Unidade 1

1. O desenvolvimento de software

O desenvolvimento de software é uma tarefa altamente complexa. Para termos uma idéia da complexidade intrínseca a esta atividade, vale a pena analisar alguns números do Chaos Report (1994).

- Porcentagem de projetos que terminam dentro do prazo estimado: 10%
- Porcentagem de projetos que são abandonados antes de chear ao fim: 25%
- Porcentagem de projetos acima do custo esperado: 60%
- Atraso médio do desenvolvimento de projetos: 1 ano.

Os últimos dados publicados pelos Chaos Report (2009)[1] mostrou que:

- Porcentagem dos projetos que fracassam: 24%
- Porcentagem dos projetos entregues dentro do prazo com sucesso parcial: 44%
- Apenas 32% dos projetos obtiveram sucesso.

Os principais fatores que ajudaram no sucesso dos projetos foram:

- Envolvimento do usuário: 15.9%
- Apoio executivo: 13.9%
- Declaração de requisitos clara e limpa: 13%
- Planejamento apropriado: 9.6%
- Expectativas realistas: 8.2%
- Milestones pequenos: 7.7%
- Equipe competente: 7.2%
- Propriedade: 5.3%
- Visão e objetivos claros: 2.9%
- Trabalho duro e equipe focada: 2.4%
  Outros: 13.9%

O estudo mostrou também que quanto maior o tamanho do projeto, maior a probabilidade de fracasso.

Tentativas de lidar com esta complexidade e de minimizar os problemas envolvidos no desenvolvimento de software envolvem a definição de processos de desenvolvimento de

1 Disponível em: http://www1.standishgroup.com/newsroom/chaos_2009.php

software.

Conceito: Um **processo** ou uma **metodologia** de desenvolvimento de software compreende todas as atividades necessárias para definir, desenvolver, testar e manter um produto de software.

Algumas destas atividades (num grau bem genérico de especificação) são as seguintes:

- Definir QUAIS as atividades a serem executadas ao longo do projeto;
- Definir QUANDO, COMO e POR QUEM as atividades serão executadas;
- Prover pontos de controle para verificar o andamento do desenvolvimento;
- Estabelecer os padrões a serem seguidos no desenvolvimento do software.

O desenvolvimento de software envolve, assim, vários aspectos: 1. atividades, 2. pessoas e 3. modelos. Além destes aspectos, podem ser usadas a técnica de Prototipagem e ferramentas CASE.

2. Atividades típicas de um processo de desenvolvimento de software

Existem diferentes processos (métodos) de desenvolvimento de software, e existe o consenso em relação à inexistência de um processo que possa ser caracterizado como "o melhor". A adoção de um processo é dependente de contexto.

Embora existam diversos processos de desenvolvimento, há consenso, também, em relação à necessidade de algumas atividades como parte de qualquer processo:

a) Levantamento e especificação de requisitos;
b) Análise;
c) Projeto (*Design*);
d) Implementação;
e) Testes;
f) Implantação.

Observação: As fases de Levantamento e especificação de requisitos e de Análise compõem o que chamamos de Engenharia de Requisitos.

A seguir iremos tratar cade item:

a) Levantamento e especificação de requisitos

Esta atividade é também conhecida como "**elicitação**" de requisitos, que corresponde à etapa da compreensão do problema aplicada ao desenvolvimento do software.

O principal objetivo do levantamento de requisitos consiste em que usuários e desenvolvedores tenham a mesma visão do problema a ser resolvido.

Nesta etapa, os desenvolvedores, junto aos clientes (...), tentam levantar e definir (representar) as necessidades dos futuros usuários do sistema a ser desenvolvido.

Conceito: As necessidades dos futuros usuários (ou perfis de usuários) são chamadas "**requisitos**".

Formalmente, um requisito é uma condição ou capacidade que deve ser alcançada/ oferecida pelo software ou por algum dos seus componentes. O conjunto de requisitos é determinado num contrato, padrão especificação ou em outro documento formalmente definido.

Conceito: Os primeiros requisitos são determinados a partir do **domínio**. Um domínio é uma área do conhecimento ou um ramo de atividade ao qual o software vai atender. Este conceito pode ser estendido para **domínio do problema** ou **domínio do negócio**.

Cada domínio tem normalmente associado um vocabulário técnico específico. Este vocabulário deve ser conhecido e aproveitado pelo desenvolvedor na rotulação dos objetos representados nos modelos do sistema.

O levantamento de requisitos compreende, também, um estudo exploratório das necessidades dos usuários (perfis de) e um diagnóstico do sistema atual, se ele existir.

Há várias técnicas para o levantamento de requisitos, dentre as quias podem ser citadas as seguintes:

- leitura de obras de referência da C. e da área de aplicação (domínio);
- leitura de documentação técnica (manuais, rotinas, processos padrão,...) do (sub)ambiente da organização onde o software vai ser usado;
- observação da realização da atividade fim da organização durante certo período de tempo. Uma simples visita pode induzir a uma percepção errônea. Um processo de acompanhamento desde dentro e mais longo pode tornar a compreensão do problema mais fidedigna;
- realização de entrevistas com o cliente, os diferentes perfis de usuário, especialistas no domínio e todo e qualquer perfil humano de interesse para o sistema (que gere ou se beneficie das suas saídas);
- comparação com sistemas semelhantes para reaproveitamento de parte de soluções disponíveis no mesmo domínio.

O produto do levantamento de requisitos é um documento que declara os diversos tipos de requisitos do sistema. É comum esse documento ser escrito numa notação informal (texto). As principais seções deste documento, correspondentes aos diferentes tipos de requisitos são as seguintes.

Requisitos funcionais: Definem as funcionalidades do sistema

Exemplos:

- "O sistema deve permitir que cada professor realize o lançamento de notas das turmas nas quais ele lecionou.";
- "O sistema deve permitir que um aluno realize a sua matrícula nas disciplinas oferecidas no semestre letivo.";
- "Os coordenadores da escola devem poder obter o número de aprovações, reprovações e trancamentos em cada disciplina oferecida em determinado

período.”

Requisitos não-funcionais: Declaram as características de qualidade que o sistema deve possuir.

Tipos:

- **confiabilidade**: Corresponde a medidas quantitativas exigidas, tais como tempo médio aceitável entre falhas, recuperação de falhas, quantidade de erros por linhas de código,...;
- **desempenho**: Determinam tempos de resposta esperados para as diferentes funções do sistema;
- **portabilidade**: Define as possíveis restrições sobre as plataformas de hardware e software nas quias o sistema será implantado e sobre o grau de facilidade com que ele deve poder ser transportado para outras plataformas;
- **segurança**: Determina as questões associadas à necessidade de proteção do sistema contra acessos indevidos ou não-autorizados;
- **usabilidad**e (e outros conceitos...!): Correspondem às condições que o software deve cumprir para ser fácil de usar pelos diferentes perfis de usuário esperados.

Requisitos normativos: Estabelecem restrições impostas sobre o desenvolvimento do sistema.

Exemplos:

- custos;
- prazos;
- plataforma tecnológica;
- aspectos legais (ex.: licenças);
- necessidades de comunicação com outros sistemas;
- regras do negócio (restrições ou políticas de funcionamento específicas da organização no domínio em questão).

Requisitos inovadores: São capacidades do sistema que não tinham sido explicitados pelo usuário como requisitos e que transcendem as necessidades do usuário, determinando adição de valor ao produto final, por meio da introdução de possibilidades na organização antes inexistentes sem ele.

Observações: Os requisitos funcionais, assim como os normativos, são mais facilmente expressos pelo cliente e/ou usuários potenciais. Alguns dos requisitos não-funcionais são esperados pelo cliente/usuário e, portanto, omitidos na especificação ao desenvolvedor. Os requisitos inovadores são, muitas vezes, contribuição do projetista.

Uma das formas de se medir a qualidade de um sistema é pela sua utilidade. Um sistema será útil aos seus usuários se ele atender aos requisitos definidos e se esses requisitos refletirem as necessidades reais dos usuário. (problema se a elicitação de requisitos for feita apenas junto ao cliente...!)

O documento de requisitos não deve descrever as soluções técnicas que serão adotadas. O enfoque principal deste documento deve ser a informação de resposta à pergunta “O que o usuário necessita do (novo) sistema?”. Em outras palavras, os requisitos definem o problema a ser resolvido, não os programas que os resolvem.

O levantamento de requisitos é a etapa mais importante do ponto de vista do retorno em investimento. Muitos sistemas são abandonados ou nem chegam a ser usados porque os membros da equipe não dispensaram tempo suficiente para compreender as necessidades do cliente (conjunto de perfis de usuário!).

O documento de requisitos deve servir como um termo de consenso entre a equipe técnica e o cliente (conjunto de perfis de usuário!). Ele constitui a base para as atividades subseqüentes de desenvolvimento do sistema e proporciona um ponto de referência para qualquer validação posterior do software construído.

O documento de requisitos estabelece o **escopo** do sistema, ou seja, o que ele vai resolver e o que não.

Os requisitos são voláteis. Eles costumam sofrer modificações durante o desenvolvimento. Requisitos novos podem aparecer e outros serem retirados do documento. Isto acontece porque as organizações estão inseridas num meio em que as coisas mudam numa velocidade significativa, principalmente no que se refere à informação. Durante o desenvolvimento de um sistema, várias coisas podem mudar: as tecnologias envolvidas, as expectativas dos usuários, as regras do negócio,...

Assim, o documento de requisitos serve como um consenso inicial. O objetivo principal da construção deste documento de referência consiste em permitir a compreensão do problema antes de começar a construção do sistema que irá resolvê-lo.

À medida em que novos requisitos forem sendo identificados, ou que os antigos requisitos mudem, o projetista, juntamente com o cliente (conjunto de perfis de usuário) devem verificar a relação custo X benefício da alteração do conjunto de requisitos originais.

É praticamente impossível, nos sistemas atuais, conseguir especificar todos os requisitos de forma correta desde o início. Além disso, somente quando o sistema é prototipado é que os usuários enxergarão o potencial da tecnologia utilizada e identificarão novos requisitos. (Ciclo da tecnologia: Prós e contras disto...?)

O documento de requisitos deve classificar os requisitos pelo grau de prioridade para o usuário. Isto permite que os pontos mais críticos do problema do usuário vão sendo atacados desde o início do desenvolvimento.

b) Análise

Conceito: A fase de **Análise** consiste em “quebrar” um sistema em seus componentes e em estudar como esses componentes devem interagir entre si com o objetivo de resolver o problema de hipótese.

Nesta fase, são estudados cuidadosamente os requisitos levantados na fase anterior, e são construídos os modelos que representarão o sistema a ser construído.

O foco de interesse é a construção de uma estratégia de solução, desconsiderando a forma em que essa estratégia será implementada. Em outras palavras, a modelagem

deve determinar **o que** o sistema deve fazer, e não ainda o como ele deverá fazê-lo.

A fase de Análise pode ser subdividida em duas subfaces: Análise do domínio (ou do negócio) e Análise da aplicação.

Conceito: A **Análise do domínio**, **Análise dos processos de negócio** ou **Análise do negócio** tem por objetivos identificar e modelar os objetos do mundo real que, de alguma forma, serão utilizados pelo sistema em desenvolvimento.

Exemplo: No contexto de um sistema acadêmico, aluno é um objeto do mundo real que deve ser identificado e modelado.

A identificação dos objetos do domínio ou do negócio deve ser realizada junto aos especialistas da organização, pois são eles que mais entendem do domínio, do negócio e, portanto, têm maior poder de contribuição.

Nesta sub-fase devem ser identificadas as **regras** e os **processos do negócio**.

Conceito: A **Análise da aplicação** consiste na identificação dos objetos que não fazem muito sentido para os especialistas do domínio, mas que são necessários para garantir a funcionalidade requerida da aplicação.

Estes aspectos estão relacionados com os aspectos computacionais de alto nível da aplicação e, nesta fase, são apenas definidos, sem detalhar a sua implementação.

Exemplo: Uma tela de inscrição em disciplinas é um objeto componente de uma aplicação acadêmica.

Motivação: A motivação para que a Análise do domínio identifique objetos do domínio e a fase de Análise da aplicação objetos da aplicação está no potencial de reuso. Ao se modelar o domínio de forma separada da aplicação, os componentes resultantes têm maior potencial de reuso no desenvolvimento de outras aplicações no domínio do problema.

Os analistas e/ou projetistas não devem se estagnar durante a fase de Análise, mas tampouco devem sair dela sem antes validar e verificar os modelos construídos. Esta validação deve ser feita tanto pelos técnicos (analista, projetistas, programadores) quanto pelo cliente (conjunto de perfis de usuário potencial).

Conceito: A **Validação** consiste em assegurar que as necessidades do usuário estejam sendo atendidas pelo sistema modelado. "*Será que o software em construção é o software correto?*" "*A especificação de requisitos e os modelos do sistema são corretos, consistentes completos e realistas?*"

Na Validação, podem surgir diferentes interpretações entre projetistas e usuários potenciais. Estas diferenças refletem ambigüidades dos requisitos. Estas ambigüidades devem ser eliminadas na etapa de Análise, mais precisamente na Validação.

No entanto, a Validação dos modelos não é uma tarefa simples, se considerarmos que eles não podem ser obrigados a entender as linguagens técnicas (ex. UML) utilizadas na modelagem. Assim, toda e qualquer forma de representação, seja ela formal ou informal, visual ou textual, pode e deve ser aproveitada com o intuito de servir como **língua franca**, desde que suficientemente precisa para não perder informação em relação à expressividade dos modelos.

Conceito: A **Verificação** se refere à análise comparativa entre os modelos construídos e os requisitos antes definidos para o sistema. "*Os modelos refletem os requisitos?*" Nesta fase também são analisadas a corretude de cada modelo em separado e a consistência entre os modelos.

Resumo: O foco principal na Análise está nos aspectos lógicos e independentes de implementação do sistema, os requisitos.

c) Projeto (*Design*)

Conceito: Na fase de Projeto determina-se **o como** o sistema funcionará para atender aos requisitos. A fase de Projeto considera os aspectos físicos e dependentes de implementação. Nesta fase, aos modelos construídos durante a Análise, são adicionadas as **restrições tecnológicas**. Nesta fase são consideradas a **arquitetura do sistema**, a **linguagem de programação** a ser usada, o **gerenciador de banco de dados** e o **padrão de interface-usuário**.

O Projeto produz uma descrição computacional do que o software deve fazer, o que deve ser coerente com a descrição feita na Análise.

O Projeto é subdividido em duas subfaces: O Projeto da arquitetura (dito Projeto de alto nível) e o Projeto detalhado (chamado de Projeto de baixo nível).

Conceito: Durante o desenvolvimento OO, o **Projeto da arquitetura** consiste em:

- distribuir as classes de objetos relacionadas entre si em diversos subsistemas;
- distribuir fisicamente os componentes de cada subsistema entre os recursos de hardware disponíveis.

Conceito: No **Projeto detalhado**, é feita:

g) a modelagem da colaboração entre as classes de objetos de cada módulo que respondem pela sua funcionalidade plena;

h) o Projeto da interface-usuário;

i) o Projeto do banco de dados;

j) a consideração das questões de concorrência e distribuição do sistema;

k) o Projeto dos algoritmos que vão implementar os módulos.

Os diagramas mais utilizados no Projeto detalhado são o Diagrama de classes, o diagrama de casos de uso, o Diagrama de interação o Diagrama de estados e o Diagrama de atividades.

Embora para efeitos didáticos o processo de Análise seja separado do Projeto, não existe, no desenvolvimento de sistemas reais OO, um separador claro entre estas fases; elas se misturam.

d) Implementação

Conceito: Na fase de Implementação o sistema é **codificado**, ou seja, tem seus módulos "traduzidos" para uma ou mais linguagens de programação.

No desenvolvimento OO, a implementação envolve:

- a criação do código fonte correspondente às classes de objetos;
- o (eventual) reuso de componentes de software e de bibliotecas de classes.

e) Testes

Conceito: O objetivo da fase de **Testes** consiste em verificar se o sistema realmente faz tudo o que foi especificado na fase de Levantamento de requisitos.

O principal produto da fase de Testes é o **Relatório de testes**, que contém a relação de erros identificados no software.

Após a realização dos testes, os diversos módulos do sistema são integrados, resultando finalmente (após novos Testes) no produto de software final.

f) Implantação

Conceito: A implantação consiste em:

- empacotar;
- distribuir;
- instalar o sistema no ambiente de destino.

Isto envolve:

- elaborar os manuais do sistema (de referência e do usuário);
- carregar os arquivos;
- importar os dados;
- treinar os diferentes perfis de usuário;
- (eventualmente) migrar sistemas e dados pré-existentes.

3. O componente humano (participantes do processo)

O desenvolvimento de software é uma atividade altamente colaborativa. Tecnologias complexas demandas especialistas em áreas específicas. Uma equipe de desenvolvimento pode envolver vários especialistas , como, por exemplo, profissionais de Informática e especialistas do domínio.

Uma equipe de desenvolvimento típica consiste de:

- gerente;
- analistas;
- projetistas ou “arquitetos de software”;
- programadores;
- “clientes” ou especialistas do domínio;
- grupos de avaliação da qualidade.

Esta divisão de tarefas é de cunho puramente didático. Na prática, várias destas funções podem ser assumidas por um único profissional, assim como uma única atividade pode ser atribuída a mais de uma pessoa.

3.1 Gerentes de projeto

Conceito: Como o próprio nome diz, este é o profissional responsável pela gerência das atividades necessárias à construção do sistema.

Dentre as responsabilidades atribuídas a este perfil profissional estão:

- a verificação da viabilidade do sistema;
- o escalonamento da equipe de desenvolvimento;
- elaboração do orçamento do projeto;
- a estimativa do tempo necessário;
- a definição do processo desenvolvimento;
- o cronograma de execução;
- a mão de obra especializada a ser alocada / contratada;
- os recursos de hardware e software necessários;
- o acompanhamento das atividades;
- a verificação de se os recursos alocados estão sendo usados na taxa esperada;
- a intervenção em caso de ocorrerem problemas no percurso.

3.2 Analistas

Conceito: O analista de sistemas é o profissional que deve compreender o problema do cliente, entender do "negócio" ou domínio de aplicação e levantar e especificar os requisitos do sistema. Ele representa "a ponte" entre os profissionais da Computação e os profissionais "do negócio".

O analista não necessita ser especialista na área de aplicação, mas necessita dominar razoavelmente o vocabulário da área para poder se comunicar com os especialistas do domínio e este não precisar parar a cada momento para explicar conceitos básicos.

O analista funciona como um "tradutor" entre a linguagem do domínio e a linguagem técnica (modelos).

Dada a importância, para uma organização, de ter uma pessoa que entende, simultaneamente, de desenvolvimento de sistemas e do negócio, é comum que, terminado o projeto, o analista receba um convite para fazer parte da organização para a qual estava prestando serviços.

O analista deve ter boa capacidade de expressão oral e escrita, pois ele age como facilitador da comunicação entre o "cliente" e a equipe técnica. (Desenvoltura pode até ser mais importante ainda...)

A ética é especialmente importante, na medida em que os analistas costumam ter acesso a dados sigilosos das organizações.

3.3 Arquitetos de software

Conceito: O arquiteto de software é um profissional especializado encontrado apenas em grandes organizações que desenvolvem software de alta complexidade. São atribuições deste profissional:

- elaborar a arquitetura do sistema como um todo;
- determinar os subsistemas componentes e as interfaces entre eles;
- tomar decisões técnicas detalhadas (Ex.: decisões que têm influência no desempenho do sistema);
- trabalhar em conjunto com o gerente de projeto para priorizar as atividades e planejar o trabalho.

3.4 Programadores

Conceito: Estes profissionais são responsáveis pela implementação do sistema. É comum haver mais de um programador nas equipes de desenvolvimento.

O programador deve conhecer bem:

- a metodologia utilizada na análise (modelos);
- uma ou mais linguagens de programação;
- banco de dados.

Enquanto o analista está envolvido em todas as etapas do desenvolvimento, o programador costuma participar unicamente das etapas finais.

Os programadores, por sua vez, podem se preocupar apenas (ou melhor, principalmente) com os aspectos técnicos do desenvolvimento.

Analistas devem ter conhecimento de Programação para poderem produzir especificações técnicas a serem repassados aos programadores.

É comum que os programadores que se destacam sejam “promovidos” a analistas.

Mito: Todo bom programador é bom analista, e vice-versa. Na verdade, são requeridas habilidades distintas...

3.5 “Clientes” ou especialistas do domínio

O especialista do domínio ou do negócio é o indivíduo ou o grupo de indivíduos que possui conhecimentos na área em que o sistema estará inserido.

Conceitos: Há dois tipos de cliente: o **cliente contratante** e o **cliente usuário**.

É junto ao cliente usuário que o analista deve interagir para levantar os requisitos do sistema.

Por outro lado, o cliente contratante é responsável por encomendar o sistema e aprovar os custos de desenvolvimento.

É importante ressaltar que a única forma de ter um usuário satisfeito com o sistema é torná-lo um legítimo participante do processo de desenvolvimento.

No caso dos sistemas desenvolvidos para venda posterior, quem faz o papel de especialista do domínio é o pessoal de Marketing.

3.6 Avaliadores de qualidade

A qualidade de um sistema envolve aspectos tais como desempenho e confiabilidade.

O grupo que tenha por função avaliar a qualidade do sistema será encarregado de checar a adequação do sistema aos requisitos previamente definidos.

4. Modelos de ciclo de vida

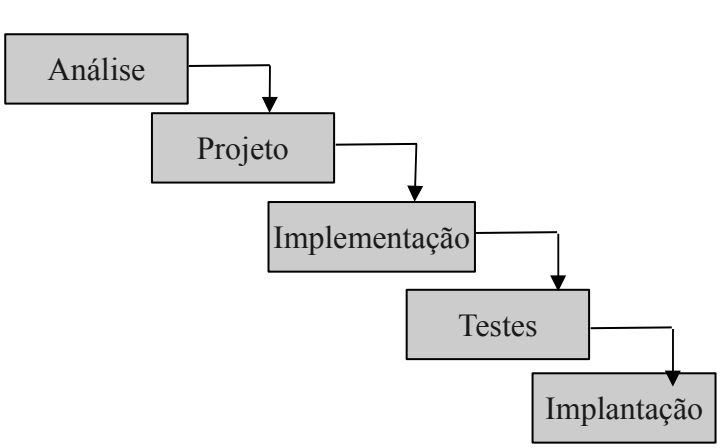
Conceito: As fases descritas nas seções anteriores são encadeadas na construção do sistema. A esse encadeamento chama-se **Modelo de Ciclo de Vida**.

Há diversos modelos de ciclo de vida. A diferença entre eles está na forma como eles encadeam as diversas fases no processo.

Abordaremos dois modelos: O modelo em cascata e o modelo iterativo e incremental.

3.1 Modelo de ciclo de vida em cascata

Conceito: Este modelo é também chamado de **modelo clássico** ou **linear**, por ter como tendncia a execução linear das diversas fases. Pode até haver retro-alimentação, mas é rara neste modelo.



O modelo em cascata têm uma série de problemas, todos decorrentes da sua característica principal, a seqüencialidade:

- Projetos reais raramente seguem o fluxo linear proposto pelo modelo. Algumas atividades de desenvolvimento podem ser realizadas em paralelo;
- A abordagem tem como pressuposto a possibilidade de especificação completa e detalhada dos requisitos no início do desenvolvimento. A discussão sobre a volatilidade de requisitos, juntamente com a do ciclo de evolução da tecnologia, mostra que esta hipótese não é realista;
- Uma versão passível de uso por usuários potenciais só estará disponível ao final do processo, quando todo o esforço terá sido realizado. Se o sistema for suficientemente complexo, pode passar um tempo significativo entre a especificação de requisitos e a entrega da primeira versão passível de uso, e isto pode levar a discrepâncias entre os requisitos originais e requisitos correntes

quando da entrega da versão.

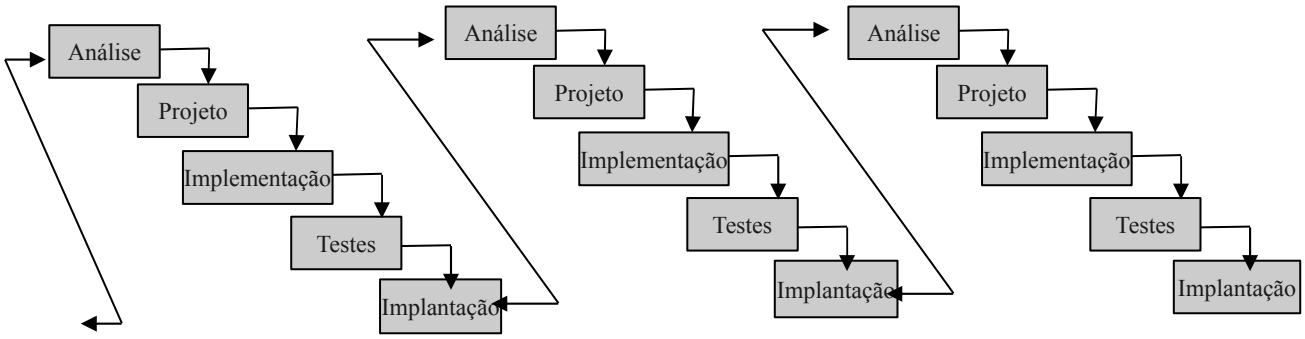
Deviso a estes problemas, o ciclo de vida mais utilizado atualmente é o que se aproveita das fases do ciclo de vida clássico, mas de uma forma iterativa.

3.2 Modelo de ciclo de vida iterativo e incremental

Conceito: Esta abordagem divide o processo de desenvolvimento de um produto de software em **ciclos**. Cada ciclo é composto da seqüência de fases de desenvolvimento da análise tradicional, envolve apenas parte dos requisitos e tem como produto uma parte do sistema passível de uso.

Num ciclo seguinte, outro subconjunto de requisitos são tratados, resultando num subsistema mais extenso e refinado em relação àquele emergente do ciclo anterior.

No sistema iterativo incremental, cada ciclo é uma cascata em miniatura.



Os subconjuntos de requisitos vão sendo determinados por um lado, pela inter-relação entre requisitos e, por outro, pelos graus de prioridade e de risco relativos associados aos requisitos no projeto. Projetos com maior prioridade e/ou com alto risco são normalmente escolhidos para compor os primeiros ciclos.