Engenharia de Software

Capítulo 1

Engenharia de Software — Conceitos Básicos

1. Introdução

Nos anos 40, quando se iniciou a evolução dos sistemas computadorizados, grande parte dos esforços, e conseqüentes custos, era concentrada no desenvolvimento do hardware, em razão, principalmente das limitações e dificuldades encontradas na época. À medida que a tecnologia de hardware foi sendo dominada, as preocupações se voltaram, no início dos anos 50, para o desenvolvimento dos sistemas operacionais, onde surgiram então as primeiras realizações destes sistemas, assim como das chamadas linguagens de programação de alto nível, como FORTRAN e COBOL, e dos respectivos compiladores. A tendência da época foi de poupar cada vez mais o usuário de um computador de conhecer profundamente as questões relacionadas ao funcionamento interno da máquina, permitindo que este pudesse concentrar seus esforços na resolução dos problemas computacionais em lugar de preocupar-se com os problemas relacionados ao funcionamento do hardware.

Já no início dos anos 60, com o surgimento dos sistemas operacionais com características de multiprogramação, a eficiência e utilidade dos sistemas computacionais tiveram um considerável crescimento, para o que contribuíram também, de forma bastante significativa, as constantes quedas de preço do hardware.

Uma conseqüência deste crescimento foi a necessidade, cada vez maior, de desenvolver grandes sistemas de software em substituição aos pequenos programas aplicativos utilizados até então. Desta necessidade, surgiu um problema nada trivial devido à falta de experiência e à não adequação dos métodos de desenvolvimento existentes para pequenos programas, o que foi caracterizado, ainda na década de 60 como a "crise do software", mas que, por outro lado, permitiu o nascimento do termo "Engenharia de Software".

Atualmente, apesar da constante queda dos preços dos equipamentos, o custo de desenvolvimento de software não obedece a esta mesma tendência. Pelo contrário, corresponde a uma percentagem cada vez maior no custo global de um sistema informatizado. A principal razão para isto é que a tecnologia de desenvolvimento de software implica, ainda, em grande carga de trabalho, os projetos de grandes sistemas de software envolvendo em regra geral um grande número de pessoas num prazo relativamente longo de desenvolvimento. O desenvolvimento destes sistemas é realizado, na maior parte das vezes, de forma "ad-hoc", conduzindo a freqüentes desrespeitos de cronogramas e acréscimos de custos de desenvolvimento.

O objetivo deste curso é apresentar propostas de soluções às questões relacionadas ao desenvolvimento de software, através da transferência dos principais conceitos relativos à Engenharia de Software, particularmente no que diz respeito ao uso de técnicas, metodologias e ferramentas de desenvolvimento de software.

2. PRINCIPAIS ASPECTOS DO SOFTWARE

2.1. Software: definição e características

Pode-se definir o **software**, numa forma clássica, como sendo: "um **conjunto de instruções** que, quando executadas, produzem a função e o desempenho desejados, **estruturas de dados** que permitam que as informações relativas ao problema a resolver sejam manipuladas adequadamente e a **documentação** necessária para um melhor entendimento da sua operação e uso".

Entretanto, no contexto da Engenharia de Software, o software deve ser visto como um **produto** a ser "vendido". É importante dar esta ênfase, diferenciando os "programas" que são concebidos num contexto mais restrito, onde o usuário ou "cliente" é o próprio autor. No caso destes programas, a documentação associada é pequena ou (na maior parte das vezes) inexistente e a preocupação com a existência de erros de execução não é um fator maior, considerando que o principal usuário é o próprio autor do programa, este não terá dificuldades, em princípio, na detecção e correção de um eventual "bug". Além do aspecto da correção, outras boas características não são também objeto de preocupação como a portabilidade, a flexibilidade e a possibilidade de reutilização.

Um **produto de software** (ou software, como vamos chamar ao longo do curso), por outro lado, é sistematicamente destinado ao uso por pessoas outras que os seus programadores. Os eventuais usuários podem, ainda, ter formações e experiências diferentes, o que significa que uma grande preocupação no que diz respeito ao desenvolvimento do produto deve ser a sua interface, reforçada com uma documentação rica em informações para que todos os recursos oferecidos possam ser explorados de forma eficiente. Ainda, os produtos de software devem passar normalmente por uma exaustiva bateria de testes, dado que os usuários não estarão interessados (e nem terão capacidade) de detectar e corrigir os eventuais erros de execução.

Resumindo, um programa desenvolvido para resolver um dado problema e um produto de software destinado à resolução do mesmo problema são duas coisas totalmente diferentes. É óbvio que o esforço e o conseqüente custo associado ao desenvolvimento de um produto serão muito superiores.

Dentro desta ótica, é importante, para melhor caracterizar o significado de software, é importante levantar algumas particularidades do software, quando comparadas a outros produtos, considerando que o software é um elemento lógico e não físico como a maioria dos produtos:

- ***o software é concebido e desenvolvido como resultado de um trabalho de engenharia*** e não manufaturado no sentido clássico;
- ***o software não se desgasta***, ou seja, ao contrário da maioria dos produtos, o software não se caracteriza por um aumento na possibilidade de falhas à medida que o tempo passa (como acontece com a maioria dos produtos manufaturados);
- ***a maioria dos produtos de software é concebida inteiramente sob medida***, sem a utilização de componentes pré-existentes.

Em função destas características diferenciais, o processo de desenvolvimento de software pode desembocar um conjunto de problemas, os quais terão influência direta na **qualidade do produto**.

Tudo isto porque, desde os primórdios da computação, o desenvolvimento dos programas (ou, a programação) era visto como uma forma de arte, sem utilização de metodologias formais e sem qualquer preocupação com a documentação, entre outros fatores importantes. A experiência do programador era adquirida através de tentativa e erro. A verdade é que esta tendência ainda se verifica. Com o crescimento dos custos de software (em relação aos de hardware) no custo total de um sistema computacional, o

processo de desenvolvimento de software tornou-se um item de fundamental importância na produção de tais sistemas.

A nível industrial, algumas questões que caracterizaram as preocupações com o processo de desenvolvimento de software foram:

- por que o software demora tanto para ser concluído?
- por que os custos de produção têm sido tão elevados?
- por que não é possível detectar todos os erros antes que o software seja entregue ao cliente?
- por que é tão difícil medir o progresso durante o processo de desenvolvimento de software?

Estas são algumas das questões que a **Engenharia de Software** pode ajudar a resolver. Enquanto não respondemos definitivamente a estas questões, podemos levantar alguns dos problemas que as originam:

- raramente, durante o desenvolvimento de um software, é dedicado tempo para coletar dados sobre o processo de desenvolvimento em sí; devido à pouca quantidade deste tipo de informação, as tentativas em estimar a duração/custo de produção de um software têm conduzido a resultados bastante insatisfatórios; além disso, a falta destas informações impede uma avaliação eficiente das técnicas e metodologias empregadas no desenvolvimento;
- a insatisfação do cliente com o sistema "concluído" ocorre freqüentemente, devido, principalmente, ao fato de que os projetos de desenvolvimento são baseados em informações vagas sobre as necessidades e desejos do cliente (problema de comunicação entre cliente e fornecedor);
- a qualidade do software é quase sempre suspeita, problema resultante da pouca atenção que foi dada, historicamente, às técnicas de teste de software (até porque o conceito de **qualidade de software** é algo relativamente recente);
- o software existente é normalmente muito difícil de manter em operação, o que significa que o custo do software acaba sendo incrementado significativamente devido às atividades relacionadas à manutenção; isto é um reflexo da pouca importância dada à manutenibilidade no momento da concepção dos sistemas.

2.2. Software: Mitos e Realidade

É possível apontar, como causas principais dos problemas levantados na seção anterior, três principais pontos:

- a falta de experiência dos profissionais na condução de projetos de software;
- a falta de treinamento no que diz respeito ao uso de técnicas e métodos formais para o desenvolvimento de software;
- a “cultura de programação” que ainda é difundida e facilmente aceita por estudantes e profissionais de Ciências da Computação;
- a incrível "resistência" às mudanças (particularmente, no que diz respeito ao uso de novas técnicas de desenvolvimento de software) que os profissionais normalmente apresentam.

Entretanto, é importante ressaltar e discutir os chamados "mitos e realidades" do software, o que, de certo modo, explicam alguns dos problemas de desenvolvimento de software apresentados.

2.2.1. Mitos de Gerenciamento

Mito 1. *"Se a equipe dispõe de um manual repleto de padrões e procedimentos de desenvolvimento de software, então a equipe está apta a encaminhar bem o desenvolvimento."*

Realidade 1. Isto verdadeiramente não é o suficiente... é preciso que a equipe aplique efetivamente os conhecimentos apresentados no manual... é necessário que o que conste no dado manual reflita a moderna prática de desenvolvimento de software e que este seja exaustivo com relação a todos os problemas de desenvolvimento que poderão aparecer no percurso...

Mito 2. *"A equipe tem ferramentas de desenvolvimento de software de última geração, uma vez que eles dispõem de computadores de última geração."*

Realidade 2. Ter à sua disposição o último modelo de computador (seja ele um mainframe, estação de trabalho ou PC) pode ser bastante confortável para o desenvolvedor do software, mas não oferece nenhuma garantia quanto à qualidade do software desenvolvido. Mais importante do que ter um hardware de última geração é ter ferramentas para a automatização do desenvolvimento de software (as ferramentas CASE)...

Mito 3. *"Se o desenvolvimento do software estiver atrasado, basta aumentar a equipe para honrar o prazo de desenvolvimento."*

Realidade 3. Isto também dificilmente vai ocorrer na realidade... alguém disse um dia que "... acrescentar pessoas em um projeto atrasado vai torná-lo ainda mais atrasado...". De fato, a introdução de novos profissionais numa equipe em fase de condução de um projeto vai requerer uma etapa de treinamento dos novos elementos da equipe; para isto, serão utilizados elementos que estão envolvidos diretamente no desenvolvimento, o que vai, conseqüentemente, implicar em maiores atrasos no cronograma.

2.2.2. Mitos do Cliente

Mito 4. *"Uma descrição breve e geral dos requisitos do software é o suficiente para iniciar o seu projeto... maiores detalhes podem ser definidos posteriormente."*

Realidade 4. Este é um dos problemas que podem conduzir um projeto ao fracasso, o cliente deve procurar definir o mais precisamente possível todos os requisitos importantes para o software: funções, desempenho, interfaces, restrições de projeto e critérios de validação são alguns dos pontos determinantes do sucesso de um projeto.

Mito 5. *"Os requisitos de projeto mudam continuamente durante o seu desenvolvimento, mas isto não representa um problema, uma vez que o software é flexível e poderá suportar facilmente as alterações."*

Realidade 5. É verdade que o software é flexível (pelo menos mais flexível do que a maioria dos produtos manufaturados). Entretanto, não existe software, por mais flexível que suporte alterações de requisitos significativas com adicional zero em relação ao custo de desenvolvimento. O fator de multiplicação nos custos de desenvolvimento do software devido a alterações nos requisitos cresce em função do estágio de evolução do projeto, como mostra a figura 1.1.

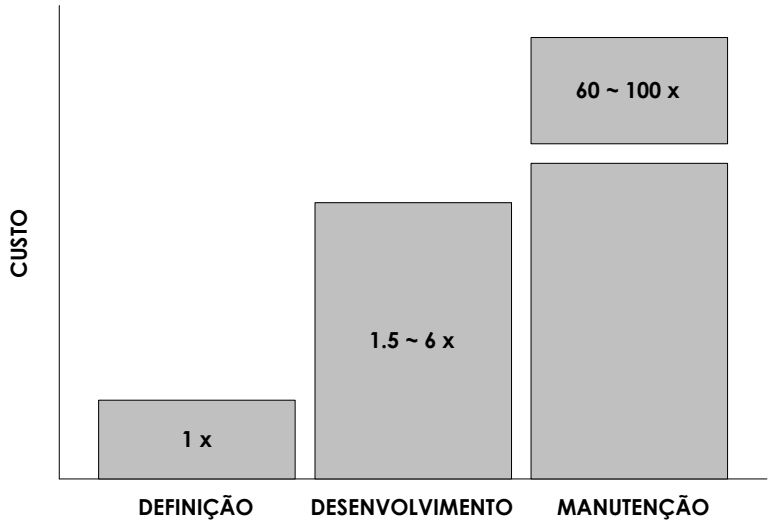

Figura 1.1 - Influência das alterações de requisitos no custo de um sistema.

2.2.2. Mitos do Profissional

Mito 6. *"Após a edição do programa e a sua colocação em funcionamento, o trabalho está terminado."*

Realidade 6. O que ocorre na realidade é completamente diferente disto. Segundo dados obtidos a partir de experiências anteriores, 50 a 70% do esforço de desenvolvimento de um software é despendido após a sua entrega ao cliente (manutenção).

Mito 7. *"Enquanto o programa não entrar em funcionamento, é impossível avaliar a sua qualidade."*

Realidade 7. Na realidade, a preocupação com a garantia do software deve fazer parte de todas as etapas do desenvolvimento, sendo que, ao fim de cada uma destas etapas, os documentos de projeto devem ser revisados observando critérios de qualidade.

Mito 8. *"O produto a ser entregue no final do projeto é o programa funcionando."*

Realidade 8. O programa em funcionamento é uma das componentes do software...além do software, um bom projeto deve ser caracterizado pela produção de um conjunto importante de documentos, os quais podem ser identificados com auxílio da figura 1.2.

3. PARADIGMAS DA ENGENHARIA DE SOFTWARE

3.1 Definição de Engenharia de Software

Os problemas apontados nas seções anteriores não serão, é claro, resolvidos da noite para o dia, mas reconhecer a existência dos problemas e definí-los da forma mais precisa e eficaz são, sem dúvida, um primeiro passo para a sua solução.

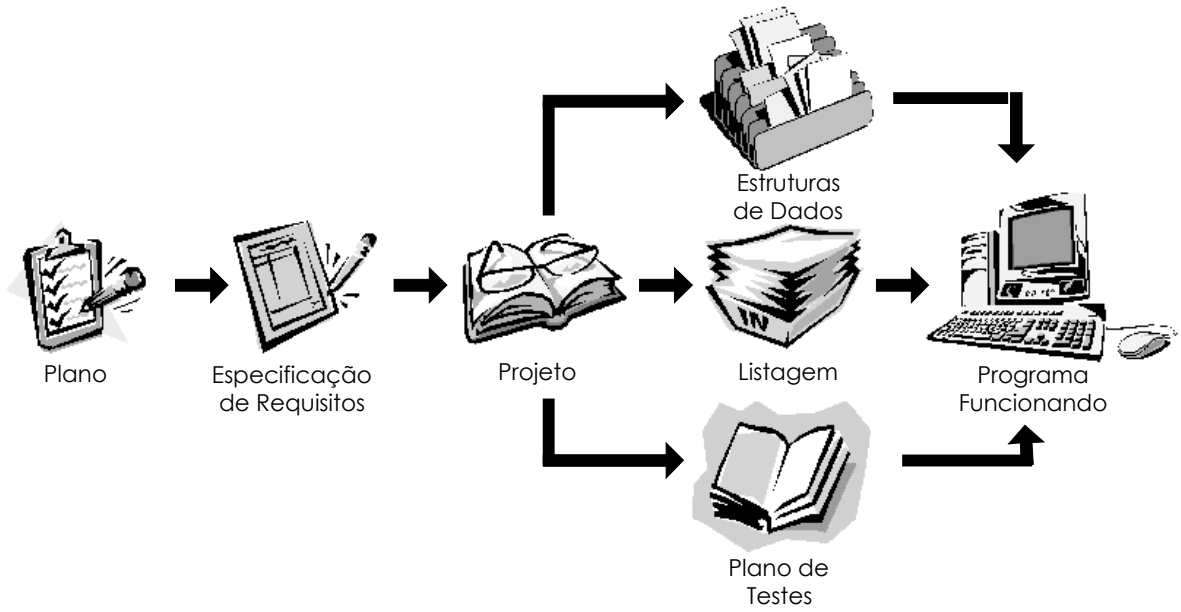

Figura 1.2 - Componentes do software.

Em primeiro lugar, é preciso estar ciente também de que não existe uma abordagem mágica que seja a melhor para a solução destes problemas, mas uma combinação de métodos que sejam abrangentes a todas as etapas do desenvolvimento de um software.

Além disto, é importante e desejável que estes métodos sejam suportados por um conjunto de ferramentas que permita automatizar o desenrolar destas etapas, juntamente com uma definição clara de critérios de qualidade e produtividade de software. São estes aspectos que caracterizam de maneira mais influente a disciplina de **Engenharia de Software**.

Na literatura, pode-se encontrar diversas definições da Engenharia de Software:

"O estabelecimento e uso de sólidos princípios de engenharia para que se possa obter economicamente um software que seja confiável e que funcione eficientemente em máquinas reais" [NAU 69].

"A aplicação prática do conhecimento científico para o projeto e a construção de programas computacionais e a documentação necessária à sua operação e manutenção." [Boehm, 76]

"Abordagem sistemática para o desenvolvimento, a operação e a manutenção de software" [Afnor, 83]

"Conjunto de métodos, técnicas e ferramentas necessárias à produção de software de qualidade para todas as etapas do ciclo de vida do produto." [Krakowiak, 85]

Num ponto de vista mais formal, a Engenharia de Software pode ser definida como sendo a aplicação da ciência e da matemática através das quais os equipamentos computacionais são colocados à disposição do homem por meio de programas, procedimentos e documentação associada. De modo mais objetivo, pode-se dizer que a Engenharia de Software busca prover a tecnologia necessária para produzir software de alta qualidade a um baixo custo. Os dois fatores motivadores são essencialmente a qualidade e o custo. A qualidade de um produto de software é um parâmetro cuja quantificação não é trivial, apesar dos esforços desenvolvidos nesta direção. Por outro lado, o fator custo pode

ser facilmente quantificado desde que os procedimentos de contabilidade tenham sido corretamente efetuados.

3.2. Modelos de Desenvolvimento de software

O resultado de um esforço de desenvolvimento deve resultar normalmente num **produto**. O **processo de desenvolvimento** corresponde ao conjunto de atividades e um ordenamento destas de modo a que o produto desejado seja obtido.

Um **modelo de desenvolvimento** corresponde a uma representação abstrata do processo de desenvolvimento que vai, em geral, definir como as etapas relativas ao desenvolvimento do software serão conduzidas e interrelacionadas para atingir o objetivo do desenvolvimento que é a obtenção de um produto de software de alta qualidade a um custo relativamente baixo.

As seções que seguem vão descrever alguns dos modelos conhecidos e utilizados em desenvolvimento de software.

3.2.1. O Modelo Queda d'Água

Este é o modelo mais simples de desenvolvimento de software, estabelecendo uma ordenação linear no que diz respeito à realização das diferentes etapas. Como mostra a figura 1.3, o ponto de partida do modelo é uma etapa de **Engenharia de Sistemas**, onde o objetivo é ter uma visão global do sistema como um todo (incluindo hardware, software, equipamentos e as pessoas envolvidas) como forma de definir precisamente o papel do software neste contexto. Em seguida, a etapa de **Análise de Requisitos** vai permitir uma clara definição dos requisitos de software, sendo que o resultado será utilizado como referência para as etapas posteriores de **Projeto**, **Codificação**, **Teste** e **Manutenção**.

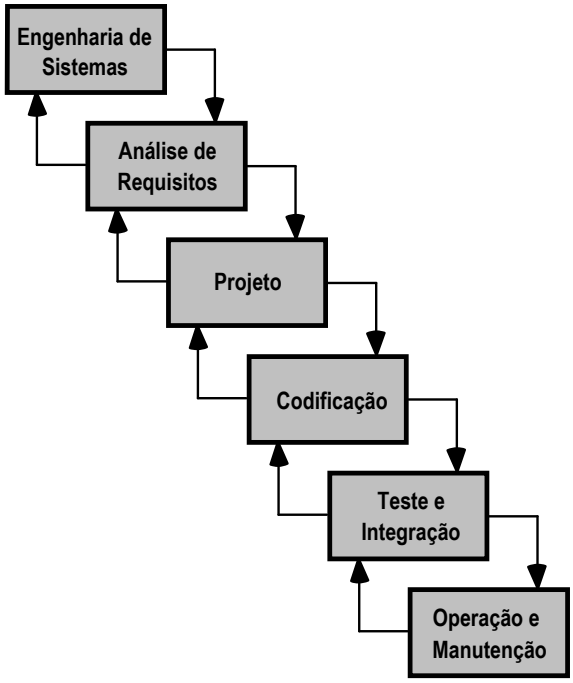

Figura 1.3 - Ilustração do modelo Queda d'Água.

O modelo Queda d'Água apresenta características interessantes, particularmente em razão da definição de um ordenamento linear das etapas de desenvolvimento. Primeiramente, como forma de identificar precisamente o fim de uma etapa de o início da seguinte, um mecanismo de certificação (ou revisão) é implementado ao final de cada

etapa; isto é feito normalmente através da aplicação de algum método de validação ou verificação, cujo objetivo será garantir de que a saída de uma dada etapa é coerente com a sua entrada (a qual já é a saída da etapa precedente). Isto significa que ao final de cada etapa realizada, deve existir um resultado (ou saída) a qual possa ser submetida à atividade de certificação.

Estas saídas, obtidas ao final de cada etapa, são vistas como produtos intermediários e apresentam-se, normalmente, na forma de documentos (documento de especificação de requisitos, documento de projeto do sistema, etc...).

Outra característica importante deste modelo é que as saídas de uma etapa são as entradas da seguinte, o que significa que uma vez definidas, elas não devem, em hipótese alguma ser modificadas.

Duas diretivas importantes que norteiam o desenvolvimento segundo o modelo Queda d'Água, são:

- todas as etapas definidas no modelo devem ser realizadas, isto porque, em projetos de grande complexidade, a realização formal destas vai determinar o sucesso ou não do desenvolvimento; a realização informal e implícita de algumas destas etapas poderia ser feita apenas no caso de projetos de pequeno porte;
- a ordenação das etapas na forma como foi apresentada deve ser rigorosamente respeitada; apesar de que esta diretiva poderia ser questionada, a ordenação proposta pelo modelo, por ser a forma mais simples de desenvolver, tem sido também a mais adotada a nível de projetos de software.

É importante lembrar, também que os resultados de um processo de desenvolvimento de software não devem ser exclusivamente o programa executável e a documentação associada. Existe uma quantidade importante de resultados (ou produtos intermediários) os quais são importantes para o sucesso do desenvolvimento. Baseados nas etapas apresentadas na figura 1.3, é possível listar os resultados mínimos esperados de um processo de desenvolvimento baseado neste modelo: Documento de Especificação de Requisitos, Projeto do Sistema, Plano de Teste e Relatório de Testes, Código Final, Manuais de Utilização, Relatórios de Revisões.

Apesar de ser um modelo bastante popular, pode-se apontar algumas limitações apresentadas por este modelo:

- o modelo assume que os requisitos são inalterados ao longo do desenvolvimento; isto em boa parte dos casos não é verdadeira, uma vez que nem todos os requisitos são completamente definidos na etapa de análise;
- muitas vezes, a definição dos requisitos pode conduzir à definição do hardware sobre o qual o sistema vai funcionar; dado que muitos projetos podem levar diversos anos para serem concluídos, estabelecer os requisitos em termos de hardware é um tanto temeroso, dadas as freqüentes evoluções no hardware;
- o modelo impõe que todos os requisitos sejam completamente especificados antes do prosseguimento das etapas seguintes; em alguns projetos, é às vezes mais interessante poder especificar completamente somente parte do sistema, prosseguir com o desenvolvimento do sistema, e só então encaminhar os requisitos de outras partes; isto não é previsto a nível do modelo;
- as primeiras versões operacionais do software são obtidas nas etapas mais tardias do processo, o que na maioria das vezes inquieta o cliente, uma vez que ele quer ter acesso rápido ao seu produto.

De todo modo, pode vir a ser mais interessante a utilização deste modelo para o desenvolvimento de um dado sistema do que realizar um desenvolvimento de maneira totalmente anárquica e informal.

3.2.2. Prototipação

O objetivo da Prototipação é um modelo de processo de desenvolvimento que busca contornar algumas das limitações existentes no modelo Queda d'Água. A idéia por trás deste modelo é eliminar a política de "congelamento" dos requisitos antes do projeto do sistema ou da codificação.

Isto é feito através da obtenção de um protótipo, com base no conhecimento dos requisitos iniciais para o sistema. O desenvolvimento deste protótipo é feito obedecendo à realização das diferentes etapas já mencionadas, a saber, a análise de requisitos, o projeto, a codificação e os testes, sendo que não necessariamente estas etapas sejam realizadas de modo muito explícito ou formal.

Este protótipo pode ser oferecido ao cliente em diferentes formas, a saber:

- protótipo em papel ou modelo executável em PC retratando a interface homem-máquina capacitando o cliente a compreender a forma de interação com o software;
- um protótipo de trabalho que implemente um subconjunto dos requisitos indicados;
- um programa existente (pacote) que permita representar todas ou parte das funções desejadas para o software a construir.

Colocado à disposição do cliente, o protótipo vai ajudá-lo a melhor compreender o que será o sistema desenvolvido. Além disso, através da manipulação deste protótipo, é possível validar ou reformular os requisitos para as etapas seguintes do sistema.

Este modelo, ilustrado na figura 1.4, apresenta algumas características interessantes, tais como:

- é um modelo de desenvolvimento interessante para alguns sistemas de grande porte os quais representem um certo grau de dificuldade para exprimir rigorosamente os requisitos;
- através da construção de um protótipo do sistema, é possível demonstrar a realizabilidade do mesmo;
- é possível obter uma versão, mesmo simplificada do que será o sistema, com um pequeno investimento inicial.

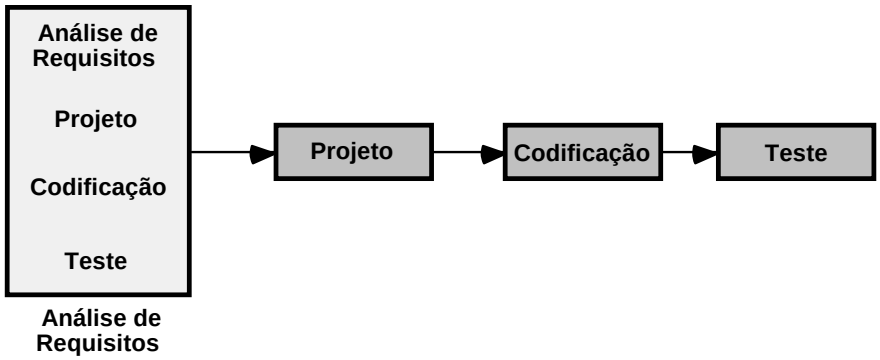

Figura 1.4 - Esquema de evolução da prototipação.

Os protótipos não são sistemas completos e deixam, normalmente, a desejar em alguns aspectos. Um destes aspectos é normalmente a interface com o usuário. Os esforços de desenvolvimento são concentrados principalmente nos algoritmos que implementem as principais funções associadas aos requisitos apresentados, a interface

sendo, a este nível parte supérflua do desenvolvimento, o que permite caracterizar esta etapa por um custo relativamente baixo.

Por outro lado, a experiência adquirida no desenvolvimento do protótipo vai ser de extrema utilidade nas etapas posteriores do desenvolvimento do sistema real, permitindo reduzir certamente o seu custo, resultando também num sistema melhor concebido.

3.2.3. Desenvolvimento Iterativo

Este modelo também foi concebido com base numa das limitações do modelo Queda d'Água e combinar as vantagens deste modelo com as do modelo Prototipação. A idéia principal deste modelo, ilustrada na figura 1.5, é a de que um sistema deve ser desenvolvido de forma incremental, sendo que cada incremento vai adicionando ao sistema novas capacidades funcionais, até a obtenção do sistema final, sendo que, a cada passo realizado, modificações podem ser introduzidas.

Uma vantagem desta abordagem é a facilidade em testar o sistema, uma vez que a realização de testes em cada nível de desenvolvimento é, sem dúvida, mais fácil do que testar o sistema final. Além disso, como na Prototipação, a obtenção de um sistema, mesmo incompleto num dado nível, pode oferecer ao cliente interessantes informações que sirvam de subsídio para a melhor definição de futuros requisitos do sistema.

No primeiro passo deste modelo uma implementação inicial do sistema é obtida, na forma de um subconjunto da solução do problema global. Este primeiro nível de sistema deve contemplar os principais aspectos que sejam facilmente identificáveis no que diz respeito ao problema a ser resolvido.

Um aspecto importante deste modelo é a criação de uma **lista de controle de projeto**, a qual deve apresentar todos os passos a serem realizados para a obtenção do sistema final. Ela vai servir também para se medir, num dado nível, o quão distante se está da última iteração. Cada iteração do modelo de Desenvolvimento Iterativo consiste em retirar um passo da lista de controle de projeto através da realização de três etapas: o projeto, a implementação e a análise. O processo avança, sendo que a cada etapa de avaliação, um passo é retirado da lista, até que a lista esteja completamente vazia. A lista de controle de projeto gerencia todo o desenvolvimento, definindo quais tarefas devem ser realizadas a cada iteração, sendo que as tarefas na lista podem representar, inclusive, redefinições de componentes já implementados, em razão de erros ou problemas detectados numa eventual etapa de análise.

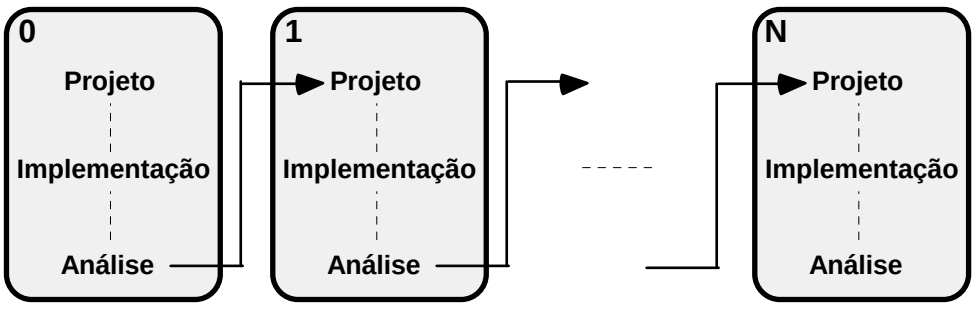

Figura 1.5 - O modelo Desenvolvimento Iterativo.

3.2.4. O Modelo Espiral

Este modelo, proposto em 1988, sugere uma organização das atividades em espiral, a qual é composta de diversos ciclos. Como mostrado na figura 1.6, a dimensão vertical representa o custo acumulado na realização das diversas etapas; a dimensão angular representa o avanço do desenvolvimento ao longo das etapas.

Cada ciclo na espiral inicia com a identificação dos objetivos e as diferentes alternativas para se atingir aqueles objetivos assim como as restrições impostas. O próximo passo no ciclo é a avaliação das diferentes alternativas com base nos objetivos fixados, o que vai permitir também definir incertezas e riscos de cada alternativa. No passo seguinte, o desenvolvimento de estratégias permitindo resolver ou eliminar as incertezas levantadas anteriormente, o que pode envolver atividades de prototipação, simulação, avaliação de desempenho, etc... Finalmente, o software é desenvolvido e o planejamento dos próximos passos é realizado.

A continuidade do processo de desenvolvimento é definida como função dos riscos remanescentes, como por exemplo, a decisão se os riscos relacionados ao desempenho ou à interface são mais importantes do que aqueles relacionados ao desenvolvimento do programa. Com base nas decisões tomadas, o próximo passo pode ser o desenvolvimento de um novo protótipo que elimine os riscos considerados.

Por outro lado, caso os riscos de desenvolvimento de programa sejam considerados os mais importantes e se o protótipo obtido no passo corrente já resolve boa parte dos riscos ligados a desempenho e interface, então o próximo passo pode ser simplesmente a evolução segundo o modelo Queda d'Água.

Como se pode ver, o elemento que conduz este processo é essencialmente a consideração sobre os riscos, o que permite, de certo modo, a adequação a qualquer política de desenvolvimento (baseada em especificação, baseada em simulação, baseada em protótipo, etc...).

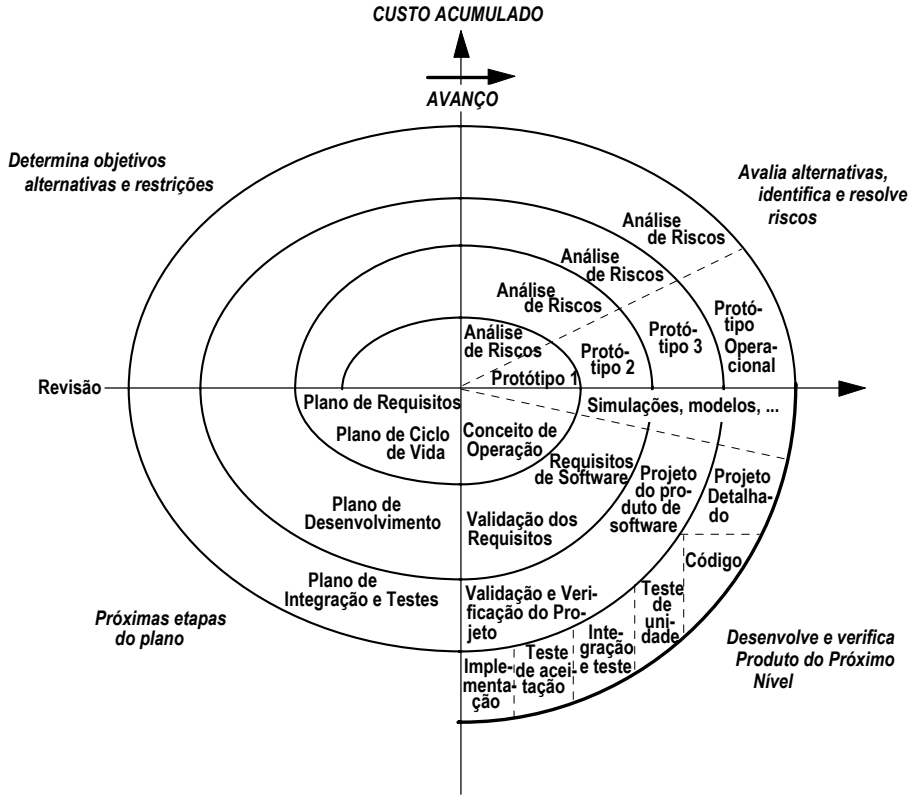

Figura 1.7 - O modelo espiral.

Uma característica importante deste modelo é o fato de que cada ciclo é encerrado por uma atividade de revisão, onde todos os produtos do ciclo são avaliados, incluindo o plano para o próximo passo (ou ciclo). Numa aplicação típica do modelo, pode-se imaginar a realização de um ciclo zero, onde se avalie a realizabilidade do projeto, o resultado devendo ser a conclusão de que será possível implementar ou não o projeto de desenvolvimento. As alternativas consideradas neste caso são de muito alto nível, como por

exemplo, se a organização deve desenvolver o sistema ela própria ou se deve contratar o desenvolvimento junto a uma empresa especializada.

O modelo se adequa principalmente a sistemas que representem um alto risco de investimento para o cliente.

4. VISÃO GERAL DA ENGENHARIA DE SOFTWARE

Analisando os modelos apresentados na seção precedente, é possível observar que, apesar de apresentar denominações às vezes diferentes e de estarem associadas de modo relativamente distinto, as etapas apresentadas são caracterizadas por atividades similares.

De um modo geral, pode-se organizar o processo de desenvolvimento de um software a partir de três grandes fases: a **fase de definição**, a **fase de desenvolvimento** e a **fase de manutenção** as quais serão discutidas nas seções abaixo.

4.1. Fase de Definição

A fase de definição está associada à determinação do **que** vai ser feito. Nesta fase, o profissional encarregado do desenvolvimento do software deve identificar as informações que deverão ser manipuladas, as funções a serem processadas, qual o nível de desempenho desejado, que interfaces devem ser oferecidas, as restrições do projeto e os critérios de validação. Isto terá de ser feito não importando o modelo de desenvolvimento adotado para o software e independente da técnica utilizada pra fazê-lo.

Esta fase é caracterizada pela realização de três etapas específicas:

- a **Análise (ou Definição) do Sistema**, a qual vai permitir determinar o papel de cada elemento (hardware, software, equipamentos, pessoas) no sistema, cujo objetivo é determinar, como resultado principal, as funções atribuídas ao software;
- o **Planejamento do Projeto de Software**, no qual, a partir da definição do escopo do software, será feita uma análise de riscos e a definição dos recursos, custos e a programação do processo de desenvolvimento;
- a **Análise de Requisitos**, que vai permitir determinar o conjunto das funções a serem realizadas assim como as principais estruturas de informação a serem processadas.

4.2. Fase de Desenvolvimento

Nesta fase, será determinado **como** realizar as funções do software. Aspectos como a arquitetura do software, as estruturas de dados, os procedimentos a serem implementados, a forma como o projeto será transformado em linguagem de programação, a geração de código e os procedimentos de teste devem ser encaminhados nesta fase.

Normalmente, esta fase é também organizada em três principais etapas:

- o **Projeto de Software**, o qual traduz, num conjunto de representações gráficas, tabulares ou textuais, os requisitos do software definidos na fase anterior; estas representações (diversas técnicas de representação podem ser adotadas num mesmo projeto) permitirão definir, com um alto grau de abstração, aspectos do software como a arquitetura, os dados, lógicas de comportamento (algoritmos) e características da interface;
- a **Codificação**, onde as representações realizadas na etapa de projeto serão mapeadas numa ou em várias linguagens de programação, a qual será caracterizada por um conjunto de instruções executáveis no computador; nesta etapa, considera-se também a geração de código de implementação, aquele obtido a partir do uso de ferramentas (compiladores, linkers, etc...) e que será executado pelo hardware do sistema;

- os **Testes de Software**, onde o programa obtido será submetido a uma bateria de testes para verificar (e corrigir) defeitos relativos às funções, lógica de execução, interfaces, etc...

4.3. Fase de Manutenção

A fase de manutenção, que se inicia a partir da entrega do software, é caracterizada pela realização de alterações de naturezas as mais diversas, seja para corrigir erros residuais da fase anterior, para incluir novas funções exigidas pelo cliente, ou para adaptar o software a novas configurações de hardware.

Sendo assim, pode-se caracterizar esta fase pelas seguintes atividades:

- a **Correção** ou **Manutenção Corretiva**, a qual consiste da atividade de correção de erros observados durante a operação do sistema;
- a **Adaptação** ou **Manutenção Adaptativa**, a qual realiza alterações no software para que ele possa ser executado sobre um novo ambiente (CPU, arquitetura, novos dispositivos de hardware, novo sistema operacional, etc...);
- o **Melhoramento Funcional** ou **Manutenção Perfectiva**, onde são realizadas alterações para melhorar alguns aspectos do software, como por exemplo, o seu desempenho, a sua interface, a introdução de novas funções, etc...

A manutenção do software envolve, normalmente, etapas de análise do sistema existente (entendimento do código e dos documentos associados), teste das mudanças, teste das partes já existentes, o que a torna uma etapa complexa e de alto custo.

Além disso, considerando a atual situação industrial, foi criado, mais recentemente, o conceito de **Engenharia Reversa**, onde, através do uso das técnicas e ferramentas da Engenharia de Software, o software existente sofre uma "reforma geral", cujo objetivo é aumentar a sua qualidade e atualizá-lo com respeito às novas tecnologias de interface e de hardware.