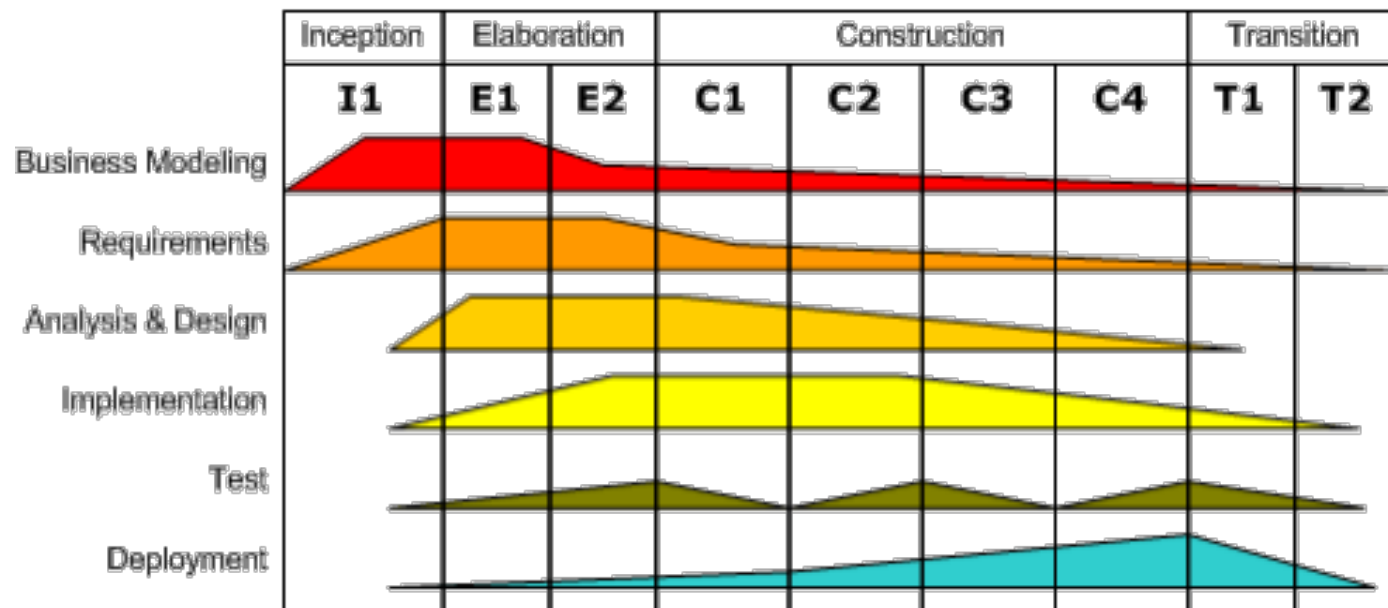
O Unified Process

- O *Unified Process* é:
 - Uma *framework* para o processo de desenvolvimento, genérica e adaptável.
- O *Unified Process* fomenta:
 - O desenvolvimento baseado em componentes.
- O *Unified Process* utiliza:
 - A UML como ferramenta de modelação durante todas as fases do processo de desenvolvimento.



Unified Process

- Três ideias chave:
 - guiado por casos de uso (*use cases*);
 - centrado na arquitetura do sistema a desenvolver;
 - iterativo e incremental:
 - Início;
 - Elaboração;
 - Construção;
 - Transição.





Unified Process

Desenvolvimento guiado por Use Cases

- Um *use case* representa uma interacção entre o sistema e um humano ou outro sistema (uma *funcionalidade*);
- O modelo de *use cases* é utilizado para:
 - guiar a concepção do sistema (captura de requisitos funcionais);
 - guiar a implementação do sistema (implementação de um sistema que satisfaça os requisitos);
 - guiar o processo de testes (testar que os requisitos são satisfeitos).



Unified Process

Desenvolvimento centrado na arquitectura

- O modelo de *use cases* descreve a função do sistema, o modelo da arquitectura descreve a forma;
- O modelo arquitectural permite tomar decisões sobre:
 - O estilo de arquitectura a adoptar;
 - Quais os componentes do sistema e quais as suas interfaces;
 - A composição de elementos estruturais e comportamentais.



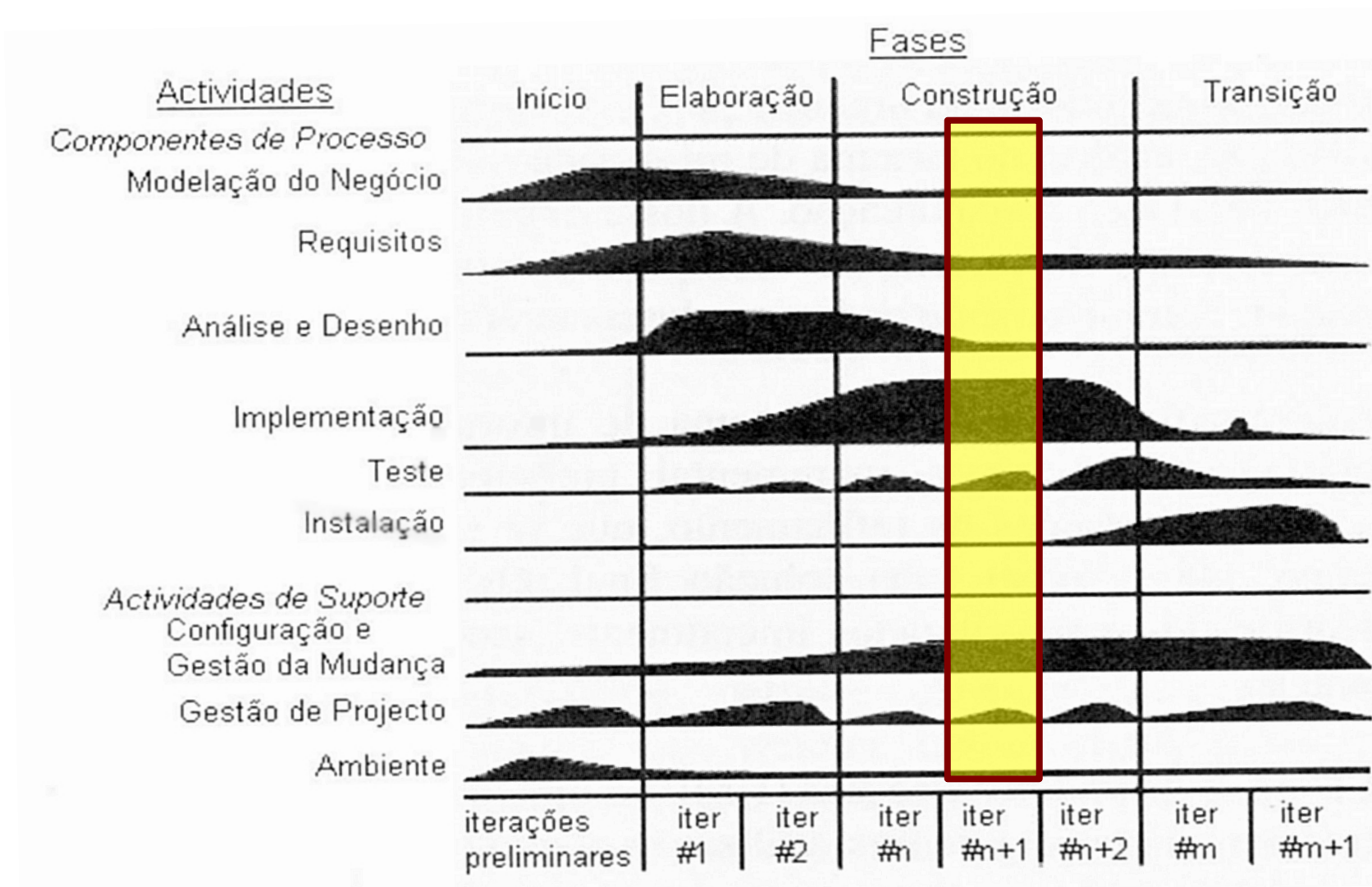
Unified Process

Desenvolvimento iterativo e incremental

- Permite dividir o desenvolvimento em “pedaços geríveis”;
- Em cada iteração:
 - identificar e especificar *use cases* relevantes;
 - criar uma arquitectura que os suporte;
 - implementar a arquitectura utilizando componentes;
 - verificar que os *use cases* são satisfeitos.
- Selecção de uma iteração:
 - grupo de *use cases* que extendam a funcionalidade;
 - aspectos de maior risco.



Ciclo de vida do Unified Process

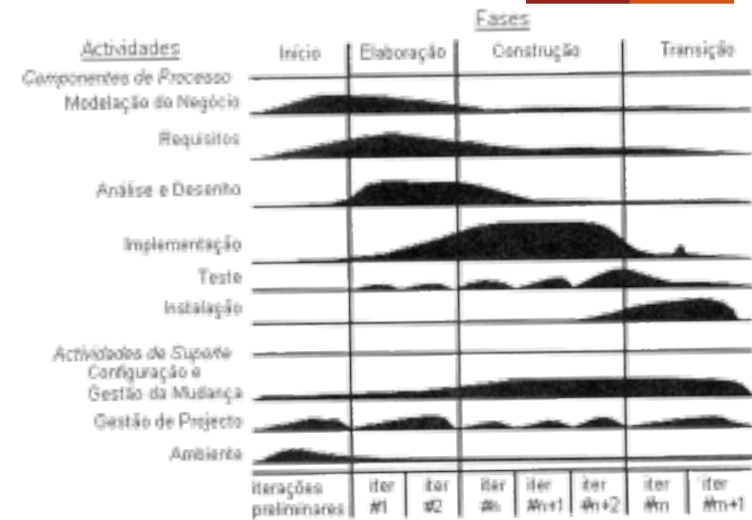




Fases do Unified Process

Início

- Identificar o problema.
- Definir âmbito e natureza do projecto.
- Fazer estudo de viabilidade.



Resultado da fase: decisão de avançar com o projecto.

Elaboração (Análise / Concepção Lógica)

- Identificar o que vai ser construído (quais os requisitos?).
- Identificar como vai ser construído (qual a arquitectura?).
- Definir tecnologia a utilizar.

Resultado da fase: uma arquitectura geral (conceptual) do sistema.



Fases do Unified Process

Construção (Concepção Física/Implementação)

- Processo iterativo e incremental.
- Em cada iteração tratar um (conjunto de) Use Case:
análise / especificação / codificação / teste / integração

Resultado da fase: um sistema!

Transição

- Realização dos acertos finais na instalação do sistema.
- Optimização, formação.

Resultado da fase: um sistema instalado e 100% funcional.

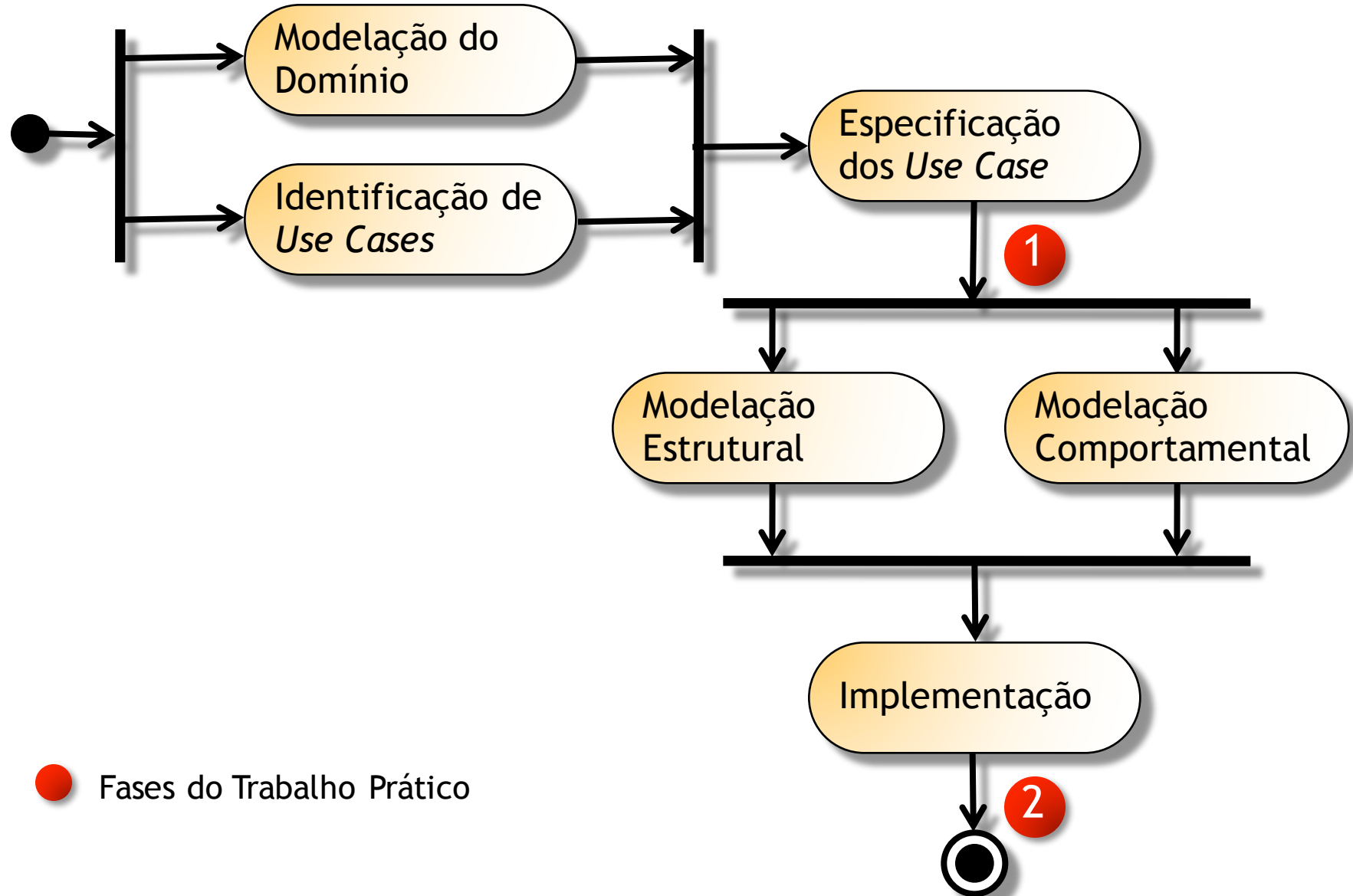


O RUP (Rational Unified Process)

- O RUP é uma concretização do Unified Process.
 - Vendida pela IBM
- O RUP fornece:
 - ferramentas de gestão do processo de desenvolvimento segundo a framework definida pelo Unified Process;
 - ferramentas para a modelação e desenvolvimento baseadas no UML;
 - uma base de conhecimento (*knowledge base*).
- Na verdade as coisas passaram-se um pouco ao contrário...
(primeiro as ferramentas e depois o processo...)



Abordagem em DSS





Unified Software Development Process

Sumário:

- Processos de desenvolvimento de software
 - Sequenciais - modelo em cascata
 - Iterativos - modelo em espiral; Unified Process
 - Ágeis - Extreme programming
- Abordagem em DSS
 - O Unified Process
 - O ciclo de vida do Unified Process
 - O RUP (Rational Unified Process)