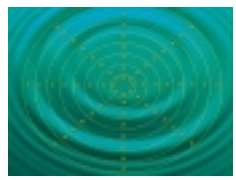
O Processo de Desenvolvimento de Software

Estruturar o desenvolvimento de software

Ciclo de vida

Metodologias



Ciclo de Vida

O processo de desenvolvimento de software pode, numa visão genérica, ser estruturado em três fases distintas que correspondem ao seu **ciclo de vida**:

- Fase de **definição**, ou concepção inicial do produto
- Fase de **desenvolvimento**
- Fase de **manutenção**, que decorre desde a entrega ao cliente até ao envelhecimento do produto

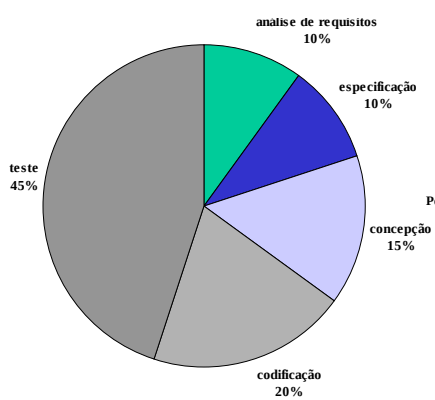
Estas três fases existem em qualquer projecto, independentemente da sua área de aplicação, dimensão ou complexidade

Ciclo de Vida - Processo de Software

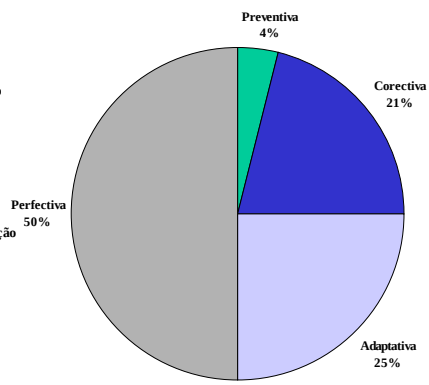
- **São o conjunto de actividades e resultados associados que produzem um produto de software.**
- **Os processo vistos atrás podem ainda refinar-se:**
  - À fase de desenvolvimento, pode acrescentar-se o processo de validação. O software deve ser validado de forma a que seja assegurado que fará o que o cliente deseja.
  - À fase de manutenção é também associado o processo de evolução do software.O software deve evoluir para responder às alterações das necessidades do cliente.

Ciclo de Vida - Esforço e Manutenção

Esforço Relativo das Actividades de Desenvolvimento de Software



Distribuição das Actividades de Manutenção



Fase de Definição

Identifica-se o **problema**: que informação deve ser processada, que funções e desempenho são pretendidos, que interfaces são necessárias, que restrições devem ser consideradas e que critérios devem ser utilizados na avaliação do projecto

Tipicamente, engloba três tipos de tarefas:

- **Análise do sistema** - Papel do produto no sistema global (software, hardware)
- **Análise de requisitos** - Identificação das funções a realizar pelo software
- **Planeamento do projecto** - Análise dos riscos, custos e recursos alocados pelo projecto, definição de tarefas e plano de execução

Fase de Desenvolvimento

Identifica-se a **solução**: como é que as estruturas de dados, arquitectura do software e funções serão realizadas; como é que o desenho se traduzirá numa linguagem de programação; e como serão efectuados os testes do produto

Tipicamente, engloba três tarefas:

- **Desenho -** Tradução dos requisitos num conjunto de representações (texto, gráficos) que descrevem a estrutura de dados, arquitectura e funções
- **Codificação -** Tradução do desenho em instruções
- **Teste -** Procura e eliminação de defeitos na funcionalidade do produto

Fase de Manutenção

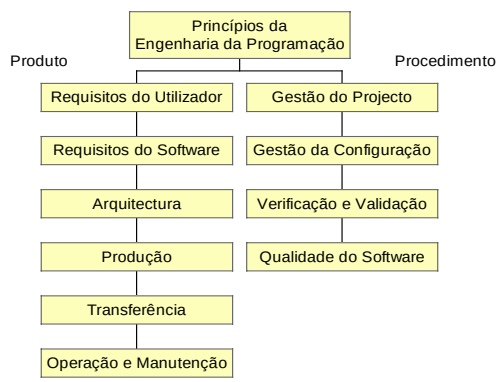
Foca nas **alterações do software**, devidas a erros não detectados nas fases anteriores ou alterações propostas pelo cliente. Volta a aplicar as fases de definição e desenvolvimento mas partindo do código já desenvolvido

Tipicamente, engloba três tipos de tarefas:

- **Correcção -** Eliminação de erros
- **Adaptação -** Modificação do software devido a alterações no ambiente (software, hardware)
- **Evolução -** Extenção do software a pedido do cliente

Ciclo de Vida

Visão detalhada (recomendada pelo IEEE)



Metodologias

Como devem ser organizadas as actividades que levam à realização do produto

- Considerar todo o ciclo de vida do produto: desde a concepção inicial até ao seu envelhecimento
- Definir um processo de produção
- Aplicar os princípios do desenvolvimento de software

- **Metodologias**:
  - Codificar e reparar (*code-and-fix*)
  - Cascata
  - Evolutiva
  - Transformação
  - Espiral

Codificar e Reparar

Metodologia primitiva:

- Anos 50
- Linguagem de baixo nível
- Não havia distinção entre programador e utilizador
- Problemas simples e bem conhecidos

Desenvolvimento do processo em dois passos:

- Escrever código
- Modificar código, de forma a reparar erros, melhorar ou acrescentar funcionalidades

Codificar e Reparar

Avaliação crítica:

- Após uma sequência de alterações o código torna-se tão confuso que alterações subsequentes são cada vez mais difíceis e imprevisíveis
- Hoje em dia, os domínios de aplicação são cada vez mais complexos, pelo que os problemas são cada vez menos compreendidos
- Existe hoje uma distinção clara entre utilizador e programador
- O software tem de ser desenvolvido em grupo
- Um produto precisa de marketing, ser vendido e instalado em diferentes máquinas

Falhas do modelo:

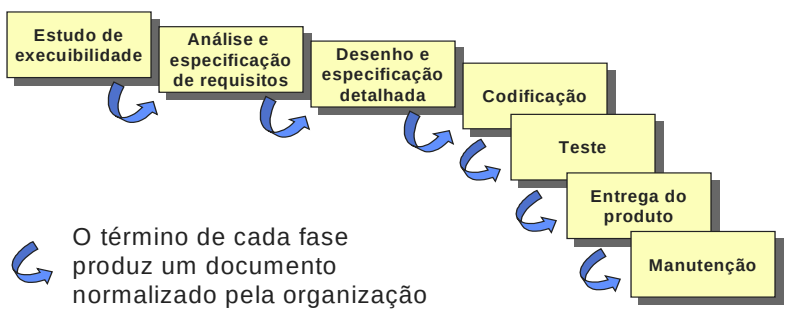
- Dificuldade em gerir a complexidade de modo não estruturado
- Falta de documentação
- Descoberta que o produto não cumpria os objectivos dos utilizadores
- Impossibilidade de controlar o processo

Cascata

Metodologia seguinte:

- Popularizada nos anos 70
- Elaborada num projecto militar Americano

Desenvolvimento do processo numa cascata de fases:



O término de cada fase produz um documento normalizado pela organização

Estudo de Exequibilidade

Objectivo

- Avaliar os custos e benefícios do produto a desenvolver. Avaliar se o projecto deve ser ou não executado

- Depende do tipo de produto
- Tipicamente vago, realizado com poucos recursos e sob pressão
- Contém:
  - Definição do problema
  - Soluções alternativas
  - Recursos necessários, custos, prazos de entrega para cada solução alternativa

Análise e Especificação de Requisitos

Identificar as qualidades requeridas para o produto

- Funcionalidade, desempenho, portabilidade…

- Indicar **que** qualidades são necessárias e não **como** as obter
- Desta fase resulta um documento de especificação de requisitos
  - Analisado e confirmado pelo cliente
  - utilizado para desenvolver uma solução que realize os requisitos
- Quem realiza a análise de requisitos deve seguir os princípios do software:
  - Separação de preocupações, abstracção e modularização

Desenho e Especificação Detalhada

Decomposição do sistema em módulos

- Decomposição lógica (desenho de alto nível)
- Decomposição física (definição de estruturas de dados e algoritmos)
- Processo iterativo do topo para a base

- Desta fase resulta um documento com:
  - Arquitectura do software
  - Funcionalidade de cada módulo
  - Interdependências entre módulos

Codificação

Escrita do software, utilizando uma linguagem de programação

- A codificação pode seguir padrões definidos pela organização (cabeçalhos, comentários, convenções de nomes, etc.)

- Resultado desta fase:
  - Módulos de software

Teste, Entrega e Manutenção

O teste do produto inclui

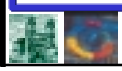
- Teste individual de cada módulo
- Teste de integração dos diversos módulos
- Teste global do sistema

A entrega do produto segue habitualmente duas fases:

- Versão beta - Entregue segundo condições controladas
- Versão oficial - Distribuída sem restrições

A manutenção do produto considera normalmente:

- ~20% manutenção correctiva
- ~20% manutenção adaptativa
- >50% manutenção evolutiva
- 42% alterações de requisitos dos utilizadores, 17% alterações no formato de dados, 12% emergências, 9% debug, 6% alterações no hardware, 5% melhoramentos na documentação, 4% melhoramento do desempenho

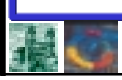


Cascata

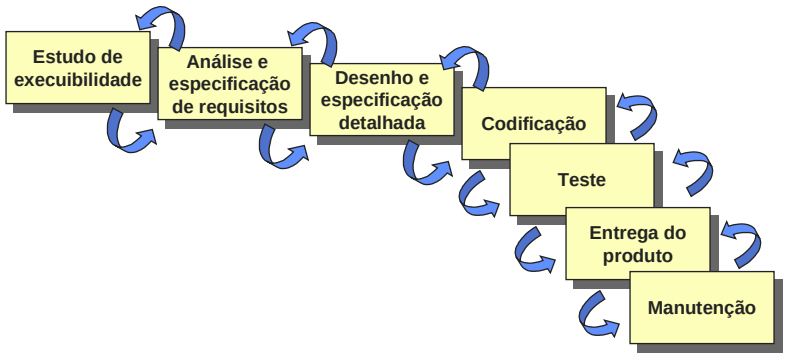
Avaliação crítica:

- Impõe uma disciplina ao desenvolvimento de software
- Cada fase exige um documento a especificar entradas e saídas
- As fases iniciais exigem o planeamento do processo

- Duas contribuições importantes:
  - O desenvolvimento de software deve ser disciplinado, planeado e gerido
  - A realização do produto deve ser adiada até que os objectivos sejam plenamente conhecidos
- Trata-se de um modelo ideal, apenas aproximável na prática
  - Rígido - Pressupõe que o desenvolvimento segue linearmente da análise até à codificação e teste. Os resultados de cada fase não podem ser alterados posteriormente
  - Monolítico - O produto só existe no fim do processo



Cascata com Retorno



- Permite voltar atrás no processo, para realizar modificações e adaptações
- Mantém a disciplina do processo

Cascata

Continuação da avaliação crítica:

- Difícil estimar recursos
- A análise de requisitos só é efectivamente validada após a entrega do produto
- Os utilizadores não sabem antecipadamente todos os requisitos da aplicação
- Metodologia não antecipa as modificações ao software
- Baseia-se na produção de documentos em alturas específicas, preconizando um processo burocrático

- Falhas do modelo:
  - Elevados custos de manutenção
  - Rápida divergência entre a documentação e o produto, após a entrega do produto
  - Não aplicável a produtos instáveis, com funcionalidades desconhecidas a priori ou peso elevado das interfaces utilizador

Cascata

- **De salientar que cada fase inclui a chamada verificação e validação (V & V)**
  - a verificação significa: o sistema corresponde aos requisitos (estamos a construir o sistema correcto?)
  - a validação significa: o sistema responde aos requisitos do clientes?

  Há uma ênfase considerável numa análise cuidadosa antes do sistema ser efectivamente construído
  - tentam identificar-se e cimentar os requisitos do utilizador tão cedo quanto possível
  - estes requisitos são documentados através do da especificação de requisitos

Cascata

- **Outra deficiência:**
  - em muitos dos projectos de desenvolvimento de software, a sequência estrita de fases advogada pelo modelo em cascata, não é realmente obedecida; há realmente uma sobreposição de actividades.

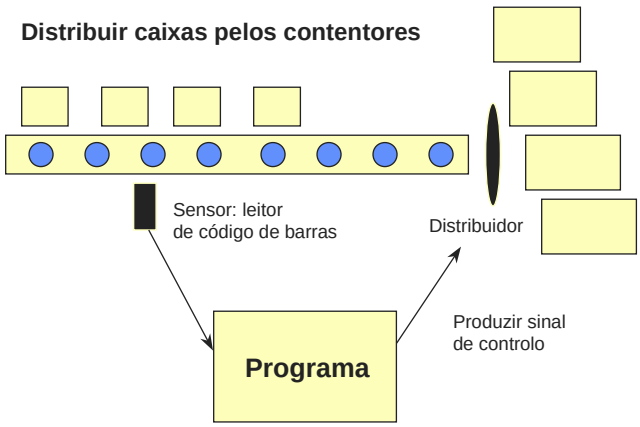
Cerca de 16% do esforço de concepção ocorre realmente depois do sistema ser suporto estar acabado

Só 50% do esforço de concepção, ocorre realmente durante a fase de concepção

Activity	Phase			
	Design	Coding	Integration testing	Acceptance testing
Integration testing	4.7	43.4	26.1	25.8
Coding	6.9	70.3	15.9	6.9
Design	49.2	34.1	10.3	6.4

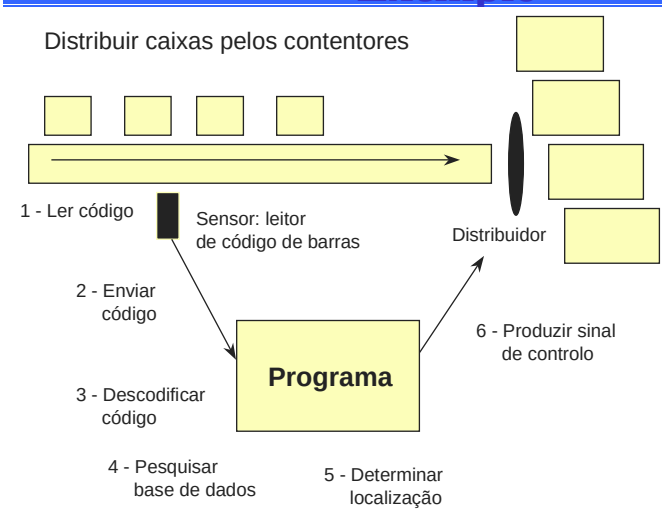
Exemplo

Distribuir caixas pelos contentores



Exemplo

Distribuir caixas pelos contentores

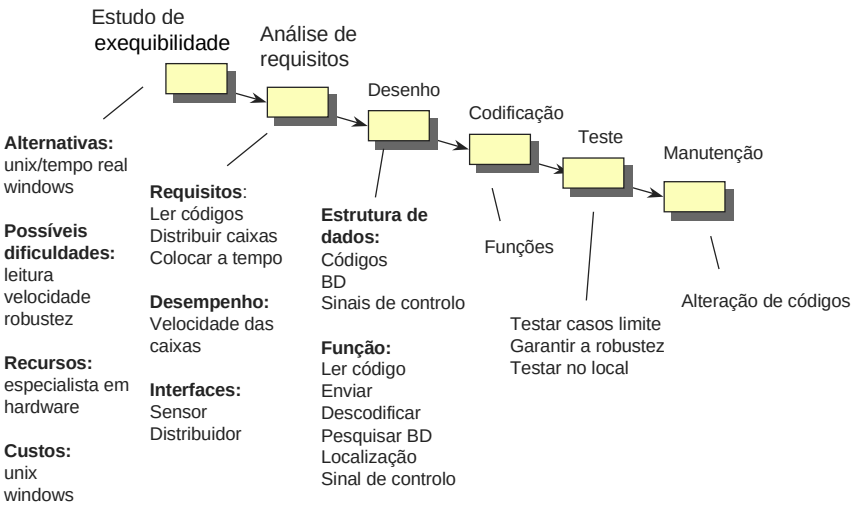


Recursos:
Sensor
Distribuidor
Base de dados
MS Access
MS Developer Studio

Restrições:
Velocidade das caixas
Velocidade do sensor
Robustezdo sistema

Interfaces:
Leitor de códigos
Distribuidor
Base de dados

Exemplo



Metodologia Evolutiva

- A primeira versão do produto é vista como uma tentativa (protótipo) destinada a avaliar a execuibilidade e verificar os requisitos
- Esta versão deve ser deitada fora, sendo recomeçado o desenvolvimento do produto em bases mais sólidas
- O segundo desenvolvimento segue o modelo em cascata

Desenvolvimento do processo:

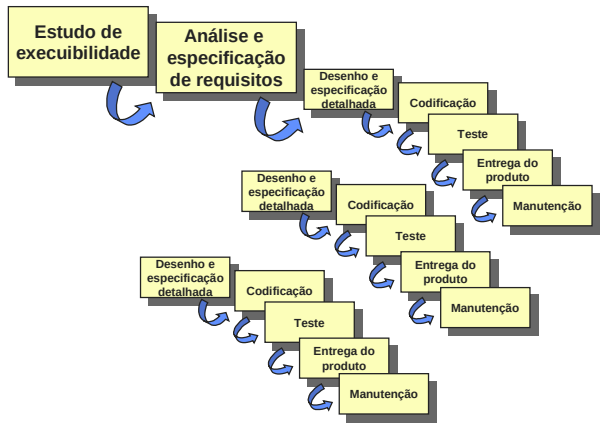
  - Entregar algo ao utilizador
  - Receber informação de retorno do utilizador
  - Ajustar os objectivos e desenho do produto

- Duas alternativas
  - Desenvolvimento e entrega incrementais
  - Prototipagem

Desenvolvimento e Entrega Incrementais

O produto é desenvolvido desde o início, mas de modo incremental:

- Alguns módulos do sistema são desenvolvidos primeiro
- Novos módulos são acrescentados de modo suave
- Sequência de mini-cascatas
- O ciclo de vida do produto assemelha-se a uma manutenção interminável



Desenvolvimento e Entrega Incrementais

- **O Objectivo do processo é trabalhar com o cliente de forma exploratória, para encontrar os requisitos e entregar um sistema final de acordo com as necessidades do cliente.**
- **Normalmente são desenvolvidos subconjuntos da aplicação cujos requisitos estão bem compreendidos, permitindo:**
  - obter feed-back o mais cedo possível, permitindo que aplicação evolua de forma controlada
  - uma entrega mais rápida de parte do produto ao cliente; pode assim contribuir para uma melhor oportunidade
- **Evita-se o efeito "Big Bang":**
  - Por um tempo longo nada acontece, de repente, há uma situação completamente nova
- **Em vez de construir o software, o software cresce**
- **O utilizador é envolvido de perto no planeamento do próximo passo**

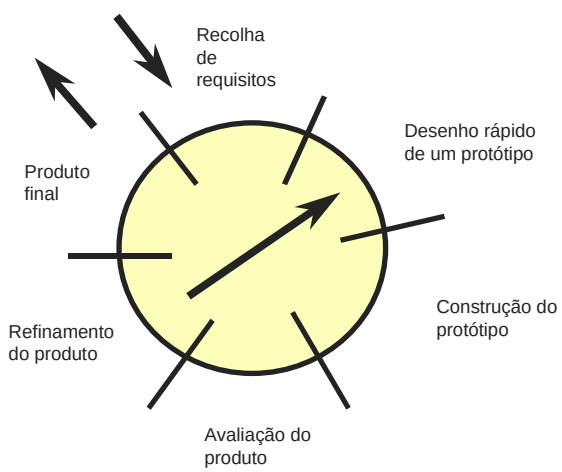
Desenvolvimento e Entrega Incrementais

- **Redireccionar o projecto torna-se mais fácil, dado que poderemos antecipar as alterações nas circunstâncias mais depressa**
- **Combate-se o síndroma da sobrefuncionalidade:**
  - dado que o utilizador tem dificuldade em formular as suas necessidades, tende a pedir demais, pensando que tudo é fácil de fazer e que tudo pode ser realizado, daí:
    - podem despender-se enormes esforços na implementação de características completamente inopinadas
    - muitas funcionalidades, não é sinónimo de adaptação às tarefas em mãos
    - maior dificuldade na utilização do produto, dada a muito maior complexidade (dada a riqueza de funcionalidades)
- **Assim:**
  - a atenção é focada nas **características essenciais**
  - as funcionalidades adicionais são incluídas **só e quando** são necessárias.

Prototipagem

A versão inicial é progressivamente transformada no produto final:

- Inicialmente, alguns módulos simulam a funcionalidade pretendida
- Os componentes reais são desenvolvidos progressivamente
- O protópipo é visto como um modo de capturar os requisitos dos utilizadores



Prototipagem

Protótipo pode ser definido como "modelo de trabalho de (possivelmente partes) de um sistema de software, que dá ênfase a certos aspectos"

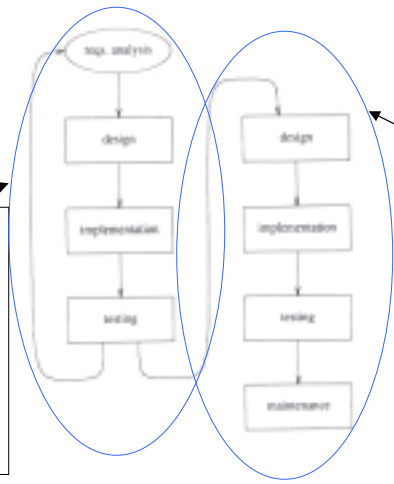
- **desenvolvimento do modelo relativamente barato**
- **rápido**
  - empregar linguagens de muito alto nível (que eventualmente produzam uma versão não eficiente, mas que possa ser utilizada para testar a funcionalidade do sistema)
    - utilizáveis, particularmente em aplicações orientada a dados, aquelas com ênfase considerável na interface ou que mostrem um grau elevado de interacção como utilizador
  - desenvolvimento de um sistema com menor funcionalidade, em particular no que se relaciona com atributos de qualidade:
    - velocidade
    - robustez
    - e outros semelhantes

Prototipagem x Abordagem Tradicional

Em experiências realizadas por (Boehm84 e Alavi84), em que se comparou o desenvolvimento utilizando a abordagem tradicional versos prototipagem, concluiu-se:

- a abordagem prototipagem levou menos 40% do tempo e resultou em 45% menos código;
- a abordagem tradicional resultou num produto mais robusto, que se prevê ser de mais fácil manutenção
- utilizando protótipos na 1ª fase:
  - os utilizadores ficaram com uma apreciação positiva do sistema desenvolvido
  - os utilizadores sentiram-se mais envolvidos no processo de desenvolvimento
  - os utilizadores tiveram menos conflitos com a concepção
  - houve dificuldades no controlo do processo de desenvolvimento

Prototipagem



relacionada com as fases da prototipagem;

a iteração corresponde ao processo de validação do utilizador

requisitos novos ou alterados vão ser considerados no ciclo seguinte

Produção efectiva do sistema

(já não haverá o relaxamento das propriedades que assegurarão a qualidade do software desenvolvido)

Metodologia Evolutiva

Avaliação crítica:

- O processo é evolutivo e não iterativo
- O desenho do produto é baseado em resultados experimentais
- Resolve os problemas da rigidez e monolitismo do modelo cascata (permite alterações à especificação e evita que o produto apenas exista no fim do processo)
- Adequado a software com elevado peso das interfaces utilizador
- Falhas do modelo:
  - existe a possibilidade de a metodologia evolutiva se tornar na codificar e reparar
  - Para além disso, no modelo de prototipagem:
    - O cliente pensa que tem um produto, o que não é verdade
    - O que fazer com o protótipo?
      - Deitar fora (e ser obrigado a desenvolver um novo)
      - Atirá-lo ao cliente (e viver com os compromissos próprios de um protótipo)

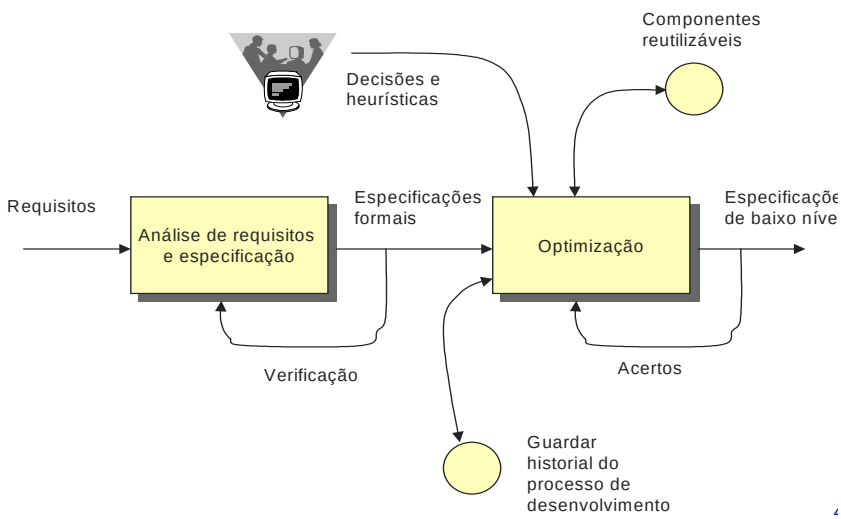
Modelo de Transformação

- Teórico, baseado em especificação formal
- O desenvolvimento pode ser visto como uma sequência de passos que transformam gradualmente a especificação na realização

Desenvolvimento do processo:

- Analisar requisitos informais e especificar formalmente as funções do produto
- Transformar a especificação formal numa descrição mais detalhada e menos abstracta
- Progressivamente, a descrição torna-se executável, utilizado um processador abstracto

Modelo de Transformação



Modelo de Transformação

Avaliação crítica:

- Grande capacidade para suportar a evolução do software
- As alterações no software são integradas no processo, desde os requisitos até à codificação
- Cria um historial do processo de desenvolvimento
- Utiliza suporte computacional ao longo de todo o processo
- Geração automática de código
- Falhas do modelo:
  - Ambientes disponíveis são incompletos
  - É tão difícil especificar formalmente como fazer código
  - O processo não é totalmente mecânico, exigindo conhecimentos e criatividade dos programadores
  - Apenas funciona para projectos pequenos

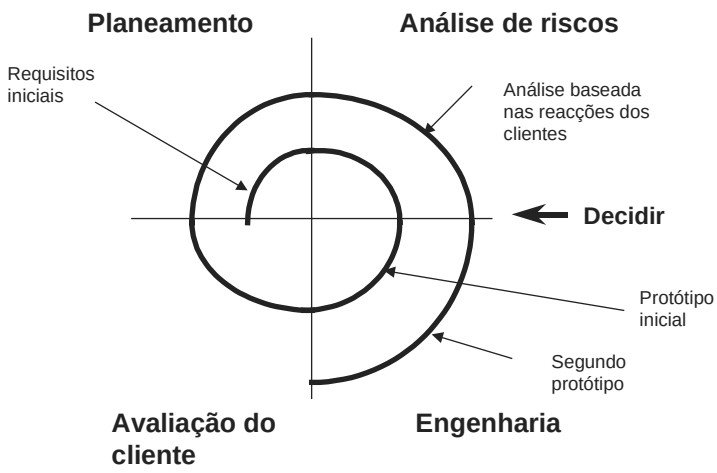
Modelo em Espiral

- Guiar o processo de desenvolvimento de software de acordo com os níveis de risco a ele associados
- Pode ser visto como um meta-modelo, pois pode acomodar os modelos anteriores

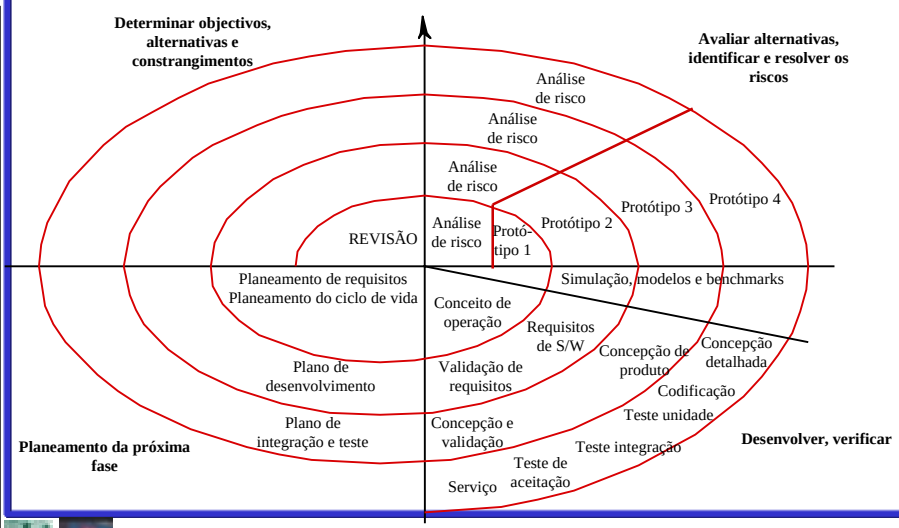
Desenvolvimento do processo:

- Analisar os riscos do projecto
  - **Riscos** - Circunstâncias adversas que podem impedir o desenrolar do processo ou reduzir a qualidade do produto
  - **Gestão dos riscos** - Identificar, englobar e eliminar os riscos antes de estes poderem pôr o projecto em risco
- Aplicar ciclicamente a análise de riscos

Modelo em Espiral (Simplificado)

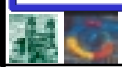


Modelo em Espiral



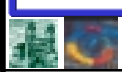
Sectores p/ cada Loop no Modelo em Espiral

- **Estabelecimento de Objectivos**
  - os objectivos para a fase do projecto são identificados; são identificados riscos e constrangimentos do produto ou processo; possível planeamento de estratégias alternativas
- **Identificar e resolver riscos**
  - os risco chave são identificados, analisados e é obtida informação para reduzir esses riscos (p. ex. construção de protótipo)
- **Desenvolvimento e validação**
  - é escolhido o modelo apropriado para a próxima fase de desenvolvimento, de acordo com os riscos identificados; ex. se a interface de utilizador constitui o risco, um modelo evolucionário deverá ser apropriado
- **Planeamento**
  - o projecto é revisto e feitos planos para a próxima volta da espiral, se avaliada como necessária



Gestão de Riscos

- **A distinção mais importante entre o modelo em espiral e outros modelos de processo de software é a consideração expressa de risco**
- **Informalmente, o risco pode ser definido como algo que pode ir mal**
  - Riscos são uma consequência de informação inadequada;
  - São resolvidos através de acções que permitam descobrir informação que reduza a incerteza.



Gestão de Riscos

- **Um ciclo da espiral tem início com a elaboração de objectivos como:**
  - desempenho, funcionalidade, etc.
- **Formas alternativas de atingir esses objectivos e constrangimentos impostos por cada alternativa são enumerados;**
- **Cada alternativa é avaliada em função de cada objectivo;**
- **São assim identificadas fontes possíveis de riscos;**
- **Segue-se análise mais detalhada através de prototipagem, simulação, etc.**

Exemplo para uma Revolução no Modelo e em Espiral

- **Objectivos - alvos da análise**
- **Constrangimentos - factores que limitam as possibilidades**
- **Alternativas - diferentes formas possíveis de atingir os objectivos**
- **Riscos - riscos possíveis com alternativas identificadas**
- **Resolução dos Riscos - estratégias usadas para reduzir os riscos identificados**
- **Resultados - resultados das estratégias de resolução de risco**
- **Planos - como tratar a próxima fase da análise**
- **Acordo - decisões da gestão acerca da forma de continuar**

exemplo: Melhoria de Qualidade

- **Objectivos**
  - melhorar significativamente a qualidade do software produzido
- **Constrangimentos**
  - prazo de 3 anos
  - sem grandes investimentos de capital
  - sem alterações radicais nos standards da empresa
- **Alternativas**
  - reutilização de software certificado
  - introduzir especificação e verificação formal
  - investir em ferramentas de teste e validação

exemplo: Melhoria de Qualidade

- **Riscos**
  - não é possível melhoria de qualidade a baixos custos
  - melhorias de qualidade podem aumentar os custos excessivamente
  - novos métodos podem levar a que funcionários saiam
- **Resolução de riscos**
  - pesquisa na literatura de caso idêntico
  - projecto piloto
  - pesquisa de potenciais componentes reutilizáveis
  - avaliação de ferramentas de suporte disponíveis
  - formação de pessoal e seminários de motivação

exemplo: Melhoria de Qualidade

- **Resultados**
  - a experiência em métodos formais é limitada - é muito difícil quantificar as melhorias
  - disponibilidade limitada de ferramentas de suporte para o sistema standard de desenvolvimento da empresa
  - componentes reutilizáveis disponíveis, mas poucas ferramentas de suporte para a reutilização
- **Planos**
  - explorar a opção de reutilização em mais detalhe
  - desenvolver um protótipo de uma ferramenta de suporte à reutilização
  - explorar um esquema de certificação de componentes
- **Acordo**
  - conseguiram-se fundos para uma nova fase de estudo de 12 meses

exemplo: Catálogo de Software

- **Objectivos**
  - encontrar catálogo de componentes de software
- **Constrangimentos**
  - prazo de um ano
  - deve suportar tipos de componentes existentes
  - custo total inferior a 20000c
- **Alternativas**
  - adquirir software de pesquisa de informação
  - adquirir uma base de dados e desenvolver um catálogo utilizando essa base de dados
  - desenvolver um catálogo específico

exemplo: Catálogo de Software

- **Riscos**
  - **Pode tornar-se impossível atingir o objectivo dados os constrangimentos**
  - **A funcionalidade do catálogo pode ser desapropriada**
- **Resolução dos riscos**
  - **Desenvolver um protótipo do catálogo (usando 4GL e uma SGBD disponível) para clarificar os requisitos**
  - **Contratar relatórios de consultores relativos a capacidades de sistemas de pesquisa de informação**
  - **Aumentar os constrangimentos temporais**

exemplo: Catálogo de Software

- **Resultados**
  - os sistemas de pesquisa de informação são inflexíveis. Não respondem aos requisitos identificados
  - um protótipo utilizando SGBD pode ser melhorado para completar o sistema
  - o desenvolvimento de um catálogo específico ultrapassa de longe o orçamento
- **Planos**
  - desenvolver um catálogo utilizando um SGBD existente, melhorando o protótipo e a interface com o utilizador
- **Acordo**
  - aumentado o prazo e orçamento para mais 12 meses

Modelo em Espiral

Análise crítica:

- É considerado um paradigma bastante realista
- Introduz a análise de riscos
- É igualmente um processo evolutivo, mas desenvolve-se o produto e não um protótipo
- O quadrante de engenharia pode incorporar, por exemplo, o modelo em cascata
- Falhas:
  - Muito dependente de um apertado controlo do processo

Flexibilidade do Modelo em Espiral

- **Em sistemas bem compreendidos (baixo risco técnico) - Modelo cascata. A análise de risco é relativamente barata**
- **Requisitos estáveis e especificação formal. Segurança crítica - modelo de transformação formal**
- **Alto risco, especificações incompletas - modelo de prototipagem**
- **Modelos híbridos podem ser acomodados para partes diferentes do projecto**

Vantagens do Modelo em Espiral

- Foca a atenção nas opções de reutilização
- Foca a atenção na eliminação prematura dos erros
- Dá relevo aos objectivos de qualidade
- Integra o desenvolvimento e manutenção
- Proporciona um referencial para o desenvolvimento de hardware / software

Problemas do Modelo em Espiral

- Muitas vezes, os contratos de desenvolvimento especificam antecipadamente o modelo de desenvolvimento e entregas
- Requer experiência na avaliação de riscos
- Necessita de refinamento para utilização genérica

Visibilidade do Processo

- **Os sistemas de software são intangíveis. Dessa forma, os gestores necessitam de documentos para avaliar do progresso**
- **Contudo, isto pode causar problemas**
  - desfasamento entre o momento de entrega de entregas relativas ao progresso das tarefas e o tempo necessário a completar as actividades
  - a necessidade de produzir documentos coloca constrangimento à iteração do processo, pois que se forem descobertos problemas durante o processo, são muitas vezes adoptadas soluções deselegantes para evitar a alteração de documentos já aprovados
  - o tempo que leva ao acordo de revisão e aprovação de documentos é significativo, levando a que o desenvolvimento continue antes do documento relativo à fase anterior ser revisto e aprovado
- **O modelo cascata é ainda o modelo baseado em entregas mais utilizado**

Documentos do Modelo Cascata

Actividade	Documentos de Saída
Análise de requisitos	Estudo de exequibilidade
Definição de requisitos	Documento de requisitos
Especificação de sistema	Especificação funcional, plano de teste de aceitação e draft do manual do utilizador
Concepção arquitectural	Especificação arquitectural, plano de teste do sistema
Concepção de interface	Especificação de interface, plano de teste de integração
Concepção detalhada	Especificação de concepção, plano de teste de unidade
Codificação	Código do programa
Teste de unidade	relatório de teste de unidade
teste de módulos	relatório de teste de módulos
Teste de integração	Relatório de teste de integração, manual final do utilizador
Teste de sistema	Relatório de teste do sistema
Teste de aceitação	Sistema final mais documentação

Visibilidade dos Modelos dos Processos

Modelo do processo	Visibilidade do processo
Modelo queda de água	Boa visibilidade, cada actividade produz uma entrega
Desenvolvimento evolucionário	Visibilidade pobre, não é económico produzir documentos durante iterações rápidas
Transformação formal	Boa visibilidade; os documentos devem ser produzidos a cada fase para que o processo continue
Desenvolvimento orientado à reutilização	Visibilidade moderada, pode ser artificial produzir documentos a descrever a reutilização e componentes reutilizados
Modelo espiral	Boa visibilidade, cada segmento e cada anel da espiral deve produzir algum documento

Metodologias

Apreciação final

- **Codificar e reparar** - Na realidade, não é modelo nenhum, pois não há qualquer planeamento do processo
- **Cascata** - Cai no outro extremo: rígido, não adaptativo, monolítico baseado na documentação do processo
- **Evolutivo** - Se o anterior é baseado na documentação, este é baseado na incrementalidade
- **Transformação** - Baseado na especificação
- **Espiral** - Baseado na análise de riscos

Metodologias

Apreciação final

- Resultados experimentais demonstraram:
  - Comparativamente aos modelos de prototipagem e transformação, o modelo de cascata melhora o controlo sobre o processo e o produto
  - Comparativamente aos modelos de transformação e cascata, o modelo de prototipagem adapta-se melhor à construção de interfaces utilizador
  - A produtividade dos projectos que utilizaram prototipagem e cascata foi semelhante, mas o projecto evolucionário reduz o tempo de desenvolvimento em 40% e o código em 40%
  - O processo em cascata teve menos problemas de *debug* e integração
  - Existe um consenso de que o modelo cascata deve ser substituído pro uma aproximação mais flexível