UFES - Universidade Federal do Espírito Santo

Engenharia de Software

Notas de Aula

PARTE I

Ricardo de Almeida Falbo

E-mail: falbo@inf.ufes.br

Curso: **Engenharia da Computação**

(Atualizadas por Ricardo de Almeida Falbo e Monalessa Perini Barcellos - 2011)

ÍNDICE

Capítulo 1 – Introdução

O desenvolvimento de software é uma atividade de crescente importância na sociedade contemporânea. A utilização de computadores nas mais diversas áreas do conhecimento humano tem gerado uma crescente demanda por soluções computadorizadas.

Para os iniciantes na Ciência de Computação, desenvolver software é, muitas vezes, confundido com programação. Essa confusão inicial pode ser atribuída, parcialmente, pela forma como as pessoas são introduzidas nesta área de conhecimento, começando por desenvolver habilidades de raciocínio lógico, através de programação e estruturas de dados. Aliás, nada há de errado nessa estratégia. Começamos resolvendo pequenos problemas que gradativamente vão aumentando de complexidade, requerendo maiores conhecimentos e habilidades.

Entretanto, chega-se a um ponto em que, dado o tamanho ou a complexidade do problema que se pretende resolver, essa abordagem individual, centrada na programação não é mais indicada. De fato, ela só é aplicável para resolver pequenos problemas, tais como calcular médias, ordenar conjuntos de dados etc, envolvendo basicamente o projeto de um único algoritmo. Contudo, é insuficiente para problemas grandes e complexos, tais como aqueles tratados na automação bancária, na informatização de portos ou na gestão empresarial. Em tais situações, uma abordagem de engenharia é necessária.

Observando outras áreas, tal como a Engenharia Civil, podemos verificar que situações análogas ocorrem. Por exemplo, para se construir uma casinha de cachorro, não é necessário elaborar um projeto de engenharia civil, com plantas baixa, hidráulica e elétrica, ou mesmo cálculos estruturais. Um bom pedreiro é capaz de resolver o problema a contento. Talvez não seja dada a melhor solução, mas o produto resultante pode atender aos requisitos pré-estabelecidos. Essa abordagem, contudo, não é viável para a construção de um edifício. Nesse caso, é necessário realizar um estudo aprofundado, incluindo análises de solo, cálculos estruturais etc, seguido de um planejamento da execução da obra e desenvolvimento de modelos (maquetes e plantas de diversas naturezas), até a realização da obra, que deve ocorrer por etapas, tais como fundação, alvenaria, acabamento etc. Ao longo da realização do trabalho, deve-se realizar um acompanhamento para verificar prazos, custos e a qualidade do que se está construindo.

Visando melhorar a qualidade dos produtos de software e aumentar a produtividade no processo de desenvolvimento, surgiu a *Engenharia de Software*. A Engenharia de Software trata de aspectos relacionados ao estabelecimento de processos, métodos, técnicas, ferramentas e ambientes de suporte ao desenvolvimento de software.

Assim como em outras áreas, em uma abordagem de engenharia de software, inicialmente o problema a ser tratado deve ser analisado e decomposto em partes menores, em uma abordagem "dividir para conquistar". Para cada uma dessas partes, uma solução deve ser elaborada. Solucionados os subproblemas isoladamente, é necessário integrar as soluções. Para tal, uma arquitetura deve ser estabelecida. Para apoiar a resolução de problemas, procedimentos (métodos, técnicas, roteiros etc) devem ser utilizados, bem como ferramentas para parcialmente automatizar o trabalho.

Neste cenário, muitas vezes não é possível conduzir o desenvolvimento de software de maneira individual. Pessoas têm de trabalhar em equipes, o esforço tem de ser planejado, coordenado e acompanhado, bem como a qualidade do que se está produzindo tem de ser sistematicamente avaliada.

1.1 – Qualidade de Software

Uma vez que um dos objetivos da Engenharia de Software é melhorar a qualidade dos produtos de software desenvolvidos, uma questão deve ser analisada: O que é qualidade de software?

Se perguntarmos a um usuário, provavelmente, ele dirá que um produto de software é de boa qualidade se ele satisfizer suas necessidades, sendo fácil de usar, eficiente e confiável. Essa é uma perspectiva externa de observação pelo uso do produto. Por outro lado, para um desenvolvedor, um produto de boa qualidade tem de ser fácil de manter, sendo o produto de software observado por uma perspectiva interna. Já para um cliente, o produto de software deve agregar valor a seu negócio (qualidade em uso).

Em última instância, podemos perceber que a qualidade é um conceito com múltiplas facetas (perspectivas de usuário, desenvolvedor e cliente) e que envolve diferentes características (por exemplo, usabilidade, confiabilidade, eficiência, manutenibilidade, portabilidade, segurança, produtividade) que devem ser alcançadas em níveis diferentes, dependendo do propósito do software. Por exemplo, um sistema de tráfego aéreo tem de ser muito mais eficiente e confiável do que um editor de textos. Por outro lado, um software educacional a ser usado por crianças deve primar muito mais pela usabilidade do que um sistema de venda de passagens aéreas a ser operado por agentes de turismo especializados.

O que há de comum nas várias perspectivas discutidas acima é que todas elas estão focadas no produto de software. Ou seja, estamos falando de qualidade do produto. Entretanto, como garantir que o produto final de software apresenta essas características? Apenas avaliar se o produto final as apresenta é uma abordagem indesejável para o pessoal de desenvolvimento de software, tendo em vista que a constatação *a posteriori* de que o software não apresenta a qualidade desejada pode implicar na necessidade de refazer grande parte do trabalho. É necessário, pois, que a qualidade seja incorporada ao produto ao longo de seu processo de desenvolvimento. De fato, a qualidade dos produtos de software depende fortemente da qualidade dos processos usados para desenvolvê-los e mantê-los.

Seguindo uma tendência de outros setores, a qualidade do processo de software tem sido apontada como fundamental para a obtenção da qualidade do produto. Abordagens de qualidade de processo, tal como a série de padrões ISO 9000, sugerem que melhorando a qualidade do processo de software, é possível melhorar a qualidade dos produtos resultantes. A premissa por detrás dessa afirmativa é a de que processos bem estabelecidos, que incorporam mecanismos sistemáticos para acompanhar o desenvolvimento e avaliar a qualidade, no geral, conduzem a produtos de qualidade. Por exemplo, quando se diz que um fabricante de eletrodomésticos é uma empresa certificada ISO 9001 (uma das normas da série ISO 9000), não se está garantindo que todos os eletrodomésticos por ele produzidos são produtos de qualidade. Mas sim que ele tem um bom processo produtivo, o que deve levar a produtos de qualidade.

Um processo de software, em uma abordagem de Engenharia de Software, envolve diversas atividades que podem ser classificadas quanto ao seu propósito em:

- Atividades de Desenvolvimento (ou Técnicas ou de Construção): são as atividades diretamente relacionadas ao processo de desenvolvimento do software, ou seja, que contribuem diretamente para o desenvolvimento do produto de software a ser entregue ao cliente. São exemplos de atividades de desenvolvimento: especificação e análise de requisitos, projeto e implementação.
- Atividades de Gerência de Projeto: são aquelas relacionadas ao planejamento e acompanhamento gerencial do projeto, tais como realização de estimativas, elaboração de cronogramas, análise dos riscos do projeto etc.
- Atividades de Garantia da Qualidade: são aquelas relacionadas com a garantia da qualidade do produto em desenvolvimento e do processo de software utilizado, tais como revisões e inspeções de produtos (intermediários ou finais) do desenvolvimento.

As atividades de desenvolvimento formam a espinha dorsal do desenvolvimento e, de maneira geral, são realizadas segundo uma ordem estabelecida no planejamento. As atividades de gerência e de controle da qualidade são, muitas vezes, ditas atividades de apoio, pois não estão ligadas diretamente à construção do produto final: o software a ser entregue para o cliente, incluindo toda a documentação necessária. Essas atividades, normalmente, são realizadas ao longo de todo o ciclo de vida, sempre que necessário ou em pontos pré-estabelecidos durante o planejamento, ditos marcos ou pontos de controle. A Figura 1.1 mostra a relação entre esses tipos de atividades.

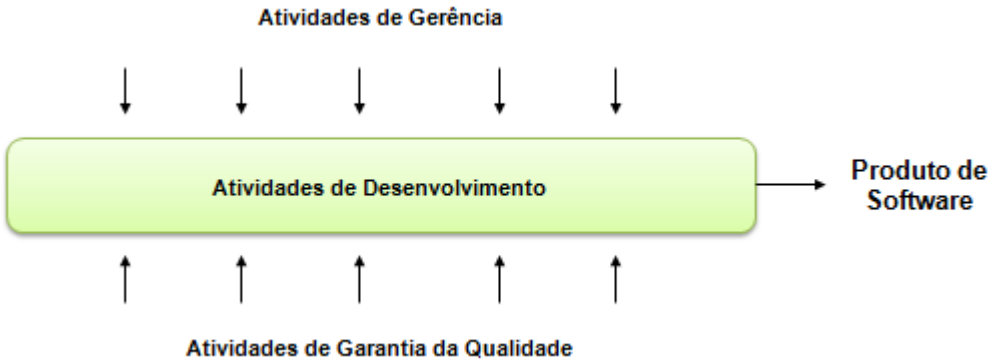

Figura 1.1 – Atividades do Processo de Software.

1.2 – A Organização deste Texto

Nesta disciplina, procuramos oferecer uma visão geral da Engenharia de Software, discutindo as principais atividades do processo e como realizá-las. Este material está dividido em duas partes. Na Parte I são abordados o processo de software em si e as atividades de gerência de projetos e de garantia da qualidade. Esses temas são tratados em três capítulos, a saber:

- Capítulo 2 – *Processo de Software* – enfoca os processos de software, os elementos que compõem um processo, a definição de processos para projetos, modelos de processo, normas e modelos de qualidade de processo de software e a automatização do processo de software.
- Capítulo 3 –*Gerência de Projetos* – são abordadas as principais atividades da gerência de projetos, a saber: definição do escopo do projeto, estimativas, análise de riscos, elaboração de cronograma, elaboração do plano de projeto e acompanhamento de projetos.
- Capítulo 4 – *Gerência da Qualidade* – trata de algumas atividades relacionadas à garantia da qualidade, incluindo a medição e métricas associadas, revisões e inspeções e a gerência de configuração de software.

Na Parte II deste material são abordadas as atividades de desenvolvimento. Para isso, quatro capítulos são apresentados:

- Capítulo 5 – *Levantamento e Análise de Requisitos* – são discutidos o que é um requisito de software e tipos de requisitos. Em seguida, são abordadas a especificação e a análise de requisitos. É oferecida uma visão geral da abordagem de Análise Orientada a Objetos. As técnicas de modelagem de casos de uso, modelagem estrutural e modelagem de estados são apresentadas.
- Capítulo 6 – *Projeto de Sistema* – aborda os conceitos básicos de projeto de sistemas, tratando da arquitetura do sistema a ser desenvolvido e do projeto de seus componentes. Também são discutidos o projeto de dados e o projeto de interface com o usuário.
- Capítulo 7 – *Implementação e Testes* – são enfocadas as atividades de implementação e testes, sendo esta última tratada em diferentes níveis, a saber: teste de unidade, teste de integração, teste de validação e teste de sistema.
- Capítulo 8 – *Entrega e Manutenção* – discute as questões relacionadas à entrega do sistema para o cliente, tais como o treinamento e a documentação de entrega, e a atividade de manutenção do sistema.

Referências

- S.L. Pfleeger, *Engenharia de Software: Teoria e Prática*, São Paulo: Prentice Hall, 2ª edição, 2004. Capítulo 1.

Capítulo 2 – Processo de Software

Para se construir um produto ou sistema, seja de que natureza for, é necessário seguir uma série de passos previsíveis, isto é, um guia, que ajude a chegar a um resultado de qualidade, dentro do tempo previsto [1a]. No caso do desenvolvimento de software, esse "guia" é o processo de software.

2.1 - O que é um Processo de Software

Um processo de software pode ser visto como o conjunto de atividades, métodos, práticas e transformações que guiam pessoas na produção de software. Um processo eficaz deve, claramente, considerar as relações entre as atividades, os artefatos produzidos no desenvolvimento, as ferramentas e os procedimentos necessários e a habilidade, o treinamento e a motivação do pessoal envolvido.

Elementos que compõem um processo de software

Os processos de software são, geralmente, decompostos em diversos processos, tais como processo de desenvolvimento, processo de garantia da qualidade, processo de gerência de projetos etc. Esses processos, por sua vez, são compostos de atividades, que também podem ser decompostas. Para cada atividade de um processo é importante saber quais as suas subatividades, as atividades que devem precedê-las (pré-atividades), os artefatos de entrada (insumos) e de saída (produtos) da atividade, os recursos necessários (humanos, hardware, software etc) e os procedimentos (métodos, técnicas, modelos de documento etc) a serem utilizados na sua realização. A Figura 2.1 mostra os elementos que compõem um processo de forma esquemática.

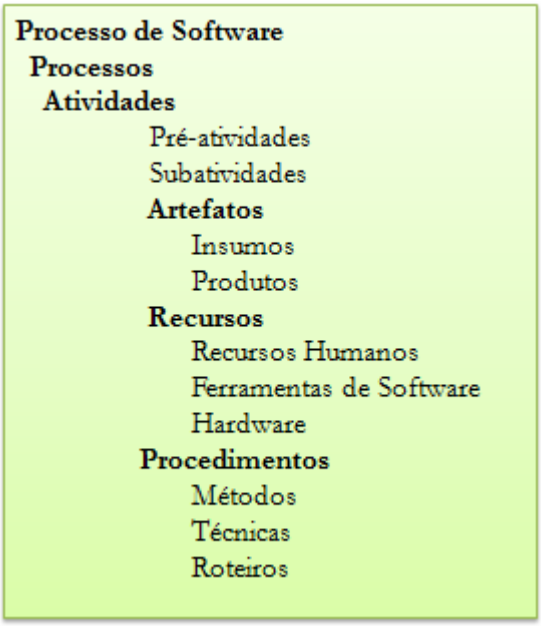

Figura 2.1 – Elementos que compõem um Processo de Software.

Definição de Processos

O objetivo de se definir um processo de software é favorecer a produção de sistemas de alta qualidade, atingindo as necessidades dos usuários finais, dentro de um cronograma e um orçamento previsíveis.

Um processo de software não pode ser definido de forma universal. Para ser eficaz e conduzir à construção de produtos de boa qualidade, um processo deve ser adequado às especificidades do projeto em questão. Deste modo, processos devem ser definidos caso a caso, considerando-se as características da aplicação (domínio do problema, tamanho, complexidade etc), a tecnologia a ser adotada na sua construção (paradigma de desenvolvimento, linguagem de programação, mecanismo de persistência etc), a organização onde o produto será desenvolvido e a equipe de desenvolvimento.

Há vários aspectos a serem considerados na definição de um processo de software. No centro da arquitetura de um processo de desenvolvimento estão as atividades-chave desse processo: análise e especificação de requisitos, projeto, implementação e testes, que são a base sobre a qual o processo de desenvolvimento deve ser construído. Entretanto, a definição de um processo envolve a escolha de um modelo de ciclo de vida (ou modelo de processo), o detalhamento (decomposição) de suas macro-atividades, a escolha de métodos, técnicas e roteiros (procedimentos) para a sua realização e a definição de recursos e artefatos necessários e produzidos.

A escolha de um modelo de ciclo de vida (ou modelo de processo) é o ponto de partida para a definição de um processo de desenvolvimento de software. Um modelo de ciclo de vida, geralmente, organiza as macro-atividades básicas do processo, estabelecendo precedência e dependência entre as mesmas. Para a definição completa do processo, cada atividade descrita no modelo de ciclo de vida deve ser decomposta e para suas subatividades, devem ser associados métodos, técnicas, ferramentas e critérios de qualidade, entre outros, formando uma base sólida para o desenvolvimento. Adicionalmente, outras atividades, tipicamente de cunho gerencial, devem ser definidas, entre elas atividade de gerência de projetos e de controle e garantia da qualidade.

Os seguintes fatores influenciam a definição de um processo e, por conseguinte, a escolha do modelo de processo a ser usado como base: tipo de software (p.ex., sistema de informação, sistema de tempo real etc), paradigma de desenvolvimento (estruturado, orientado a objetos etc), domínio da aplicação, tamanho e complexidade do sistema, estabilidade dos requisitos, características da equipe etc.

2.2 - Modelos de Ciclo de Vida ou Modelos de Processo

Um modelo de ciclo de vida ou modelo de processo pode ser visto como uma representação abstrata de um esqueleto de processo, incluindo tipicamente algumas atividades principais, a ordem de precedência entre elas e, opcionalmente, artefatos requeridos e produzidos. De maneira geral, um modelo de processo descreve uma filosofia de organização de atividades, estruturando as atividades do processo em fases e definindo como essas fases estão relacionadas. Entretanto, ele não descreve um curso de ações preciso, recursos, procedimentos e restrições. Ou seja, ele é um importante ponto de partida para definir como o

projeto deve ser conduzido, mas a sua adoção não é o suficiente para guiar e controlar um projeto de software na prática.

Ainda que os processos tenham de ser definidos caso a caso, de maneira geral, o ciclo de vida de um software envolve, pelo menos, as seguintes fases:

- *Planejamento*: O objetivo do planejamento de projeto é fornecer uma estrutura que possibilite ao gerente fazer estimativas razoáveis de recursos, custos e prazos. Uma vez estabelecido o escopo de software, com os requisitos esboçados, uma proposta de desenvolvimento deve ser elaborada, isto é, um plano de projeto deve ser elaborado configurando o processo a ser utilizado no desenvolvimento de software. À medida que o projeto progride, o planejamento deve ser detalhado e atualizado regularmente. Pelo menos ao final de cada uma das fases do desenvolvimento (análise e especificação de requisitos, projeto, implementação e testes), o planejamento como um todo deve ser revisto e o planejamento da etapa seguinte deve ser detalhado. O planejamento e o acompanhamento do progresso fazem parte do processo de gerência de projeto.
- *Análise e Especificação de Requisitos*: Nesta fase, o processo de levantamento de requisitos é intensificado. O escopo deve ser refinado e os requisitos mais bem definidos. Para entender a natureza do software a ser construído, o engenheiro de software tem de compreender o domínio do problema, bem como a funcionalidade e o comportamento esperados. Uma vez capturados os requisitos do sistema a ser desenvolvido, estes devem ser modelados, avaliados e documentados. Uma parte vital desta fase é a construção de um modelo descrevendo *o que* o software tem de fazer (e não *como* fazê-lo).
- *Projeto*: Esta fase é responsável por incorporar requisitos tecnológicos aos requisitos essenciais do sistema, modelados na fase anterior e, portanto, requer que a plataforma de implementação seja conhecida. Basicamente, envolve duas grandes etapas: projeto da arquitetura do sistema e projeto detalhado. O objetivo da primeira etapa é definir a arquitetura geral do software, tendo por base o modelo construído na fase de análise de requisitos. Essa arquitetura deve descrever a estrutura de nível mais alto da aplicação e identificar seus principais componentes. O propósito do projeto detalhado é detalhar o projeto do software para cada componente identificado na etapa anterior. Os componentes de software devem ser sucessivamente refinados em níveis maiores de detalhamento, até que possam ser codificados e testados.
- *Implementação*: O projeto deve ser traduzido para uma forma passível de execução pela máquina. A fase de implementação realiza esta tarefa, isto é, cada unidade de software do projeto detalhado é implementada.
- *Testes*: inclui diversos níveis de testes, a saber, teste de unidade, teste de integração e teste de sistema. Inicialmente, cada unidade de software implementada deve ser testada e os resultados documentados. A seguir, os diversos componentes devem ser integrados sucessivamente até se obter o sistema. Finalmente, o sistema como um todo deve ser testado.
- *Entrega e Implantação*: uma vez testado, o software deve ser colocado em produção. Para tal, contudo, é necessário treinar os usuários, configurar o ambiente de produção e, muitas vezes, converter bases de dados. O propósito desta fase é

estabelecer que o software satisfaz os requisitos dos usuários. Isto é feito instalando o software e conduzindo testes de aceitação. Quando o software tiver demonstrado prover as capacidades requeridas, ele pode ser aceito e a operação iniciada.

- *Operação*: nesta fase, o software é utilizado pelos usuários no ambiente de produção.
- *Manutenção*: Indubitavelmente, o software sofrerá mudanças após ter sido entregue para o usuário. Alterações ocorrerão porque erros foram encontrados, porque o software precisa ser adaptado para acomodar mudanças em seu ambiente externo, ou porque o cliente necessita de funcionalidade adicional ou aumento de desempenho. Muitas vezes, dependendo do tipo e porte da manutenção necessária, essa fase pode requerer a definição de um novo processo, onde cada uma das fases precedentes é re-aplicada no contexto de um software existente ao invés de um novo.

Os modelos de processo, de maneira geral, contemplam as fases Análise e Especificação de Requisitos, Projeto, Implementação, Testes e Entrega e Implantação. A escolha de um modelo de processo é fortemente dependente das características do projeto. Assim, é importante conhecer alguns modelos de ciclo de vida e em que situações são aplicáveis. Os principais modelos de ciclo de vida podem ser agrupados em três categorias principais: modelos seqüenciais, modelos incrementais e modelos evolutivos.

2.2.1 – Modelos Sequenciais

Como o nome indica, os modelos sequenciais organizam o processo em uma sequência linear de fases. O principal modelo desta categoria é o modelo em cascata, a partir do qual diversos outros modelos foram propostos, inclusive a maioria dos modelos incrementais e evolutivos.

O Modelo em Cascata

Também chamado de "modelo de ciclo de vida clássico", o modelo em cascata organiza as atividades do processo de desenvolvimento de forma seqüencial, como mostra a Figura 2.2. Cada fase envolve a elaboração de um ou mais documentos, que devem ser aprovados antes de se iniciar a fase seguinte. Assim, uma fase só deve ser iniciada após a conclusão daquela que a precede. Uma vez que, na prática, essas fases se sobrepõem de alguma forma, geralmente, permite-se um retorno à fase anterior para a correção de erros encontrados. A entrega do sistema completo ocorre em um único marco, ao final da fase de Entrega e Implantação.

O uso de revisões ao fim de cada fase permite o envolvimento do usuário. Além disso, cada fase serve como uma base aprovada e documentada para o passo seguinte, facilitando bastante a gerência de configuração.

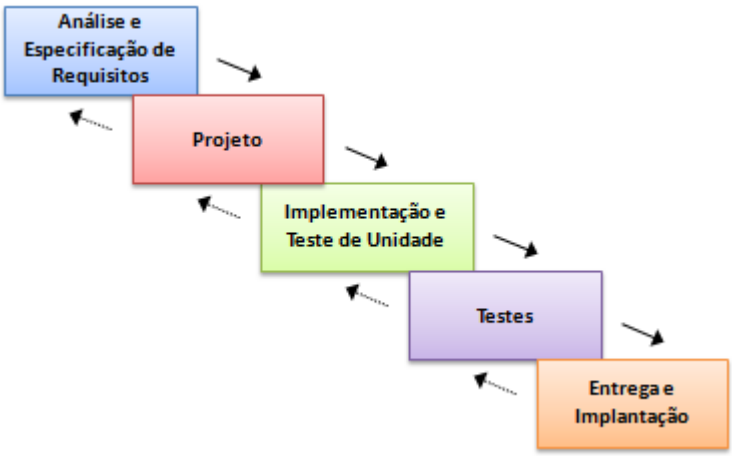

Figura 2.2 - O Modelo em Cascata.

O modelo em cascata é o modelo de ciclo de vida mais antigo e mais amplamente usado. Entretanto, críticas têm levado ao questionamento de sua eficiência. Dentre os problemas algumas vezes encontrados na sua aplicação, destacam-se [1]:

- Projetos reais muitas vezes não seguem o fluxo seqüencial que o modelo propõe.
- Os requisitos devem ser estabelecidos de maneira completa, correta e clara logo no início de um projeto. A aplicação deve, portanto, ser entendida pelo desenvolvedor desde o início do projeto. Entretanto, frequentemente, é difícil para o usuário colocar todos os requisitos explicitamente. O modelo em cascata requer isto e tem dificuldade de acomodar a incerteza natural que existe no início de muitos projetos.
- O usuário precisa ser paciente. Uma versão operacional do software não estará disponível até o final do projeto.
- A introdução de certos membros da equipe, tais como projetistas e programadores, é frequentemente adiada desnecessariamente. A natureza linear do ciclo de vida clássico leva a "estados de bloqueio" nos quais alguns membros da equipe do projeto precisam esperar que outros membros da equipe completem tarefas dependentes.

Cada um desses problemas é real. Entretanto, o modelo de ciclo de vida clássico tem um lugar definitivo e importante na engenharia de software. Muitos outros modelos mais complexos são, na realidade, variações do modelo cascata, incorporando laços de feedback [3]. Embora tenha fraquezas, ele é significativamente melhor do que uma abordagem casual para o desenvolvimento de software. De fato, para problemas bastante pequenos e bem-definidos, onde os desenvolvedores conhecem bem o domínio do problema e os requisitos podem ser claramente estabelecidos, esse modelo é indicado, uma vez que é fácil de ser gerenciado.

O Modelo em V

O modelo em V é uma variação do modelo em cascata que procura enfatizar a estreita relação entre as atividades de teste (teste de unidade, teste de integração, teste de sistema e teste de aceitação) e as demais fases do processo [3], como mostra a Figura 2.3.

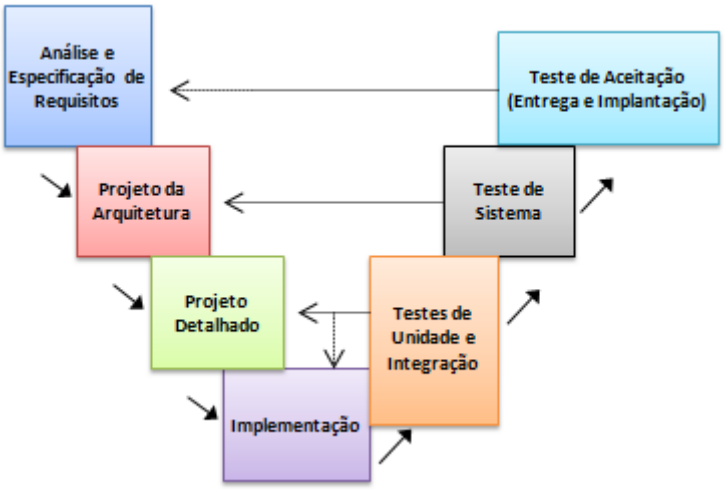

Figura 2.3 - O Modelo em V.

O modelo em V sugere que os testes de unidade são utilizados basicamente para verificar a implementação e o projeto detalhado. Uma vez que os testes de integração estão focados na integração das unidades que compõem o software, eles também são usados para avaliar o projeto detalhado. Assim, testes de unidade e integração devem garantir que todos os aspectos do projeto do sistema foram implementados corretamente no código. Quando os testes de integração atingem o nível do sistema como um todo (teste de sistema), o projeto da arquitetura passa a ser o foco. Neste momento, busca-se verificar se o sistema atende aos requisitos definidos na especificação. Finalmente, os testes de aceitação, conduzidos tipicamente pelos usuários e clientes, valida os requisitos, confirmando que os requisitos corretos foram implementados no sistema (teste de validação).

A conexão entre os lados direito e esquerdo do modelo em V implica que, caso sejam encontrados problemas em uma atividade de teste, a correspondente fase do lado esquerdo e suas fases subseqüentes podem ter de ser executadas novamente para corrigir ou melhorar esses problemas.

Os modelos sequenciais pressupõem que o sistema é entregue completo, após a realização de todas as atividades do desenvolvimento. Entretanto, nos dias de hoje, os clientes não estão mais dispostos a esperar o tempo necessário para tal, sobretudo, quando se trata de grandes sistemas [3]. Dependendo do porte do sistema, podem se passar anos até que o sistema fique pronto, sendo inviável esperar. Assim, outros modelos foram propostos visando a, dentre outros, reduzir o tempo de desenvolvimento. A entrega por partes, possibilitando ao usuário dispor de algumas funcionalidades do sistema enquanto outras estão sendo ainda desenvolvidas, é um dos principais mecanismos utilizados por esses modelos, como veremos a seguir.

2.2.2 – Modelos Incrementais

Há muitas situações em que os requisitos são razoavelmente bem definidos, mas o tamanho do sistema a ser desenvolvido impossibilita a adoção de um modelo sequencial, sobretudo pela necessidade de disponibilizar rapidamente uma versão para o usuário. Nesses casos, um modelo incremental é indicado [1a].

No desenvolvimento incremental, o sistema é dividido em subsistemas ou módulos, tomando por base a funcionalidade. Os incrementos (ou versões) são definidos, começando com um pequeno subsistema funcional que, a cada ciclo, é acrescido de novas funcionalidades. Além de acrescentar novas funcionalidades, nos novos ciclos, as funcionalidades providas anteriormente podem ser modificadas para melhor satisfazer às necessidades dos clientes / usuários. Vale destacar que a definição das versões (e a correspondente segmentação e atribuição dos requisitos a essas versões) é realizada antes do desenvolvimento da primeira versão.

O Modelo Incremental

O modelo incremental pode ser visto como uma filosofia básica que comporta diversas variações. O princípio fundamental é que, a cada ciclo ou iteração, uma versão operacional do sistema será produzida e entregue para uso ou avaliação detalhada do cliente. Para tal, requisitos têm de ser minimamente levantados e há de se constatar que o sistema é modular, de modo que se possa planejar o desenvolvimento em incrementos. O primeiro incremento tipicamente contém funcionalidades centrais, tratando dos requisitos básicos. Outras características são tratadas em ciclos subsequentes.

Dependendo do tempo estabelecido para a liberação dos incrementos, algumas atividades podem ser feitas para o sistema como um todo ou não. A Figura 2.4 mostra as principais possibilidades.

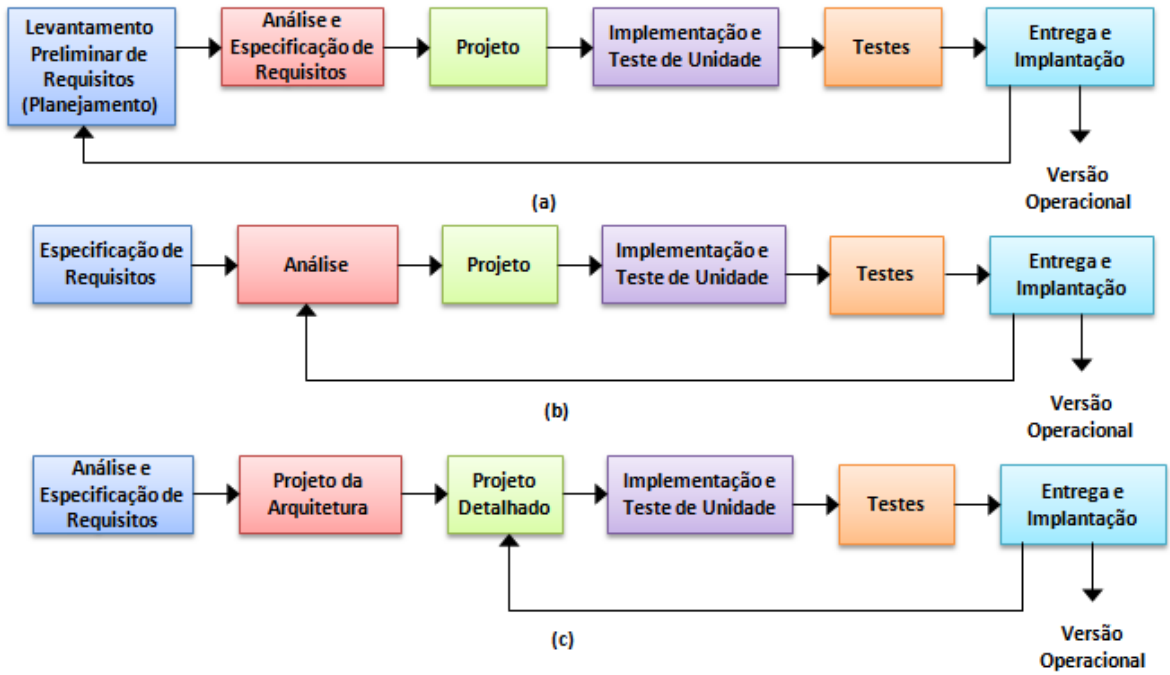

Figura 2.4 – Variações do Modelo Incremental.

Na Figura 2.4 (a), inicialmente, durante a fase de planejamento, um levantamento preliminar dos requisitos é realizado. Com esse esboço dos requisitos, planejam-se os incrementos e se desenvolve a primeira versão do sistema. Concluído o primeiro ciclo, todas as atividades são novamente realizadas para o segundo ciclo, inclusive o planejamento,

quando a atribuição de requisitos aos incrementos pode ser revista. Este procedimento é repetido sucessivamente, até que se chegue ao produto final.

Outras duas variações bastante utilizadas do modelo incremental são apresentadas na Figura 2.4 (b) e (c). Na Figura 2.4 (b), os requisitos são especificados para o sistema como um todo e as iterações ocorrem a partir da fase de análise. Na Figura 2.4 (c), o sistema tem seus requisitos especificados e analisados, a arquitetura do sistema é definida e apenas o projeto detalhado, a implementação e os testes são realizados em vários ciclos. Uma vez que nessas duas variações os requisitos são especificados para o sistema como um todo, sua adoção requer que os requisitos sejam estáveis e bem definidos.

O modelo incremental é particularmente útil quando não há pessoal suficiente para realizar o desenvolvimento dentro dos prazos estabelecidos ou para lidar com riscos técnicos, tal como a adoção de uma nova tecnologia [1a].

Dentre as vantagens do modelo incremental, podem ser citadas [5]:

- Menor custo e menos tempo são necessários para se entregar a primeira versão;
- Os riscos associados ao desenvolvimento de um incremento são menores, devido ao seu tamanho reduzido;
- O número de mudanças nos requisitos pode diminuir devido ao curto tempo de desenvolvimento de um incremento.

Como desvantagens, podemos citar [5]:

- Se os requisitos não são tão estáveis ou completos quanto se esperava, alguns incrementos podem ter de ser bastante alterados;
- A gerência do projeto é mais complexa, sobretudo quando a divisão em subsistemas inicialmente feita não se mostrar boa.

O Modelo RAD

O modelo RAD (*Rapid Application Development*), ou modelo de desenvolvimento rápido de aplicações, é um tipo de modelo incremental que prima por um ciclo de desenvolvimento curto (tipicamente de até 90 dias) [1a]. Assim, como no modelo incremental, o sistema é subdividido em subsistemas e incrementos são realizados. A diferença marcante é que os incrementos são desenvolvidos em paralelo por equipes distintas e apenas uma única entrega é feita, como mostra a Figura 2.5.

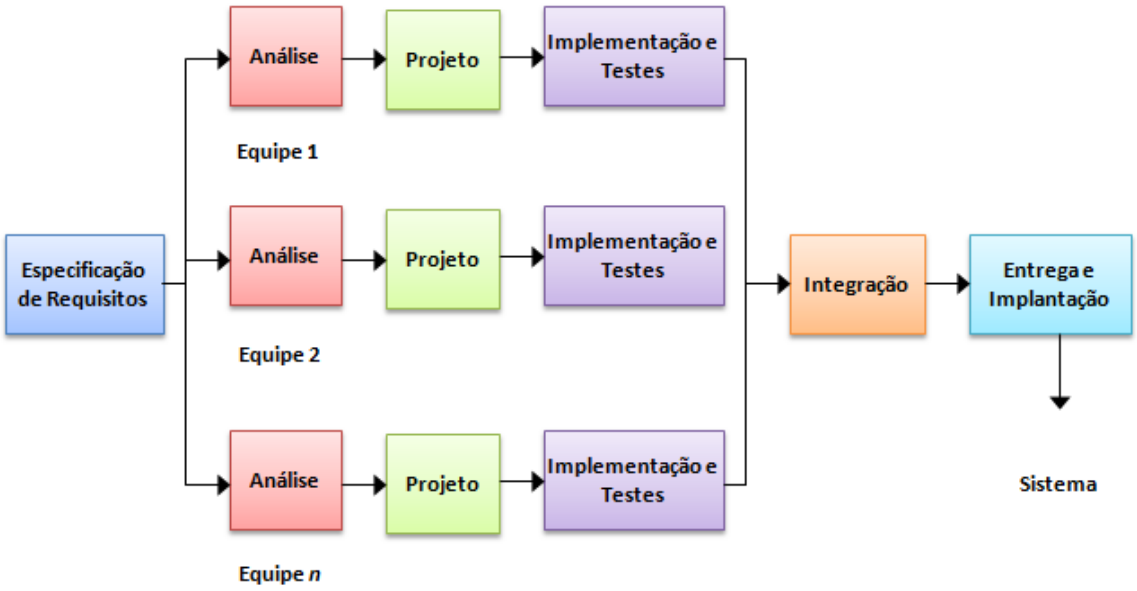

Figura 2.5 – O Modelo RAD.

Assim, como o modelo incremental, o modelo RAD permite variações. Em todos os casos, no entanto, os requisitos têm de ser bem definidos, o escopo do projeto tem de ser restrito e o sistema modular. Se o projeto for grande, por exemplo, o número de equipes crescerá demais e a atividade de integração tornar-se-á por demais complexa. Obviamente, para adotar esse modelo, uma organização tem de ter recursos humanos suficientes para acomodar as várias equipes.

2.2.3 – Modelos Evolutivos ou Evolucionários

Sistemas de software, como quaisquer sistemas complexos, evoluem ao longo do tempo. Seus requisitos, muitas vezes, são difíceis de serem estabelecidos ou mudam com freqüência ao longo do desenvolvimento [1a]. Assim, é importante ter como opção modelos de ciclo de vida que lidem com incertezas e acomodem melhor as contínuas mudanças. Alguns modelos incrementais, dado que preconizam um desenvolvimento iterativo, podem ser aplicados a esses casos, mas a grande maioria deles toma por pressuposto que os requisitos são bem definidos. Modelos evolucionários ou evolutivos buscam preencher essa lacuna.

Enquanto modelos incrementais têm por base a entrega de versões operacionais desde o primeiro ciclo, os modelos evolutivos não têm essa preocupação. Muito pelo contrário: na maioria das vezes, os primeiros ciclos produzem protótipos ou até mesmo apenas modelos. À medida que o desenvolvimento avança e os requisitos vão ficando mais claros e estáveis, protótipos vão dando lugar a versões operacionais, até que o sistema completo seja construído. Assim, quando o problema não é bem definido e ele não pode ser totalmente especificado no início do desenvolvimento, deve-se optar por um modelo evolutivo. A avaliação ou o uso do protótipo / sistema pode aumentar o conhecimento sobre o produto e melhorar o entendimento que se tem acerca dos requisitos. Entretanto, é necessária uma forte gerência do projeto e de configuração.

O Modelo Espiral

O modelo espiral, mostrado na Figura 2.6, é um dos modelos evolutivos mais difundidos.

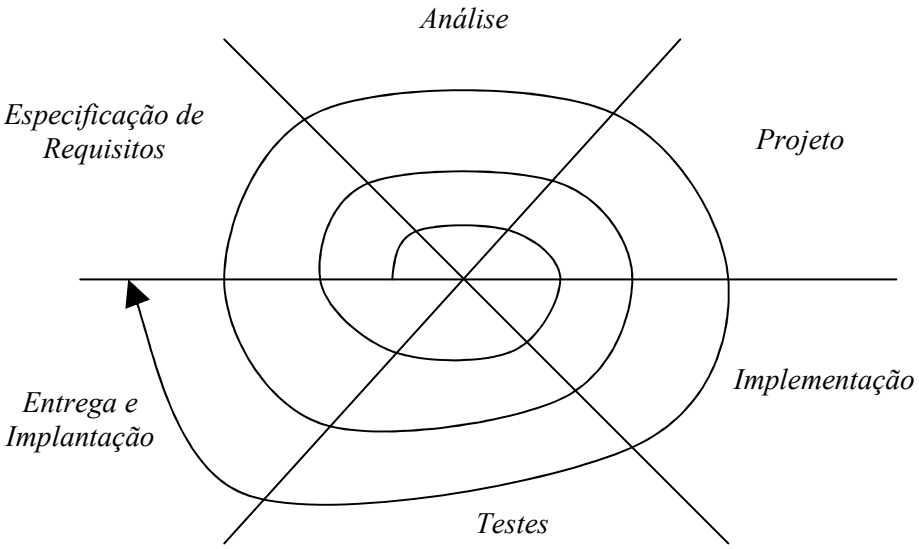

Figura 2.6 – O Modelo Espiral.

Usando o modelo espiral, o sistema é desenvolvido em ciclos, sendo que nos primeiros ciclos nem sempre todas as atividades são realizadas. Por exemplo, o produto resultante do primeiro ciclo pode ser uma especificação do produto ou um estudo de viabilidade. As passadas subsequentes ao longo da espiral podem ser usadas para desenvolver protótipos, chegando progressivamente a versões operacionais do software, até se obter o produto completo. Até mesmo ciclos de manutenção podem ser acomodados nesta filosofia, fazendo com que o modelo espiral contemple todo o ciclo de vida do software [1a].

É importante ressaltar que, a cada ciclo, o planejamento deve ser revisto com base no feedback do cliente, ajustando, inclusive, o número de iterações planejadas. De fato, este é o maior problema do ciclo de vida espiral, ou de maneira geral, dos modelos evolucionários: a gerência de projetos. Pode ser difícil convencer clientes, especialmente em situações envolvendo contrato, que a abordagem evolutiva é gerenciável [1a].

2.2.4 – Prototipação

Muitas vezes, clientes têm em mente um conjunto geral de objetivos para um sistema de software, mas não são capazes de identificar claramente as funcionalidades ou informações (requisitos) que o sistema terá de prover ou tratar. Modelos podem ser úteis para ajudar a levantar e validar requisitos, mas pode ocorrer dos clientes e usuários só terem uma verdadeira dimensão do que está sendo construído se forem colocados diante do sistema. Nestes casos, o uso da prototipação é fundamental. A prototipação é uma técnica para ajudar engenheiros de software e clientes a entender o que está sendo construído quando os

requisitos não estão claros. Ainda que tenha sido citada anteriormente no contexto do modelo espiral, ela pode ser aplicada no contexto de qualquer modelo de processo. A Figura 2.7, por exemplo, ilustra um modelo em cascata com prototipação [3].

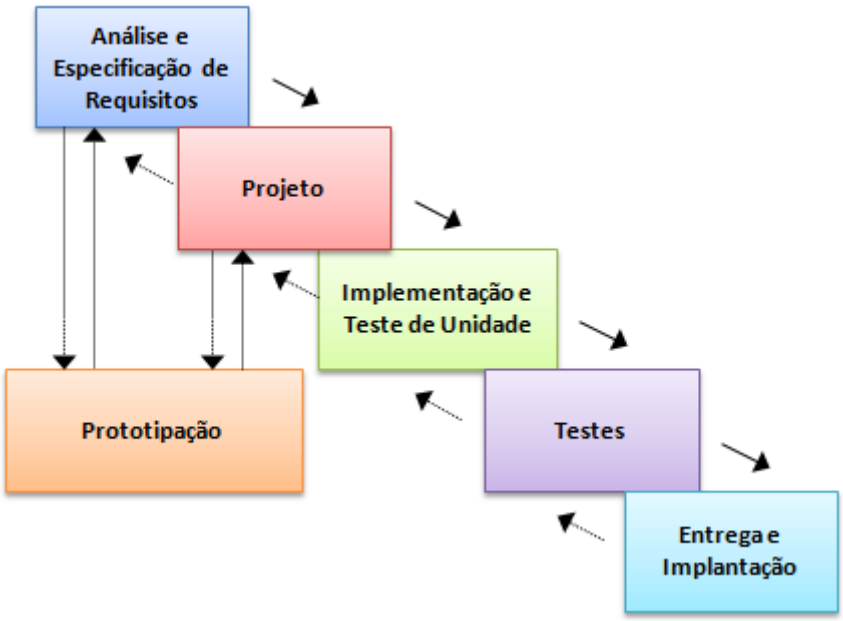

Figura 2.7 - O Modelo em Cascata com Prototipação.

2.3 – Normas e Modelos de Qualidade de Processo de Software

Conforme discutido anteriormente, os modelos de ciclo de vida se atêm apenas às atividades básicas do processo de desenvolvimento e suas inter-relações. Eles se constituem em um importante ponto de partida para a definição de processos, mas não são suficientes. Afinal, um processo de software é, na verdade, um conjunto de processos, dentre eles o processo de desenvolvimento. Mas há outros, tais como os processos de gerência de projetos e de garantia da qualidade. Para o sucesso na definição e melhoria dos processos de software, é fundamental que vários aspectos sejam considerados. Vários modelos e normas de qualidade de processo têm surgido com o objetivo de apoiar a busca por processos de maior qualidade, apontando os principais aspectos que um processo de qualidade deve considerar. Dentre essas normas e modelos destacam-se as normas Série NBR ISO 9000 [6], NBR ISO/IEC 12207 [7, 8], ISO/IEC 15504 [9] e os modelos CMMI [10] e MPS.BR [11].

A Série ISO 9000

As normas da família NBR ISO 9000 [6] foram desenvolvidas para apoiar organizações, de todos os tipos e tamanhos, na implementação e operação de sistemas eficazes de gestão da qualidade.

Um breve histórico:

- A 1ª versão foi publicada em 1987 (9000, 9001, 9002, 9003) - *foco no produto.*
- Em 1994 ocorreu a primeira revisão, visando melhorar os requisitos e enfatizar a natureza preventiva da garantia da qualidade – *foco nas inspeções.*
- Em 2000 ocorreu a segunda revisão, mais adequada aos princípios de Controle da Qualidade Total e com maior foco no cliente (9000, 9001, 9004) – *foco no cliente.*
- Em 2005 foram realizadas algumas revisões pontuais (apenas ISO 9000).
- Em 2008 ocorreu uma revisão da ISO 9001, a fim de compatibilizá-la com a ISO 14000 (Gestão Ambiental) e permitir exclusão de requisitos.
- Em 2009 ocorreu uma revisão da ISO 9004, que passou a incluir o conceito de sucesso sustentável[1] – *foco na melhoria contínua e sustentável*.

  "*A sustentabilidade é o resultado da capacidade da organização em alcançar seus objetivos em longo prazo com análise equilibrada das necessidades e expectativas das partes interessadas." (Boletim ABNT, 10/2009)*

As normas que compõem a série NBR ISO 9000 são:

- NBR ISO 9000: 2005 - *Sistemas de Gestão da Qualidade - Conceitos e Terminologia:* descreve os fundamentos de sistemas de gestão da qualidade e estabelece a terminologia para esses sistemas;
- NBR ISO 9001:2008 - *Sistemas de Gestão da Qualidade – Requisitos:* especifica os requisitos para um sistema de gestão da qualidade com enfoque na satisfação do cliente. Para uma organização ser certificada ISO 9001, ela precisa demonstrar sua capacidade para fornecer produtos que atendam aos requisitos do cliente (explícitos e implícitos) e os requisitos regulamentares aplicáveis;
- NBR ISO 9004:2010 - *Gestão para o sucesso sustentado de uma organização — Uma Abordagem da Gestão da Qualidade (ISO 9004:2009):* fornece diretrizes que ampliam os requisitos estabelecidos pela ISO 9001, buscando melhoria contínua de desempenho e sucesso sustentável.
- NBR ISO 19011:2002 - Diretrizes para Auditoria de SGQ e/ou Ambiental: fornece diretrizes para a condução das auditorias e determinação da competência dos auditores.

A ISO 9000 é de caráter geral, ou seja, não se destina especificamente à indústria de software e estabelece requisitos mínimos da garantia da qualidade que devem ser atendidos pelos fornecedores de produtos ou serviços. Ela é uma norma certificadora. Essa certificação, mundialmente reconhecida, é feita por organismos certificadores, em geral, credenciados por organismos nacionais de acreditação, no caso do Brasil, o INMETRO. Assim, a conquista da certificação ISO 9000 por uma empresa significa que a mesma alcançou um padrão internacional de qualidade em seus processos [4].

O principal problema para se adotar essa norma é precisamente o fato dela ser geral. Assim, quando aplicada ao contexto da indústria de software, muitos problemas surgem pela falta de diretrizes mais focadas nas características de processos de software. Assim, de

[1] No Brasil, a publicação dessa versão da ISO 9004 como a NBR ISO 9004 ocorreu em 2010.

maneira geral, outras normas e modelos de qualidade são usadas por organizações de software para apoiar uma certificação ISO 9000, com destaque para a norma NBR ISO/IEC 12207.

A Norma ISO/IEC 12207

A Norma ISO/IEC 12207 – Tecnologia da Informação – Processos de Ciclo de Vida de Software [7, 8] estabelece uma estrutura comum para os processos de ciclo de vida de software, com terminologia bem definida, que pode ser referenciada pela indústria de software. A estrutura contém processos, atividades e tarefas que devem ser aplicados na aquisição, fornecimento, desenvolvimento, operação e manutenção de produtos de software. Esse conjunto de processos, atividades e tarefas foi projetado para ser adaptado de acordo com as características de cada projeto de software, o que pode envolver o detalhamento, a adição e a supressão de processos, atividades e tarefas não aplicáveis ao mesmo.

Em outubro de 2002 e de 2004, a ISO/IEC 12207 recebeu duas emendas (emendas 1 e 2, respectivamente) para tratar a evolução da Engenharia de Software e, também, para harmonizá-la com a norma ISO/IEC 15504. Basicamente, essas melhorias criaram novos ou expandiram o escopo de alguns processos, além de serem adicionados, a cada processo, o seu propósito e resultados. Assim, agora, cada processo tem um propósito e um resultado associados, além de atividades e tarefas.

Em 2006 houve uma nova revisão para alinhamento com a ISO/IEC 15288 (Engenharia de Sistemas – Processos de Ciclo de Vida de Sistemas) e em 2008 uma nova versão foi publicada (padrão ISO/IEC e IEEE) unindo as alterações das emendas anteriores.

Antes da versão publicada em 2008, a ISO/IEC 12207 possuía 23 processos, agrupados em três categorias de processo (fundamentais, de apoio e organizacionais), mais um processo de adaptação, que, como o nome indica, trata da adaptação da norma à cultura e aos aspectos específicos de uma organização. A Figura 2.8 apresenta os processos presentes na ISO/IEC 12207 [7]. Os processos em vermelho foram adicionados pela Emenda 1 (2002).

<table>
<tr><td colspan="2">Processos Fundamentais</td><td>Processos de Apoio</td><td rowspan="14">Processo de Adaptação</td></tr>
<tr><td colspan="2">Aquisição</td><td>Documentação</td></tr>
<tr><td colspan="2">Fornecimento</td><td>Gerência de Configuração</td></tr>
<tr><td rowspan="8">Desenvolvimento</td><td rowspan="4">Operação</td><td>Garantia da Qualidade</td></tr>
<tr><td>Verificação</td></tr>
<tr><td>Validação</td></tr>
<tr><td>Revisão Conjunta</td></tr>
<tr><td rowspan="4">Manutenção</td><td>Auditoria</td></tr>
<tr><td>Usabilidade</td></tr>
<tr><td>Gerência de Resolução de Problemas</td></tr>
<tr><td>Gerência de Solicitação de Mudanças</td></tr>
<tr><td colspan="2">Avaliação do Produto</td></tr>
<tr><td colspan="3">Processos Organizacionais</td></tr>
<tr><td>Gerência</td><td>Engenharia de Domínio</td><td rowspan="3">Melhoria</td></tr>
<tr><td>Gerência de Ativos</td><td>Infraestrutura</td></tr>
<tr><td>Gestão de Programa de Reúso</td><td>Recursos Humanos</td></tr>
</table>

Figura 2.8 - Processos presentes na ISO/IEC 12207 – antes da versão de 2008.

Na versão de 2008, a norma passou a conter 43 processos, agrupados em sete categorias, mais o processo de adaptação. A Figura 2.9 apresenta os processos presentes na versão atual da ISO/IEC 12207 [8].

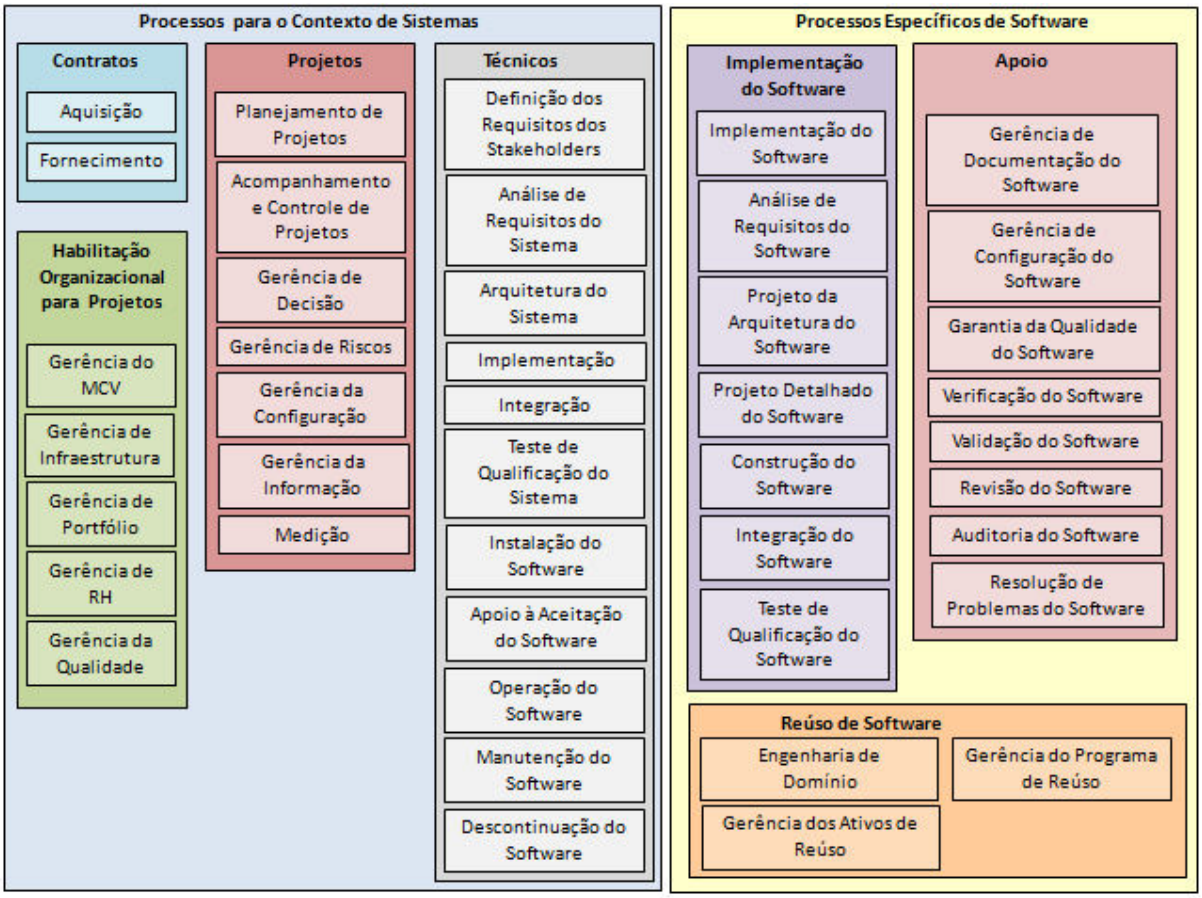

Figura 2.9 - Processos da ISO/IEC 12207:2008.

A Norma ISO/IEC 15504

Desenvolvida pela comunidade internacional em um projeto denominado SPICE (*Software Process Improvement and Capability dEtermination*), a Norma ISO/IEC 15504 – *Information Technology – Process Assessment* [9] (Tecnologia da Informação – Avaliação de Processos) é um padrão internacional ISO para avaliação de processos de software [9].

A ISO/IEC 15504 provê uma abordagem estruturada para avaliação de processos de software com os seguintes objetivos: (i) permitir o entendimento, por ou em favor de uma organização, do estado dos seus processos, visando estabelecer melhorias; (ii) determinar a adequação dos processos de uma organização para atender a um requisito particular ou classe de requisitos; (iii) determinar a adequação de processos da organização para um contrato ou classe de contratos.

Essa norma pode ser usada tanto por adquirentes de software para determinar a capacidade dos processos de software de seus fornecedores, quanto por fornecedores para determinar a capacidade de seus próprios processos ou para identificar oportunidades de melhoria.

É composta de sete partes:

- *Parte 1 (2004): Conceitos e Vocabulário:* provê uma introdução geral dos conceitos de avaliação de processos e um glossário de termos relacionados.
- *Parte 2 (2003): Estrutura do Processo de Avaliação (Normativa):* estabelece os requisitos mínimos para se realizar uma avaliação. Essa parte é a única que tem caráter normativo.
- *Parte 3 (2004): Recomendações para Realização de uma Avaliação:* provê orientações para se interpretar os requisitos para realização de uma avaliação.
- *Parte 4 (2004): Recomendações para Melhoria de Processos e Determinação de Capacidade:* fornece orientações para utilização dos resultados de uma avaliação nos contextos melhoria de processo e determinação da capacidade de processo.
- *Parte 5 (2006): Exemplo de Aplicação para Processos de Ciclo de Vida de Software:* contém um exemplo de modelo de avaliação de processo baseado no modelo de referência da ISO/IEC 12207.
- *Parte 6 (2008): Exemplo de Aplicação para Processos de Ciclo de Vida de Sistema* contém um exemplo de modelo de avaliação de processo baseado no modelo de referência da ISO/IEC 15288.
- *Parte 7 (2008): Condições para uma Avaliação de Maturidade Organizacional:* define um *framework* para avaliar e determinar a maturidade organizacional baseado em perfis de capacidade de processos derivados do processo de avaliação e define as condições necessárias para que tal avaliação seja válida.

O Modelo CMMI

O Modelo de Maturidade e Capacidade Integrado (*Capability Maturity Model Integration - CMMI*) [10] foi desenvolvido no Instituto de Engenharia de Software (*Software Engineering Institute - SEI*) da Universidade de Carnegie Mellon, com o intuito de quantificar a capacidade de uma organização produzir produtos de software de alta qualidade, de forma previsível e consistente. Sua motivação inicial foi apontar as necessidades de melhoria nos projetos de desenvolvimento de software do Departamento de Defesa dos EUA.

O CMMI é estruturado em cinco níveis de maturidade, de 1 a 5, onde o nível 1 é o menos maduro e o nível 5 é o mais maduro. Cada nível de maturidade, com exceção do nível 1, é composto de várias áreas de processo (*process areas* - PAs). As características de cada nível são apresentadas a seguir.

- Nível 1 – Inicial: O processo de software é caracterizado como *ad hoc* e, eventualmente, caótico. Poucos processos são definidos e o sucesso depende de esforços individuais. Neste nível, a organização, tipicamente, opera sem formalizar procedimentos, estimativas de custo e planos de projeto. Ferramentas não são bem integradas ao processo ou não são uniformemente aplicadas. O controle de alterações é superficial e há pouco entendimento sobre os problemas.
- Nível 2 – Gerenciado: Os processos básicos de gerência são estabelecidos para acompanhar custo, cronograma e funcionalidade. Os sucessos em projetos anteriores com aplicações similares podem ser repetidos.

- Nível 3 – Definido: A organização possui um processo padrão definido que é usado como base para todos os projetos. As atividades de engenharia e gerência de software são estáveis e há um entendimento comum e amplo das atividades, papéis e responsabilidades no processo.
- Nível 4 – Gerenciado Quantitativamente: A organização fixa metas quantitativas de qualidade para produtos e processos e fornece instrumentos para medições consistentes e bem definidas. Tanto o processo de software como os produtos são quantitativamente entendidos e controlados. É possível que a organização preveja tendências na qualidade do processo e dos produtos, dentro de fronteiras quantificadas.
- Nível 5 – Otimizado: A organização possui uma base para melhoria contínua e otimização do processo. Dados sobre a eficiência de um processo são usados para efetuar análises custo-benefício de novas tecnologias e para propor mudanças no processo.

A Figura 2.10 mostra os níveis de maturidade do CMMI e, em seguida, na Tabela 2.1 são apresentadas as áreas de processo de cada nível.

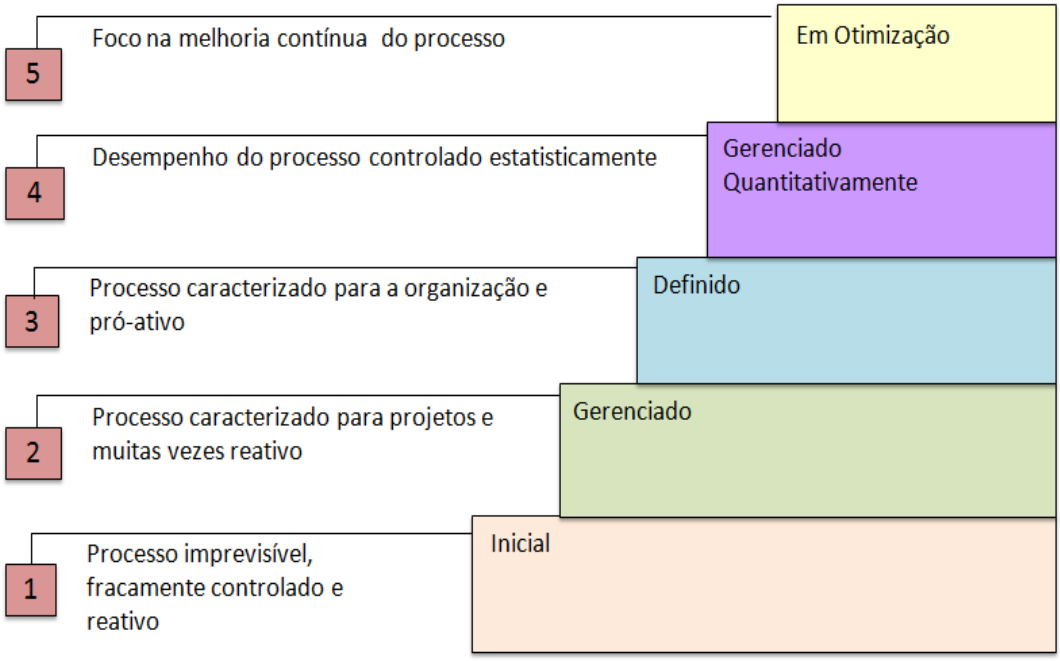

Figura 2.10 - Os níveis de maturidade do CMMI.

Tabela 2.1 – As áreas de processo dos níveis de maturidade do CMMI.

Nível	Áreas de Processo
2	Monitoração e Controle do Projeto (PMC) Gerência de Requisitos (REQM) Análise e Medição (MA) Garantia da Qualidade do Processo e do Produto (PPQA) Gerência de Configuração (CM) Gerência de Fornecedor Integrada (SAM)
3	Gerência de Projeto Integrada (IPM) Gerência de Riscos (RSKM) Definição do Processo Organizacional (OPD) Foco no Processo Organizacional (OPF) Treinamento Organizacional (OT) Desenvolvimento de Requisitos (RD) Integração do Produto (PI) Solução Técnica (TS) Validação (VAL) Verificação (VER) Análise de Decisão e Resolução (DAR)
4	Gerência Quantitativa do Projeto (QPM) Desempenho do Processo Organizacional (OPP)
5	Gerência do Desempenho Organizacional (OPM) Resolução e Análise Causal (CAR)

O CMMI oferece duas abordagens diferentes para a melhoria de processos, conhecidas como o "modelo contínuo" e o "modelo em estágios". A representação em estágio visa uma abordagem relativa à maturidade da organização. Já a representação contínua mapeia níveis de capacitação de cada área de processo, procurando se alinhar com a norma ISO/IEC 15504. Ambas as representações contém áreas de processos comuns. Porém, na representação em estágio, as Áreas de Processos são agrupadas em níveis de maturidade, enquanto na representação contínua as mesmas Áreas de Processo estão divididas em categorias.

O Modelo de Referência Brasileiro – MPS.BR

O MPS.BR – Melhoria de Processo do Software Brasileiro [11] tem como objetivo definir um modelo de melhoria e avaliação de processo de software, adequado, preferencialmente, às micro, pequenas e médias empresas brasileiras, de forma a atender as suas necessidades de negócio e a ser reconhecido nacional e internacionalmente como um modelo aplicável à indústria de software. Por este motivo, está aderente a modelos e normas internacionais.

O MPS.BR também define regras para sua implementação e avaliação, dando sustentação e garantia de que é empregado de forma coerente com as suas definições.

A base técnica utilizada para a construção do MPS.BR é composta pelas normas NBR ISO/IEC 12207 e suas emendas 1 e 2 e a ISO/IEC 15504, estando totalmente aderente a essas normas. Além disso, o MPS.BR também cobre o conteúdo do CMMI.

O MPS.BR está dividido em três componentes:

- Modelo de Referência (MR-MPS): contém os requisitos que as organizações deverão atender para estar em conformidade com o MPS.BR. Define, também, os níveis de maturidade e da capacidade de processos e os processos em si.
- Método de Avaliação (MA-MPS): contém o processo de avaliação, os requisitos para os avaliadores e os requisitos para averiguação da conformidade ao modelo MR-MPS. Está descrito de forma detalhada no Guia de Avaliação e foi baseado na norma ISO/IEC 15504.
- Modelo de Negócio (MN-MPS): contém uma descrição das regras para a implementação do MR-MPS pelas empresas de consultoria, de software e de avaliação.

O MR-MPS define sete níveis de maturidade: A (Em Otimização), B (Gerenciado Quantitativamente), C (Definido), D (Largamente Definido), E (Parcialmente Definido), F (Gerenciado) e G (Parcialmente Gerenciado). A escala de maturidade se inicia no nível G e progride até o nível A. Para cada um desses sete níveis de maturidade, foi atribuído um perfil de processos e de capacidade de processos que indicam onde a organização tem que colocar esforço para melhoria de forma a atender os objetivos de negócio. A Tabela 2.2 mostra os níveis de maturidade do MPS.BR e seus processos.

Tabela 2.2 – Níveis de Maturidade e Processos do MPS.BR.

Nível	Processos
A	Análise de Causas de Problemas e Resolução
B	Gerência de Projetos (evolução)
C	Análise de Decisão e Resolução Desenvolvimento para Reutilização Gerência de Riscos Gerência de Reutilização (evolução)
D	Desenvolvimento de Requisitos Projeto e Construção do Produto Integração do Produto Verificação Validação
E	Avaliação e Melhoria de Processo Organizacional Definição do Processo Organizacional Gerência de Recursos Humanos Gerência de Reutilização Gerência de Projetos (evolução)
F	Medição (para monitoração e controle) Gerência de Configuração Aquisição Garantia da Qualidade Gerência de Portfólio
G	Gerência de Requisitos Gerência de Projetos

2.4 – Processo Padrão da Organização e Processos Padrão Especializados

Vários dos modelos e normas de qualidade de processo discutidos anteriormente preconizam que, embora diferentes projetos requeiram processos com características específicas para atender às suas particularidades, é possível estabelecer um conjunto de ativos de processo (subprocessos, atividades, subatividades, artefatos, recursos e procedimentos) a ser utilizado na definição de processos de software de uma organização. Essas coleções de ativos de processo de software constituem os chamados processos padrão de desenvolvimento de software. Processos para projetos específicos podem, então, ser definidos a partir da instanciação do processo de software padrão da organização, levando em consideração suas características particulares. Esses processos instanciados são ditos processos de projeto.

De fato, o modelo de definição de processos baseado em processos padrão pode ser estendido para comportar vários níveis. Primeiro, pode-se definir um processo padrão da organização, contendo os ativos de processo que devem fazer parte de **todos** os processos de projeto da organização. Esse processo padrão pode ser especializado para agregar novos ativos de processo, considerando aspectos, tais como tipos de software, paradigmas ou domínios de aplicação. Assim, obtêm-se processos mais completos, que consideram características da especialização desejada. Por fim, a partir de um processo padrão ou de um processo especializado, é possível instanciar um processo de projeto, que será o processo a ser utilizado em um projeto de software específico. Para definir esse processo, devem ser consideradas as particularidades de cada projeto. A Figura 2.11 ilustra essa abordagem de definição de processos de software em níveis [4].

Uma vez que objetivo das normas e modelos de qualidade é apontar características que um bom processo de software tem de apresentar, deixando a organização livre para estruturar essas características segundo sua própria cultura, elas são uma importante base para a definição dos processos padrão das organizações. Assim, usando essas normas e modelos de qualidade em uma abordagem de definição de processos em níveis, é possível definir processos para projetos específicos, que levem em consideração as particularidades de cada projeto, sem, no entanto, desconsiderar aspectos importantes para se atingir a qualidade do processo.

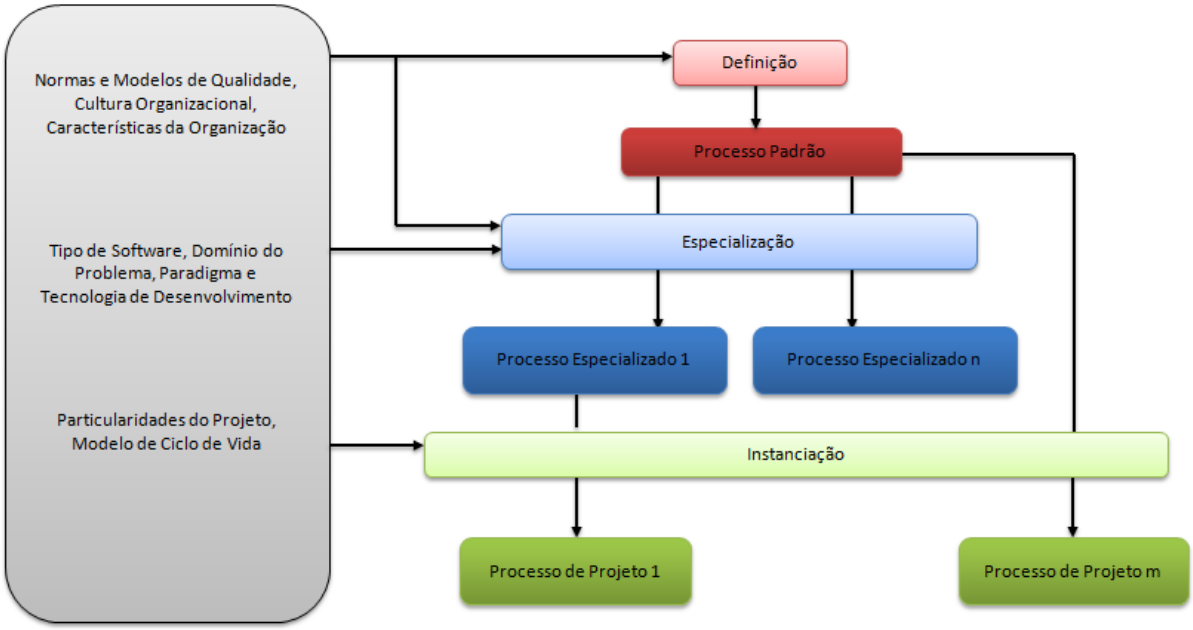

Figura 2.11 – Modelo para Definição de Processos em Níveis.

Sob o ponto de vista do conhecimento do processo, é essencial que os processos padrão (da organização ou especializados) estejam internalizados nas pessoas, ou seja, os desenvolvedores devem executá-los naturalmente. Além disso, o processo padrão organizacional deve estar institucionalizado, isto é, toda a organização deve executá-lo.

2.5 – Automatização do Processo de Software

Com o aumento da complexidade dos processos de software, passou a ser imprescindível o uso de ferramentas e ambientes de apoio à realização de suas atividades, visando, sobretudo, a atingir níveis mais altos de qualidade e produtividade. Ferramentas CASE (*Computer Aided Software Engineering*) passaram, então, a ser utilizadas para apoiar a realização de atividades específicas, tais como planejamento e análise e especificação de requisitos [1].

Apesar dos benefícios do uso de ferramentas CASE individuais, atualmente, o número e a variedade de ferramentas têm crescido a tal ponto que levou os engenheiros de software a pensarem não apenas em automatizar os seus processos, mas sim em trabalhar com diversas ferramentas que interajam entre si e forneçam suporte a todo ciclo de vida do desenvolvimento, dando origem ao Ambientes de Desenvolvimento de Software (ADSs).

ADSs buscam combinar técnicas, métodos e ferramentas para apoiar o engenheiro de software na construção de produtos de software, abrangendo todas as atividades inerentes ao processo: gerência, desenvolvimento e controle da qualidade.

Referências

1. R.S. Pressman, *Engenharia de Software*, Rio de Janeiro: McGraw Hill, Tradução da 5ª edição, 2002.

1a. R.S. Pressman, *Software Engineering: A Practitioner's Approach*, McGraw Hill, 6th edition, 2005.

O Capítulo 2 (Processo de Software) da 5ª edição discute o que é processo de software e apresenta alguns dos modelos de processo discutidos anteriormente. No entanto, por ser uma edição antiga (veja referência [1a]), algumas considerações não são válidas. Assim sendo, recomenda-se a leitura dos capítulos 2 e 3 da 6ª edição, ainda não traduzida para o português. No Capítulo 2 da 6ª edição, além de uma discussão sobre o que é processo de software, há também uma apresentação do modelo CMMI. O Capítulo 3 dessa edição, por sua vez, apresenta os principais modelos de processo.

O Capítulo 31 (Engenharia de Software Apoiada por Computador) da 5ª edição trata da automatização do processo de software. Já a 6ª edição [1a] não tem um capítulo explicitamente dedicado a esse tema.

2. I. Sommerville, *Engenharia de Software*, São Paulo: Addison-Wesley, 6ª edição, 2003.

O Capítulo 3 (Processo de Software) discute o que é processo de software e apresenta os principais modelos de processo discutidos anteriormente. Este capítulo tem também uma seção, a seção 3.7, dedicada ao tema apoio automatizado ao processo de software.

3. S.L. Pfleeger, *Engenharia de Software: Teoria e Prática*, São Paulo: Prentice Hall, 2ª edição, 2004.

O Capítulo 2 (Modelagem do Processo e Ciclo de Vida) discute o que é processo de software e apresenta alguns dos modelos de processo discutidos anteriormente.

4. A. R. C. Rocha, J. C. Maldonado, K. C. Weber, *Qualidade de Software: Teoria e Prática*, São Paulo: Prentice Hall, 2001.

O Capítulo 1 (Normas e Modelos de Qualidade de Processo) apresenta sucintamente as principais normas (ISO/IEC 12207, ISO 9000, ISO/IEC 15504) e modelos de qualidade de processo (CMM). Já o Capítulo 12 (Automatização da Definição de Processos de Software) trata do modelo para definição de processos de software em níveis.

5. M.J. Christensen, R.H. Thayer, *The Project Manager's Guide to Software Engineering Best Practices*, Wiley-IEEE Computer Society Press, 2002.
6. NBR ISO 9000 – Sistemas de Gestão da Qualidade, ABNT, 2005.
7. NBR ISO/IEC 12207 – Tecnologia da Informação – Processos de Ciclo de Vida, ABNT 1998, Emenda 1: 2002, Emenda 2: 2004.

8. ISO/IEC 12207 (2008). Systems and Software Engineering 7 Software Life Cycle Process. International Organization for Standardization and the International Electrotechnical Commission. Geneva, Switzerland.
9. ISO/IEC 15504 – Information Technology - Process Assessment, 2003/2004.
10. SEI – Software Engineering Institute, *Capability Maturity Model Integration for Development - CMMI-DEV (1.3),* 2010.
11. SOFTEX, MPS.BR – Melhoria de Processo do Software Brasileiro – Guia Geral, 2009.

Capítulo 3 – Gerência de Projetos de Software

Antes de contratar o desenvolvimento de um software, geralmente, um cliente quer saber se seu fornecedor é capaz de realizar esse trabalho, quanto o projeto custará e qual será a sua duração. Para responder a essas perguntas, é necessário definir o escopo do projeto, através de um levantamento preliminar de requisitos, realizar estimativas, levantar riscos, alocar recursos e definir um cronograma de execução. Todas essas informações são registradas em um documento, chamado Plano de Projeto, que deve ser sistematicamente revisado ao longo do projeto, de modo a permitir acompanhar o progresso e tomar ações corretivas, no caso de se detectar desvios em relação ao inicialmente planejado. Esse conjunto de atividades faz parte da gerência de projetos de software.

3.1 – Projeto de Software e Gerência de Projetos de Software

Referências: [1] Cap. 3; [2] Cap. 4; [3] Cap. 3, seção 3.2.

Projeto, como definido pelo PMBOK[2], é um empreendimento temporário com o objetivo de criar um produto ou serviço único [4]. É um trabalho que visa à criação de um produto ou à execução de um serviço específico, temporário, não repetitivo e que envolve um certo grau de incerteza na sua realização [5]. Normalmente, é caracterizado por uma seqüência de atividades (o processo do projeto), sendo executada por pessoas dentro de limitações de tempo, recursos (no caso de projetos de software, sobretudo, pessoas) e custos.

Assim sendo, a Gerência de Projetos de Software envolve, dentre outros, o planejamento e o acompanhamento das **pessoas** envolvidas no projeto, do **produto** sendo desenvolvido e do **processo** seguido para evoluir o software de um conceito preliminar para uma implementação concreta e operacional [1]. Uma vez que o processo foi objeto de estudo no capítulo 2, acreditamos que o leitor já está convencido de sua importância para o sucesso de um projeto de software. Passemos, então, a considerações sobre as pessoas e o produto.

Pessoas

Referências: [1] Cap. 3, seção 3.2; [2] Cap. 22; [3] Cap. 3, seção 3.2.

Em um projeto de software, há várias pessoas envolvidas, exercendo diferentes papéis, tais como: Gerente de Projeto, Desenvolvedor (Analistas, Projetistas, Programadores, Engenheiros de Testes), Gerente da Qualidade, Clientes, Usuários. O número de papéis e suas denominações podem ser bastante diferentes dependendo da organização e até mesmo do projeto.

[2] O PMBOK (*Project Management Body of Knowledge* – Corpo de Conhecimento em Gerência de Projetos) é um guia de orientação do conhecimento envolvido na gerência de projetos, cujo objetivo é identificar e descrever conceitos e práticas da gerência de projetos em geral, padronizando a terminologia e os processos adotados nesta área de estudo. Esse documento foi produzido e é periodicamente atualizado pelo PMI (*Project Management Institute* – Instituto de Gerência de Projetos), uma entidade internacional sem fins lucrativos que congrega profissionais atuando na área de gerência de projetos [5].

As pessoas trabalhando em um projeto são organizadas em equipes. Assim, o conceito de equipe pode ser visto como um conjunto de pessoas trabalhando em diferentes tarefas, mas objetivando uma meta comum. Essa não é uma característica do desenvolvimento de software, mas da organização de pessoas em qualquer atividade humana. Assim, a definição de equipes é importante para uma ampla variedade de situações, tal como uma formação de uma equipe de futebol.

Para a boa formação de equipes, devem ser definidos os papéis necessários e devem ser considerados aspectos fundamentais, a saber: liderança, organização (estrutura da equipe) e coordenação. Além disso, há diversos fatores que afetam a formação de equipes: relacionamentos interpessoais, tipo do projeto, criatividade etc.

No que se refere à organização / estrutura das equipes, há diversos tipos de equipes, tais como os citados por Pressman [1]:

- Democrática Descentralizada: Não tem um líder permanente e as decisões são tomadas por consenso do grupo. A comunicação é entre os membros da equipe é horizontal.
- Controlada Descentralizada: Há um líder do projeto, mas a comunicação ainda é horizontal.
- Controlada Centralizada: Há um líder do projeto e a comunicação entre ele e os demais membros da equipe é vertical.

Por fim, na formação de equipes deve-se levar em conta o tamanho da equipe. Quanto maior o número de membros da equipe, maior a quantidade de caminhos possíveis de comunicação, o que pode ser um problema, uma vez que o número de pessoas que podem se comunicar com outras pode afetar a qualidade do produto resultante.

Produto

Referências: [1]: Cap. 3, seção 3.3, Cap. 5, seção 5.3; [3]: Cap. 3, seção 3.1.

Na gerência de projetos, um gerente se depara, logo no início, com um sério problema: são necessárias estimativas quantitativas (de tempo e custo) e um plano organizado do trabalho a ser feito, entretanto, não há informação suficiente para tal. Assim, a primeira coisa a fazer é definir o escopo do software, realizando um levantamento de requisitos inicial. Neste contexto, ganha força a ideia de decompor o problema, em uma abordagem "dividir para conquistar". Inicialmente, o sistema deve ser decomposto em subsistemas que são, por sua vez, decompostos em módulos. Os módulos podem, ainda, ser recursivamente decompostos em submódulos ou funções, até que se tenha uma visão geral das funcionalidades a serem tratadas no projeto. Características especiais relacionadas a essas funções devem ser apontadas, tais como requisitos de desempenho.

Estrutura de Divisão do Trabalho

Referências: [1] Cap. 3, seção 3.4; [2] Cap. 2, seção 2.3; [3] Cap. 3, seção 3.1.

Uma boa gerência de projetos começa com a fusão das visões de produto e processo. Cada função ou módulo a ser desenvolvido pela equipe do projeto deve passar pelas várias

atividades definidas no processo de software. Essa pode ser uma base bastante efetiva para a elaboração de estimativas, incluindo a alocação de recursos, já que é sempre mais fácil estimar porções menores de trabalho. Assim, é útil elaborar uma estrutura de divisão do trabalho (*Work Breakdown Structure* – WBS), considerando essa duas dimensões - produto e processo - como mostra a Tabela 3.1.

Tabela 3.1 – Estrutura de Divisão do Trabalho considerando a fusão das visões de produto e processo.

	Atividades do processo			
Módulos / Funções	Análise e Especificação de Requisitos	Projeto	Implementação	Testes
Módulo 1				
Módulo 2				
....				

3.2 – O Processo da Gerência de Projetos de Software

Conforme discutido no Capítulo 2, o processo de software é composto de diversos processos, dentre eles o processo de gerência de projetos. Tipicamente, um processo de gerência de projetos envolve três atividades principais:

- Planejamento: no início do projeto, um plano organizado de como o projeto será conduzido deve ser elaborado. O planejamento do projeto deve tratar da definição do escopo do software, da definição do processo de software do projeto, da realização de estimativas, da elaboração de um cronograma e da identificação e tratamento dos riscos associados ao projeto.
- Acompanhamento: conforme anteriormente apontado, no início do projeto há pouca informação disponível, o que pode comprometer a precisão do escopo identificado, das estimativas realizadas e, por conseguinte, do cronograma elaborado. À medida que o trabalho avança, maior conhecimento se tem e, portanto, é possível refinar e ajustar esses elementos. Além disso, projetos são dinâmicos e, portanto, estão sujeitos às mudanças que ocorrem no contexto em que o produto será inserido. Sendo assim, é fundamental acompanhar o progresso do trabalho, refinar escopo e estimativas, alterar o processo do projeto e o cronograma, além de monitorar riscos e tomar ações corretivas. Assim sendo, as atividades realizadas (sumarizadas na Figura 3.1) no planejamento, são novamente consideradas no acompanhamento do projeto, que tipicamente se dá nos marcos definidos no projeto.
- Encerramento: terminado o projeto, a gerência ainda tem um importante trabalho a fazer: fazer uma análise crítica do que deu certo e o que não funcionou, procurando registrar lições aprendidas de sucesso e oportunidades de melhoria. Comparações entre valores estimados e realizados, identificação de problemas que

ocorreram e causas dos desvios devem ser discutidas com os membros da equipe, procurando fazer com que haja um aprendizado, não só da equipe, mas da organização como um todo. Uma técnica bastante empregada neste contexto é a análise *post-mortem*.

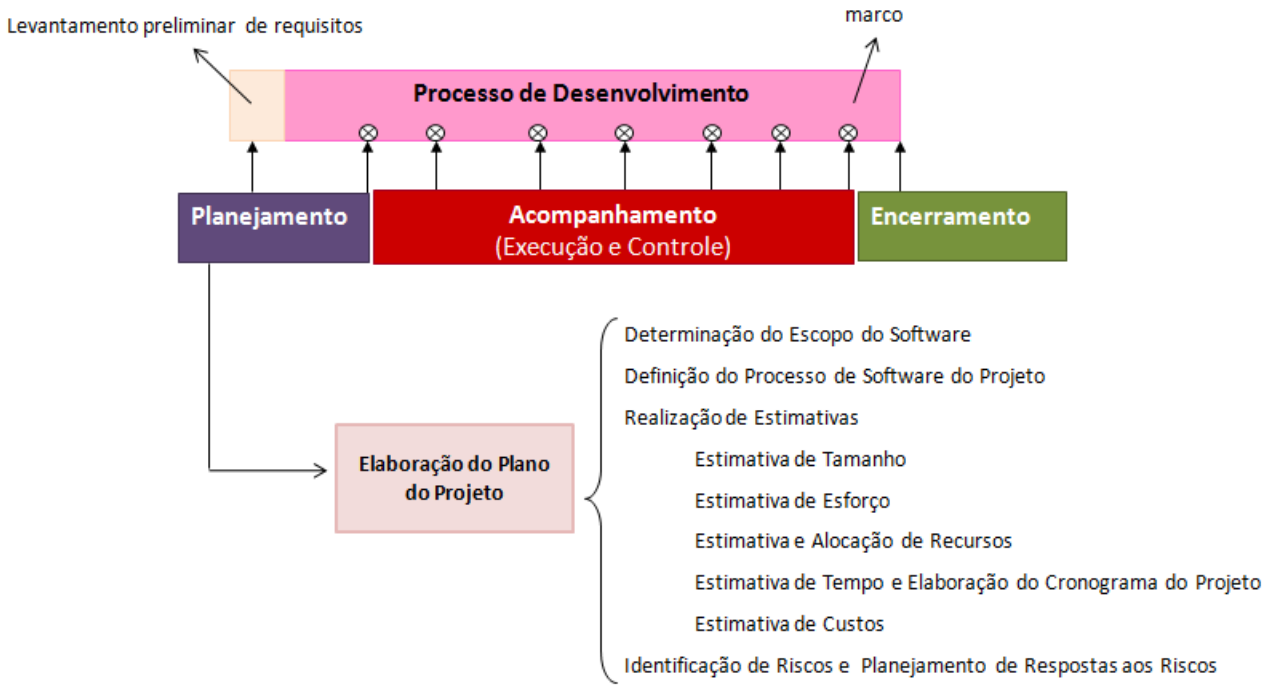

Figura 3.1 – O Processo de Gerência de Projetos de Software.

As atividades relacionadas no quadro na parte inferior na Figura 3.1 são realizadas diversas vezes ao longo do projeto. Tipicamente, no início do projeto, elas têm de ser realizadas para produzir uma primeira visão gerencial sobre o projeto, quando são conjuntamente denominadas de planejamento do projeto. À medida que o projeto avança, contudo, o plano do projeto deve ser revisto, uma vez que problemas podem surgir ou porque se ganha um maior entendimento sobre o problema. Essas revisões do plano de projeto são ditas atividades de acompanhamento do projeto e tipicamente são realizadas nos marcos do projeto.

Os marcos de um projeto são estabelecidos durante a definição do processo e tipicamente correspondem ao término de atividades importantes do processo de desenvolvimento, tais como Análise e Especificação de Requisitos, Projeto e Implementação. O propósito de um marco é garantir que os interessados tenham uma visão do andamento do projeto e concordem com os rumos a serem tomados.

Em uma atividade de acompanhamento do projeto, o escopo pode ser revisto, alterações no processo podem ser necessárias, bem como devem ser monitorados os riscos e revisadas as estimativas (de tamanho, esforço, duração e custo).

Na sequência, cada uma das atividades inerentes ao planejamento e acompanhamento de projetos é discutida, com exceção da definição do processo de software do projeto, já discutida no Capítulo 2.

3.3 – Determinação do Escopo do Software

A primeira atividade de gerência em um projeto de software consiste na determinação do escopo do software a ser desenvolvido [1]. Basicamente, o escopo do produto é composto pela especificação de um conjunto de funcionalidades (requisitos funcionais) associada a outras características desejadas (requisitos não funcionais), tais como desempenho, confiabilidade etc.

Para que o escopo do software seja determinado, um levantamento preliminar de requisitos deve ser realizado[3]. O escopo pode ser documentado de várias formas, sempre contendo uma breve descrição das funções do sistema, requisitos não funcionais importantes e o contexto e objetivos do sistema.

Um instrumento útil para mostrar as funcionalidades do sistema é um diagrama de casos de uso. Em essência, um diagrama de casos de uso mostra o sistema segundo uma perspectiva externa, na qual atores (usuários ou outros sistemas) aparecem interagindo com as funções do sistema (casos de uso), como ilustra a Figura 3.2. A técnica de modelagem de casos de uso será estudada com mais detalhes no capítulo 5.

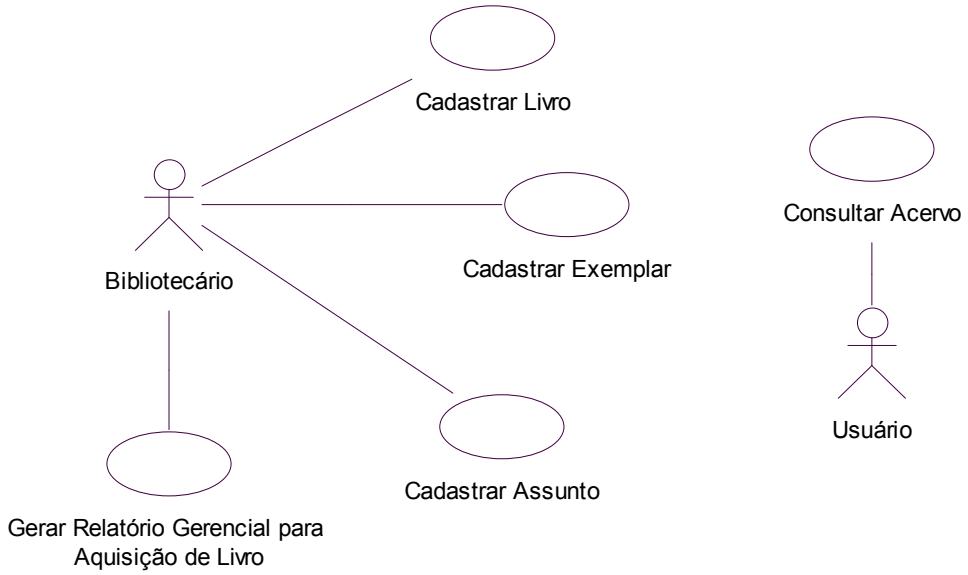

Figura 3.2 – Exemplo de um Diagrama de Casos de Uso Preliminar de um Sistema.

3.4 - Estimativas

Referências: [1] Cap. 5; [2] Cap. 23; [3] Cap. 3, seção 3.3.

Antes mesmo de serem iniciadas as atividades técnicas de um projeto, o gerente e a equipe de desenvolvimento devem estimar o trabalho a ser realizado, os recursos necessários, a duração e, por fim, o custo do projeto. Apesar das estimativas serem um pouco de arte e um pouco de ciência, essa importante atividade não deve ser conduzida desordenadamente. As estimativas podem ser consideradas a fundação para todas as outras atividades de planejamento de projeto.

[3] A atividade de levantamento de requisitos será estudada com mais detalhes no capítulo 5.

Para alcançar boas estimativas de prazo, esforço e custo, existem algumas opções [1]:

1. Postergar as estimativas até o mais tarde possível no projeto.
2. Usar técnicas de decomposição.
3. Usar um ou mais modelos empíricos para estimativas de custo e esforço.
4. Basear as estimativas em projetos similares que já tenham sido concluídos.

A primeira opção, apesar de ser atraente, não é prática, pois estimativas devem ser providas logo no início do projeto (fase de planejamento do projeto). No entanto, deve-se reconhecer que quanto mais tarde for feita a estimativa, maior o conhecimento do projeto e menores as chances de se cometer erros. Assim, é fundamental revisar as estimativas na medida em que o projeto avança (atividades de acompanhamento do projeto).

Técnicas de decomposição, a segunda opção, usam, conforme discutido anteriormente, a abordagem "dividir para conquistar" na realização de estimativas, através da decomposição do projeto em módulos / funções (decomposição do produto) e atividades mais importantes (decomposição do processo). Assim, a estrutura de divisão de trabalho, como a mostrada na Tabela 3.1, pode ser utilizada para estimar, por exemplo, tamanho ou esforço.

Modelos empíricos, tipicamente, usam fórmulas matemáticas, derivadas em experimentos, para prever esforço como uma função de tamanho (linhas de código ou pontos de função). Entretanto, deve-se observar que os dados empíricos que suportam a maioria desses modelos são derivados de um conjunto limitado de projetos. Além disso, fatores culturais da organização não são considerados no uso de modelos empíricos, pois os projetos que constituem a base de dados do modelo são externos à organização. Apesar de suas limitações, modelos empíricos podem ser úteis como um ponto de partida para organizações que ainda não têm dados históricos, até que a organização possa estabelecer suas próprias correlações.

Finalmente, na última opção, dados de projetos anteriores armazenados em um repositório de experiências da organização podem prover uma perspectiva histórica importante e ser uma boa fonte para estimativas. Através de mecanismos de busca, é possível recuperar projetos similares, suas estimativas e lições aprendidas, que podem ajudar a elaborar estimativas mais precisas. Nesta abordagem, os fatores culturais são considerados, pois os projetos foram desenvolvidos na própria organização.

Vale frisar que essas abordagens não são excludentes; muito pelo contrário. O objetivo é ter várias formas para realizar estimativas e usar seus resultados para se chegar a estimativas mais precisas.

Quando se fala em estimativas, está-se tratando na realidade de diversos tipos de estimativas: tamanho, esforço, recursos, tempo e custos. Geralmente, a realização de estimativas começa pelas estimativas de tamanho. A partir delas, estima-se o esforço necessário e, na sequência, alocam-se os recursos necessários, elabora-se o cronograma do projeto (estimativa de duração) e, por fim, estima-se o custo do projeto.

3.4.1 – Gerência de Projetos e Medição

É muito importante observar a estreita relação entre gerência de projetos e medição. Para acompanhar o andamento do projeto, é preciso medir o progresso e comparar com o

estimado. Mesmo no planejamento, sobretudo quando se pretende utilizar dados de projetos anteriores, dados de métricas são muito importantes. Não é possível controlar o que não se pode medir e, principalmente, só é possível chegar a boas estimativas com base em dados históricos se dados forem coletados criteriosamente. Assim, as medidas dão visibilidade ao estado do projeto, permitindo tanto saber para onde ir no início do projeto quanto verificar se o rumo está correto, tomando ações corretivas quando necessário [6].

Mas não basta coletar dados aleatoriamente. Para que possam ser usados eficientemente, esses dados têm de ser arranjados de modo a prover indicadores. Por exemplo, o que se pode dizer a respeito da qualidade de um produto de software que tenha apresentado cinco defeitos depois de implantado? É boa? Não? Isoladamente, esse dado pouco diz. Neste caso, combinar dados em métricas é uma boa opção. No exemplo anterior, se combinássemos a medida de número de defeitos com uma medida de tamanho (tal como linhas de código – LOC), teríamos uma métrica (número de defeitos por linha de código) capaz de nos dizer bem mais do que os dados isolados. Se agora os cinco defeitos medidos fossem em um software de 10.000 linha de código, chegar-se-ia ao valor de 0,5 defeito/KLOC. A partir dessa métrica, comparando-a com indicadores da organização, aí sim poder-se-ia chegar a alguma conclusão sobre a qualidade do produto.

Em uma abordagem desta natureza, os resultados da medição permitem uma comunicação efetiva com os vários interessados no desenvolvimento. A falta de métricas de projeto, por outro lado, prejudica de forma geral o seu acompanhamento, uma vez que pode haver um problema, mas ele não está sendo percebido por aqueles que podem direcionar esforços para a sua solução. Assim, métricas têm um importante papel na rápida identificação e correção de problemas ao longo do desenvolvimento do projeto. Com a sua utilização, fica muito mais fácil justificar e defender as decisões tomadas, afinal o gerente de projeto não decidiu apenas com base em seu sentimento e experiência, mas também fundamentado na avaliação de indicadores que refletem uma tendência de comportamento futuro. Essa tendência não é derivada apenas das experiências no projeto corrente, mas também de experiências semelhantes de outros projetos da organização (conhecimento organizacional) e até mesmo de fora dela [6].

No que tange à gerência de projetos, estabelecer classes de projetos e coletar algumas métricas pode ser bastante importante para apoiar a realização de estimativas. Por exemplo, se uma organização tem indicadores para produtividade (tamanho/esforço) e custo (R$/tamanho) para diversas classes de projetos diferentes, é possível, a partir de uma estimativa de tamanho, chegar a estimativas de esforço e custo. Dada a importância da estimativa de tamanho nessa abordagem, ela é, geralmente, a primeira a ser realizada.

3.4.2 - Estimativa de Tamanho

Referências: [1] Cap. 5, seção 5.6; [2] Cap. 23, seção 23.1; [3] Cap. 3, seção 3.3.

Ainda que anteriormente o tamanho tenha sido basicamente utilizado para normalizar indicadores de produtividade, custo e qualidade, mesmo isoladamente pode ser uma medida importante, como, por exemplo, na contratação de serviços de desenvolvimento e manutenção de software.

Dois modelos de contratação são bastante difundidos atualmente [6]: preço global fixo e por preço unitário por homem-hora. No primeiro, um preço total fixo é estabelecido por um

produto ou serviço bem definido. O grande problema desse modelo é que, normalmente, é alta a probabilidade de haver um aumento do escopo inicialmente previsto. A questão passa a ser: incorporar ou não novos requisitos ao produto? Se a decisão for por incorporar esses requisitos, quem vai pagar a conta? Afinal, aumento do escopo implica em aumento no trabalho a ser realizado. Renegociar contratos nem sempre é fácil. Além disso, as alterações nos requisitos podem ser muitas (e às vezes pequenas), sendo inviável a realização de tantas renegociações. Se a decisão for por não incorporar novos requisitos, o resultado final pode ser ainda mais desastroso: a insatisfação do cliente.

No modelo baseado em homem-hora, o risco passa a ser outro. Se a organização responsável pelo fornecimento do produto ou serviço é pouco produtiva, ela não é penalizada. Muito pelo contrário. Ela recebe ainda mais para fazer o mesmo serviço [6].

Uma forma alternativa para contratação consiste em uma variação do segundo modelo na qual o preço unitário é pago não mais por homem-hora (uma unidade de esforço), mas por uma unidade de tamanho. Assim, é possível acomodar mudanças no esforço, mas sem os desvios observados na modalidade por homem-hora. A questão passa a ser, então, que unidade usar para medir tamanho.

Entre as várias formas de se medir tamanho de um software, uma das mais simples, direta e intuitiva é a contagem do número de linhas de código (*Lines Of Code* - LOC) dos programas fonte. Existem alguns estudos que demonstram a alta correlação entre essa métrica e o tempo necessário para se desenvolver um sistema. Entretanto, o uso dessa métrica apresenta várias desvantagens. Primeiro, verifica-se que ela é fortemente ligada à tecnologia empregada, sobretudo a linguagem de programação e ao estilo do código escrito. Segundo, pode ser difícil estimar essa grandeza no início do desenvolvimento, sobretudo se não houver dados históricos relacionados com a linguagem de programação utilizada no projeto. Por fim, essa é uma métrica que pouco significativa para o cliente.

Visando possibilitar a realização de estimativas de tamanho mais cedo no processo de software e sem os problemas de dependência em relação à tecnologia, foram propostas outras métricas em um nível de abstração mais alto. O exemplo mais conhecido é a contagem de Pontos de Função (PFs), que busca medir as funcionalidades do sistema requisitadas e recebidas pelo usuário, de forma independente da tecnologia usada na implementação.

Análise de Pontos de Função

Referências: [6] [7].

A Análise de Pontos de Função (APF) é um método padrão para a medição do desenvolvimento, manutenção ou tamanho de uma aplicação de software, visando estabelecer uma medida de tamanho do software em Pontos de Função (PFs), com base na quantificação da funcionalidade solicitada ou fornecida, sob o ponto de vista do usuário.

Os objetivos da APF são [7]:

- Medir as funcionalidades do sistema requisitadas e recebidas pelo usuário;
- Medir projetos de desenvolvimento e manutenção de software, sem se preocupar com a tecnologia que será utilizada na implementação.

O processo para contagem de PFs compreende sete passos, mostrados na Figura 3.3 e descritos sucintamente adiante. Uma descrição um pouco mais detalhada do método da Análise de Pontos de Função é apresentada no Anexo A. Para uma visão completa do método, recomenda-se a leitura de [6].

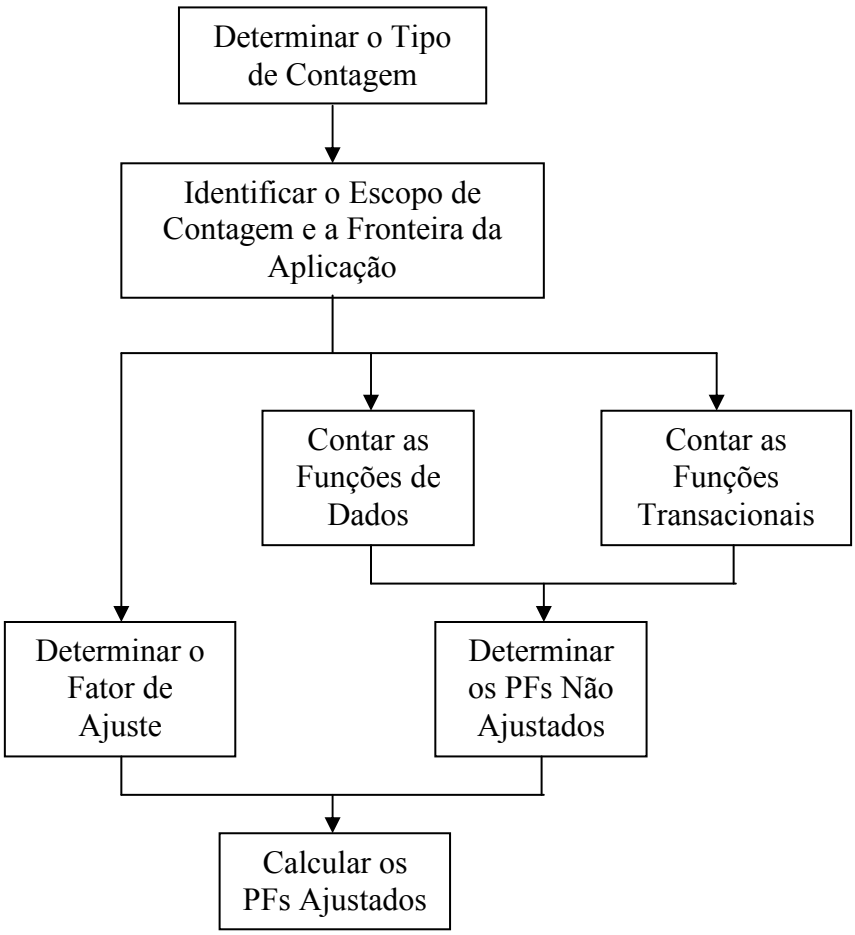

Figura 3.3 – O Processo de Contagem de Pontos de Função.

- **Determinar o tipo de contagem de pontos de função**: este é o primeiro passo no processo de contagem, sendo que existem três tipos de contagem: contagem de PF de projeto de desenvolvimento, de aplicações instaladas e de projetos de manutenção.
- **Identificar o escopo de contagem e a fronteira da aplicação**: neste passo, definem-se as funcionalidades que serão incluídas em uma contagem de PFs específica. A fronteira da aplicação é definida estabelecendo um limite lógico entre a aplicação que está sendo medida, os usuários e outras aplicações. O escopo de contagem define a parte do sistema (funcionalidades) a ser contada.
- **Determinar a contagem de pontos de função não ajustados**: os pontos de função não ajustados (PFNA) refletem as funcionalidades fornecidas pelo sistema para o usuário. Essa contagem leva em conta dois tipos de função: de dados e transacionais, bem como sua complexidade (simples, média ou complexa).
- **Contar as funções de dados**: as funções de dados representam as funcionalidades relativas aos requisitos de dados internos e externos à aplicação. São elas os arquivos lógicos internos e os arquivos de interface externa. Ambos são grupos de dados logicamente relacionados ou informações de controle que foram identificados pelo usuário. A diferença está no fato de um **Arquivo Lógico Interno (ALI)** ser mantido dentro da fronteira da aplicação, isto é, armazenar os dados mantidos através de um ou mais processos elementares da aplicação, enquanto que um **Arquivo de Interface Externa (AIE)** é apenas referenciado pela aplicação, ou seja, ele é mantido dentro da fronteira de outra aplicação. Assim, o objetivo de um AIE é armazenar os dados referenciados por um ou mais processos elementares da aplicação sendo contada, mas que são mantidos por outras aplicações.
- **Contar as funções transacionais**: as funções transacionais representam as funcionalidades de processamento de dados do sistema fornecidas para o usuário. São elas: as entradas externas, as saídas externas e as consultas externas. As **Entradas Externas (EEs)** são processos elementares que processam dados (ou informações de controle) que entram pela fronteira da aplicação. O objetivo principal de uma EE é manter um ou mais ALIs ou alterar o comportamento do sistema. As **Saídas Externas (SEs)** são processos elementares que enviam dados (ou informações de controle) para fora da fronteira da aplicação. Seu objetivo é mostrar informações recuperadas através de um processamento lógico (isto é, que envolva cálculos ou criação de dados derivados) e não apenas uma simples recuperação de dados. Uma SE pode, também, manter um ALI ou alterar o comportamento do sistema. Por fim, uma **Consulta Externa (CE)**, assim como uma SE, é um processo elementar que envia dados (ou informações de controle) para fora da fronteira da aplicação, mas sem realização de nenhum cálculo nem a criação de dados derivados. Seu objetivo é apresentar informação para o usuário, por meio apenas de uma recuperação das informações. Nenhum ALI é mantido durante sua realização, nem o comportamento do sistema é alterado.
- **Determinar o valor do fator de ajuste**: o fator de ajuste é baseado em 14 características gerais de sistemas, que avaliam a funcionalidade geral da aplicação que está sendo contada, e seus níveis de influência. O nível de influência de uma característica é determinado com base em uma escala de 0 (nenhuma influência) a 5 (forte influência). Assim, o fator de ajuste visa a ajustar os pontos de função não

ajustados em ± 35%. Esse passo tornou-se opcional em 2002 para que o método da APF passasse a ser um padrão internacional de medição funcional (ISO/IEC 20926). As principais críticas são a grande variação na interpretação das 14 características gerais de sistemas e a constatação que algumas delas estão desatualizadas.

- **Calcular os pontos de função ajustados**: finalmente, os PFs ajustados são calculados, considerando-se o tipo de contagem definido no primeiro passo e o fator de ajuste.

A Figura 3.4 apresenta uma visão esquemática dos tipos de função que são considerados na contagem da APF.

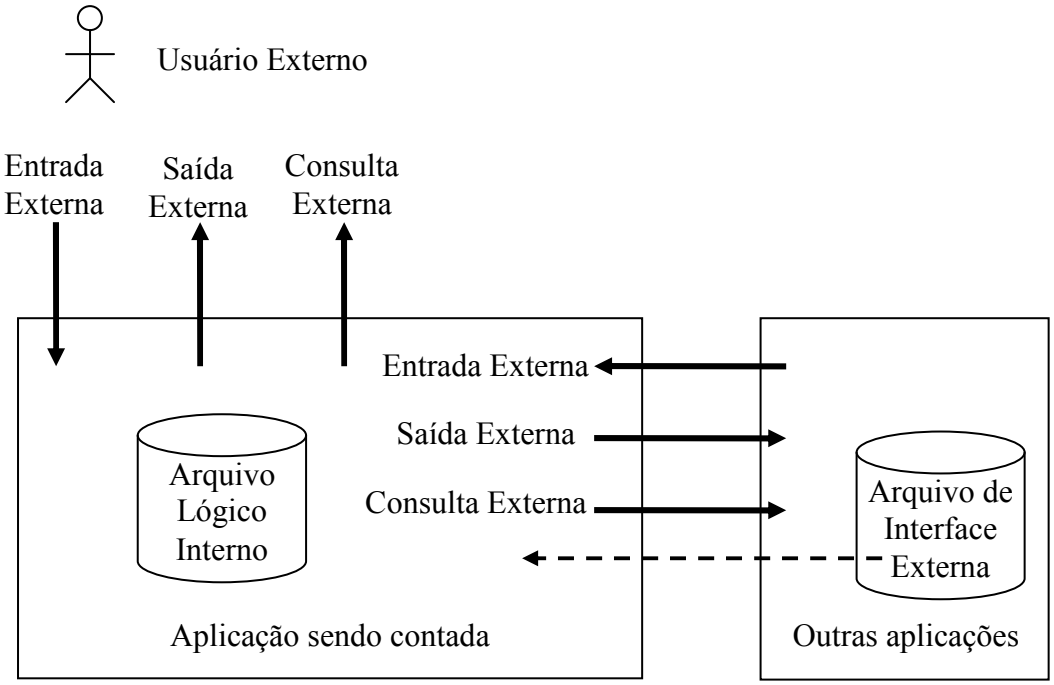

Figura 3.4 – Visão Esquemática das Funções de uma Aplicação segundo a APF.

Ainda que a obtenção dos pontos de função seja dependente unicamente do conhecimento das funcionalidades requeridas e não da tecnologia a ser empregada, o maior problema da APF é que os dados necessários para essa análise são bastante imprecisos no início de um projeto e, portanto, gerentes de projeto são, muitas vezes, obrigados a produzir estimativas antes de um estudo mais aprofundado. Assim, pode ser pouco produtivo utilizar o método da APF integralmente para realizar as primeiras estimativas. A APF é, antes de mais nada, um método de medição e, portanto, seu uso, da forma como proposta, para a realização de estimativas de tamanho é uma adaptação.

Tendo em vista isso, foram propostas algumas variações mais simples do método voltadas para a realização de estimativas e que apresentam resultados satisfatórios, dentre elas a *Contagem Indicativa*, o uso de *Complexidades Médias* e a *Contagem Estimativa*. Todas consideram apenas PFs não ajustados [6].

Na Contagem Indicativa, é necessária apenas a identificação das funções de dados (ALIs e AIEs). Considera-se, então, 35 PFs para cada ALI e 15 PFs para cada AIE

identificados, ou seja o número de PFs não ajustados é dado por: nPF = 35* nALI + 15* nAIE.

Nos casos de Complexidades Médias e Contagem Estimativa, é necessário identificar todas as funções de dados (ALIs e AIEs) e transacionais (EEs, CEs e SEs), mas não é necessário usar as tabelas de complexidade. No caso das Complexidades Médias do ISBSG Benchmark, considera-se 7,4 PFs para cada ALI, 5,5 PFs para cada AIE, 4,3 PFs para cada EE, 5,4 PFs para cada SE e 3,8 PFs para cada CE. Assim, o número de PFs não ajustados é dado por: nPF = 7,4* nALI + 5,5* nAIE + 4,3* nEE + 5,4* nSE + 3,8*nCE. No caso da Contagem Estimativa da NESMA, os pesos são um pouco diferentes, sendo o número de PFs não ajustados dado por: nPF = 7* nALI + 5* nAIE + 4* nEE + 5* nSE + 4*nCE.

Obviamente, a contagem detalhada de pontos de função é mais precisa que os demais tipos de contagem. Entretanto, requer um tempo maior para ser realizado e depende de especificações mais detalhadas que, na grande maioria das vezes, não existem no início de um projeto. Vale destacar que estudos realizados pela NESMA com mais de 100 projetos apontam que a contagem estimativa tem menor dispersão em relação à contagem detalhada do que a contagem indicativa.

Assim, uma boa opção é iniciar as estimativas com uma técnica simplificada, tal como a Contagem Estimativa da NESMA, e, à medida que um maior entendimento dos requisitos é obtido, passar à contagem detalhada. Além disso, os pontos de função devem ser recontados ao longo do processo (nas atividades de acompanhamento de projetos), para que ajustes de previsões possam ser realizados e controlados, fornecendo *feedback* para situações futuras.

3.4.3 - Estimativas de Esforço

Referências: [1] Cap. 5, seções 5.6 e 5.7; [2] Cap. 23, seção 23.1; [3] Cap. 3, seção 3.3.

Para a realização de estimativas de tempo cronológico (duração) e custo, é fundamental estimar, antes, o esforço necessário para completar o projeto ou cada uma de suas atividades. Estimativas de esforço podem ser obtidas diretamente pelo julgamento de especialistas, tipicamente usando técnicas de decomposição, ou podem ser computadas a partir de dados de tamanho ou de dados históricos.

Quando as estimativas de esforço são feitas com base apenas no julgamento dos especialistas, uma tabela como a Tabela 3.1 pode ser utilizada, em que cada célula corresponde ao esforço necessário para efetuar uma atividade no escopo de um módulo específico. Uma tabela como essa pode ser produzida também com base em dados históricos de projetos similares já realizados na organização.

Quando estimativas de tamanho são usadas como base, pode-se considerar um fator de produtividade, indicando quanto em unidades de esforço é necessário para completar um projeto (ou módulo), descrito em unidades de tamanho. Assim, uma organização pode coletar dados de vários projetos e estabelecer, por exemplo, quantos em homens-hora (uma unidade de esforço) são necessários para desenvolver 1000 LOCs (KLOC) ou 1 PF (unidades de tamanho). Esses fatores de produtividade devem levar em conta características dos projetos e da organização. Assim, pode ser útil ter vários fatores de produtividade, considerando classes de projetos específicas.

Assim como em outras situações, quando uma organização não tem ainda dados suficientes para definir seus próprios fatores de produtividade, modelos empíricos podem ser usados. Existem diversos modelos que derivam estimativas de esforço a partir de dados de LOC ou PFs.

Por exemplo, o modelo proposto por Bailey-Basili [3] estabelece a seguinte fórmula para se obter o esforço necessário em pessoas-mês para desenvolver um projeto, tomando por base o tamanho do mesmo em KLOC:

$$E = 5{,}5 + 0{,}73 * (KLOC)^{1{,}16}$$

Outros modelos são apresentados em [3] e [2]. Contudo, deve-se observar que modelos empíricos diferentes conduzem a resultados muito diferentes, o que indica que esses modelos devem ser adaptados para as condições da organização. Uma forma de se fazer essa adaptação consiste em experimentar o modelo usando resultados de projetos já finalizados, comparar os valores obtidos com os dados reais e analisar a eficácia do modelo. Se a concordância dos resultados não for boa, as constantes do modelo devem ser recalculadas usando dados organizacionais [1].

3.4.4 - Alocação de Recursos

Referências: [1] Cap. 5, seção 5.4; [2] Cap. 22, seção 22.3;

Estimar os recursos necessários para realizar o esforço de desenvolvimento é outra importante tarefa. Quando falamos em recursos, estamos englobando pessoas, hardware e software. No caso de software, devemos pensar em ferramentas de software, tais como ferramentas CASE ou software de infra-estrutura (p.ex., um sistema operacional), bem como componentes de software a serem reutilizados no desenvolvimento, tais como bibliotecas de interface ou uma camada de persistência de dados.

Em todos os casos (recursos humanos, de hardware e de software), é necessário observar a disponibilidade do recurso. Assim, é importante definir a partir de que data o recurso será necessário, por quanto tempo ele será necessário e qual a quantidade de horas necessárias por dia nesse período, o que, para recursos humanos, convencionamos denominar dedicação. Observe que já entramos na estimativa de duração. Assim, alocação de recursos e estimativa de duração são atividades realizadas normalmente em paralelo.

No que se refere a recursos humanos, outros fatores têm de ser levados em conta. A competência para realizar a atividade para a qual está sendo alocado é fundamental. Assim, é preciso analisar com cuidado as competências dos membros da equipe para poder realizar a alocação de recursos. Outros fatores, como liderança, relacionamento inter-pessoal etc, importantes para a formação de equipes, são igualmente relevantes para a alocação de recursos humanos a atividades.

3.4.5 - Estimativa de Duração e Elaboração de Cronograma

Referências: [1] Cap. 7; [2] Cap. 4, seção 4.3; [3] Cap. 3, seção 3.1.

De posse das estimativas de esforço e realizando em paralelo a alocação de recursos, é possível estimar o tempo cronológico (duração) de cada atividade e, por conseguinte, do projeto como um todo. Se a estimativa de esforço tiver sido realizada para o projeto como um

todo, então ela deverá ser distribuída pelas atividades do projeto. Novamente, dados históricos de projetos já concluídos na organização são uma boa base para se fazer essa distribuição.

No entanto, muitas vezes, uma organização não tem ainda esses dados disponíveis. Embora as características do projeto sejam determinantes para a distribuição do esforço, uma diretriz inicial útil consiste em considerar a distribuição mostrada na Tabela 3.3 [1].

Tabela 3.3 – Distribuição de Esforço pelas Principais Atividades do Processo de Software.

Planejamento	Especificação e Análise de Requisitos	Projeto	Implementação	Teste e Entrega
Até 3%	De 10 a 25%	De 20 a 25%	De 15 a 20%	De 30 a 40%

De posse da distribuição de esforço por atividade e realizando paralelamente a alocação de recursos, pode-se criar uma rede de tarefas com o esforço associado a cada uma das atividades [3]. A partir dessa rede, pode-se estabelecer qual é o caminho crítico do projeto, isto é, qual o conjunto de atividades que determina a duração do projeto. Um atraso em uma dessas atividades provocará atraso no projeto como um todo.

Finalmente, a partir da rede de tarefas, deve-se elaborar um Gráfico de Tempo (ou Gráfico de Gantt), estabelecendo o cronograma do projeto. Gráficos de Tempo podem ser elaborados para o projeto como um todo (cronograma do projeto), para um conjunto de atividades, para um módulo específico ou mesmo para um membro da equipe do projeto.

3.4.6 - Estimativa de Custo

Referências: [1] Cap. 5, seções 5.6 e 5.7; [2] Cap. 23.

De posse das demais estimativas, é possível estimar os custos do projeto. De maneira geral, os seguintes itens devem ser considerados nas estimativas de custos:

- Custos relativos ao esforço empregado pelos membros da equipe no projeto;
- Custos de hardware e software (incluindo manutenção);
- Outros custos relacionados ao projeto, tais como custos de viagens e treinamentos realizados no âmbito do projeto;
- Despesas gerais, incluindo gastos com água, luz, telefone, pessoal de apoio administrativo, pessoal de suporte etc.

Para a maioria dos projetos, o custo dominante é o que se refere ao esforço empregado, juntamente com as despesas gerais. Sommerville [2] sugere que, de modo geral, os custos relacionados com as despesas gerais correspondem a um valor equivalente aos custos relativos ao esforço empregado pelos membros da equipe no projeto. Assim, para efeitos de estimativas de custos, pode-se considerar esses dois itens como sendo um único, computado em dobro.

Custos de hardware e software, ainda que menos influentes, não devem ser desconsiderados, sob pena de provocarem prejuízos para o projeto. Uma forma de tratar esses custos é considerar a depreciação com base na vida útil do equipamento ou da versão do software utilizada.

Quando o custo do projeto estiver sendo calculado como parte de uma proposta para o cliente, então será preciso definir o preço cotado. Uma abordagem para definição do preço pode ser considerá-lo como o custo total do projeto mais o lucro. Entretanto, a relação entre o custo do projeto e o preço cotado para o cliente, normalmente, não é tão simples assim [2].

3.5 - Gerência de Riscos

Referências: [1] Cap. 6; [2] Cap. 4, seção 4.4; [3] Cap. 3, seção 3.4.

Uma importante tarefa da gerência de projetos é prever os riscos que podem prejudicar o bom andamento do projeto e definir ações a serem tomadas para evitar sua ocorrência ou, quando não for possível evitar a ocorrência, para diminuir seus impactos.

Um risco é qualquer condição, evento ou problema cuja ocorrência não é certa, mas que pode afetar negativamente o projeto, caso ocorra. Assim, os riscos envolvem duas características principais:

- *incerteza* – um risco pode ou não acontecer, isto é, não existe nenhum risco 100% provável;
- *perda* – se o risco se tornar realidade, conseqüências não desejadas ou perdas acontecerão.

Desta forma, é de suma importância que riscos sejam identificados durante um projeto de software, para que ações possam ser planejadas e utilizadas para evitar que um risco se torne real, ou para minimizar seus impactos, caso ele ocorra. Esse é o objetivo da Gerência de Riscos, cujo processo envolve as seguintes atividades:

- Identificação de riscos: visa identificar possíveis ameaças (riscos) para o projeto, antecipando o que pode dar errado;
- Análise de riscos: trata de analisar os riscos identificados, estimando sua probabilidade e impacto (grau de exposição ao risco);
- Avaliação de riscos: busca priorizar os riscos e estabelecer um ponto de corte, indicando quais riscos serão gerenciados e quais não serão;
- Planejamento de ações: trata do planejamento das ações a serem tomadas para evitar (ações de mitigação) que um risco ocorra ou para definir o que fazer quando um risco se tornar realidade (ações de contingência);
- Elaboração do Plano de Riscos: todos os aspectos envolvidos na gerência de riscos devem ser documentados em um plano de riscos, indicando os riscos que compõem o perfil de riscos do projeto, as avaliações dos riscos, a definição dos riscos a serem gerenciados e, para esses, as ações para evitá-los ou para minimizar seus impactos, caso venham a ocorrer.
- Monitoramento de riscos: à medida que o projeto progride, os riscos têm de ser monitorados para se verificar se os mesmos estão se tornando realidade ou não.

Novos riscos podem ser identificados, o grau de exposição de um risco pode mudar e ações podem ter de ser tomadas. Essa atividade é realizada durante o acompanhamento do progresso do projeto.

Na identificação de riscos, trabalhar com uma série de riscos aleatórios pode ser um fator complicador, principalmente em grandes projetos, em que o número de riscos é relativamente grande. Assim, a classificação de riscos em categorias de risco, definindo tipos básicos de riscos, é importante para guiar a realização dessa atividade. Cada organização deve ter seu próprio conjunto de categorias de riscos. Para efeito de exemplo, podem ser consideradas categorias tais como: tecnologia, pessoal, legal, organizacional, de negócio etc.

Uma vez identificados os riscos, deve ser feita uma análise dos mesmos à luz de suas duas principais variáveis: a probabilidade do risco se tornar real e o impacto do mesmo, caso ocorra. Na análise de riscos, o gerente de projeto deve executar quatro atividades básicas [1]: (i) estabelecer uma escala que reflita a probabilidade de um risco, (ii) avaliar as conseqüências dos riscos, (iii) estimar o impacto do risco no projeto e no produto e (iv) calcular o grau de exposição do risco, que é uma medida casando probabilidade e impacto.

De maneira geral, escalas para probabilidades e impactos são definidas de forma qualitativa, tais como: probabilidade - alta, média ou baixa, e impacto - baixo, médio, alto ou muito alto. Isso facilita a análise dos riscos, mas, por outro lado, pode dificultar a avaliação. Assim, a definição de medidas quantitativas para o risco pode ser importante, pois tende a diminuir a subjetividade na avaliação. Jalote [8] propõe os valores quantitativos mostrados nas tabelas 3.4 e 3.5 para as escalas acima.

Tabela 3.4 - Categorias de Probabilidade [8]

Probabilidade	Faixa de Valores
Baixa	até 30%
Média	30 a 70%
Alta	acima de 70%

Tabela 3.5 - Categorias de Impacto [8]

Impacto	Faixa de Valores
Baixo	de 0 a 3
Médio	de 3 a 7
Alto	de 7 a 9
Muito Alto	de 9 a 10

Usando valores quantitativos para expressar probabilidade e impacto, é possível obter um valor numérico para o grau de exposição ao risco, dado por: $E(r) = P(r) * I(r)$, onde $E(r)$ é o grau de exposição associado ao risco r e $P(r)$ e $I(r)$ correspondem, respectivamente, aos valores numéricos de probabilidade e impacto do risco r.

De posse do grau de exposição de cada um dos riscos que compõem o perfil de riscos do projeto, pode-se passar à avaliação dos mesmos. Uma tabela ordenada pelo grau de exposição pode ser montada e uma linha de corte nessa tabela estabelecida, indicando quais riscos serão tratados e quais serão desprezados.

Para os riscos a serem gerenciados, devem ser definidos planos de ação, estabelecendo ações a serem adotadas para, em primeiro lugar, evitar que os riscos ocorram (ações de mitigação) ou, caso isso não seja possível, ações para tratar e minimizar seus impactos (ações de contingência). Esse é o propósito da atividade de planejamento das ações.

Ao longo de todo o processo da gerência de riscos, as decisões envolvidas devem ser documentadas no plano de riscos. Esse plano servirá de base para a atividade de monitoramento dos riscos, quando os riscos têm de ser monitorados para se verificar se os mesmos estão se tornando realidade ou não. Caso estejam se tornando (ou já sejam) uma realidade, deve-se informar que ações foram tomadas. No caso do risco se concretizar, deve-se informar, também, quais as conseqüências da sua ocorrência.

3.6 - Elaboração do Plano de Projeto

Referências: [1] Cap. 7, seção 7.10; [2] Cap. 4, seção 4.2; [3] Cap. 3, seção 3.5.

Todas as atividades realizadas no contexto da gerência de projeto devem ser documentadas em um Plano de Projeto. Cada organização deve estabelecer um modelo ou padrão para a elaboração desse documento, de modo que todos os projetos da organização contenham as informações consideradas relevantes.

Referências

1. R.S. Pressman, *Engenharia de Software*, Rio de Janeiro: McGraw Hill, 5ª edição, 2002.

   O Capítulo 3 dá uma visão geral da gerência de projetos de software, com discussões sobre pessoal e formação de equipes (seção 3.2), definição do escopo do software (seção 3.3), estrutura de divisão do trabalho (seção 3.4). O Capítulo 4 aborda o tema de medição e métricas nas seções 4.1 e 4.3. O Capítulo 5 trata com mais detalhes do planejamento do projeto, com destaque para a determinação do escopo (seção 5.3), estimativas (seções 5.1, 5.5, 5.6 e 5.7) e recursos (seção 5.4). O Capítulo 6 trata da gerência de riscos. Finalmente, o Capítulo 7 aborda a elaboração e o acompanhamento do cronograma.

2. I. Sommerville, *Engenharia de Software*, São Paulo: Addison-Wesley, 6ª edição, 2003.

   O Capítulo 4 proporciona uma visão geral da gerência de projetos de software, abordando atividades da gerência (seção 4.1), planejamento do projeto (seção 4.2), elaboração do cronograma (seção 4.3) e gerência de riscos (seção 4.4).Além disso, tem dois capítulos (22 e 23) dedicados, respectivamente, à gerência de pessoal e estimativas.

3. S.L. Pfleeger, *Engenharia de Software: Teoria e Prática*, São Paulo: Prentice Hall, 2ª edição, 2004.

O Capítulo 3 é dedicado ao planejamento e à gerência de projetos de software. A seção 3.1 trata do escopo, estrutura de divisão do trabalho e elaboração do cronograma. Na seção 3.2, há uma discussão a cerca do pessoal necessário para o projeto e formação de equipes. As seções 3.3, 3.4 e 3.5 discutem, respectivamente, os temas estimativas, gerência de riscos e plano de projeto.

4. M.F. Vieira, *Gerenciamento de Projetos de Tecnologia da Informação*, Rio de Janeiro : Editora Elsevier – Campus, 2003.
5. J.C.C. Martins, *Gerenciando Projetos de Desenvolvimento de Software com PMI, RUP e UML*, 2ª edição revisada, Rio de Janeiro : Brasport, 2005.
6. C.E. Vazquez, G.S. Simões, R.M. Albert, *Análise de Pontos de Função: Medição, Estimativas e Gerenciamento de Projetos de Software*, 3ª edição, São Paulo : Editora Érica, 2005.

   Este livro traz uma apresentação bastante detalhada do método da Análise de Pontos de Função, explorando a importância da medição (Capítulo 1), uma visão geral do processo de contagem (Capítulo 3), detalhamento desse processo (capítulos 4, 5, 6 e 7) e estimativas (Capítulo 9).
7. C. Hazan. "Medição da Qualidade e Produtividade em Software", In: Qualidade e Produtividade em Software, 4ª edição, K.C. Weber, A.R.C. Rocha, C.J. Nascimento (organizadores), Makron Books, 2001, p. 25 – 41.
8. P. Jalote, *CMM in Practice: Processes For Executing Software Projects At Infosys*, Addison-Wesley Publishing Company, 1999.

Capítulo 4 – Gerência da Qualidade de Software

Uma das principais metas da Engenharia de Software é a produção de software de qualidade. Entretanto, qualidade não é algo que se atinge de forma espontânea. Qualidade tem de ser construída em um produto de software e diversos aspectos são importantes neste contexto. Neste capítulo, discutimos alguns desses aspectos, aqueles que consideramos mais importantes, a saber: documentação, controle e garantia da qualidade e gerência de configuração de software.

4.1 – Documentação

A documentação produzida em um projeto de software é de suma importância para se gerenciar a qualidade, tanto do produto sendo produzido, quanto do processo usado para seu desenvolvimento.

No desenvolvimento de software, são produzidos diversos documentos, dentre eles, documentos descrevendo processos (plano de projeto, plano de qualidade etc), registrando requisitos e modelos do sistema (documentos de especificação de requisitos, análise e projeto) e apoiando o uso do sistema gerado (manual do usuário, ajuda *on-line*, tutoriais etc).

Uma documentação de qualidade propicia uma maior organização durante o desenvolvimento de um sistema, facilitando modificações e futuras manutenções no mesmo. Além disso, reduz o impacto da perda de membros da equipe, reduz o tempo de desenvolvimento de fases posteriores, reduz o tempo de manutenção e contribui para redução de erros, aumentando assim, a qualidade do processo e do produto gerado. Dessa forma, a criação da documentação é tão importante quanto a criação do software em si [4].

Há, portanto, a necessidade de se definir um processo para controlar a documentação de uma organização, dito processo de documentação, incluindo atividades de planejamento, análise, aprovação ou reprovação, identificação de alterações, situação da revisão atual, disponibilidade das versões pertinentes de documentos aplicáveis, dentre outras. Algumas dessas atividades estão relacionadas com o controle e a garantia da qualidade de software, outras com a gerência da configuração do software, conforme discutido a seguir.

É importante notar que o planejamento da documentação tem uma estreita relação com o processo de software definido para o projeto. Ou seja, os documentos a serem gerenciados são aqueles previstos como saídas das atividades do processo. Assim, tendo sido definido o processo do projeto, o planejamento da sua documentação consiste apenas em selecionar quais artefatos, dentre os muitos produzidos ao longo do processo, serão efetivamente submetidos à gerência de configuração de software e ao controle e garantia da qualidade.

4.2 – Controle e Garantia da Qualidade

Referências: [1] Cap. 8, seções 8.3, 8.4 e 8.11; [2] Cap. 24, seções 24.1, 24.2, 24.3;

Durante o processo de desenvolvimento de software, ocorrem enganos, interpretações errôneas e outras faltas (defeitos ou erros), principalmente provocados por problemas na comunicação e transformação da informação, que podem resultar em mau funcionamento do sistema produzido. Assim, é muito importante detectar esses defeitos o quanto antes, preferencialmente na atividade em que foram cometidos, como forma de diminuir retrabalho e, por conseguinte, custos de alterações. As atividades que se preocupam com essa questão são coletivamente denominadas atividades de garantia da qualidade de software e devem ser realizadas ao longo de todo o processo de desenvolvimento de software [5].

Dentre as atividades de controle e garantia da qualidade estão as atividades de Verificação, Validação e Testes (VV&T). O objetivo da verificação é assegurar que o software esteja sendo construído de forma correta. Deve-se verificar se os artefatos produzidos atendem aos requisitos estabelecidos e se os padrões organizacionais (de produto e processo) foram consistentemente aplicados. Por outro lado, o objetivo da validação é assegurar que o software que está sendo desenvolvido é o software correto, ou seja, se os requisitos e o software dele derivado atendem ao uso específico proposto. Por fim, os testes de software são atividades de validação e verificação que consistem da análise dinâmica do mesmo, isto é, envolvem a execução do produto de software.

Uma vez que as atividades de VV&T são tão importantes, elas devem ser cuidadosamente planejadas, dando origem a um Plano de Garantia da Qualidade.

Os artefatos que compõem a documentação do projeto são as entradas (insumos) para as atividades de garantia da qualidade, quando os mesmos são verificados quanto à aderência em relação aos padrões de documentação da organização e validados em relação aos seus propósitos e aos requisitos que se propõem a atender. Assim, uma questão imprescindível para a gerência da qualidade é a definição de padrões organizacionais.

4.2.1 - Padrões Organizacionais

Uma vez que a gerência da qualidade envolve tanto a qualidade do processo quanto a qualidade do produto, devem ser estabelecidos padrões organizacionais de produto e de processo. Os padrões de processo são os ditos processos padrão ou processos padrão especializados, discutidos no Capítulo 2.

Os padrões de produto, por sua vez, são padrões que se aplicam a artefatos produzidos ao longo do processo de software. Podem ser, dentre outros, modelos de documentos, roteiros e normas, dependendo do artefato a que se aplicam. Tipicamente, para documentos, modelos de documentos e roteiros são providos.

Um modelo de documento define a estrutura (seções, subseções, informações de cabeçalho e rodapé de página etc), o estilo (tamanho e tipos de fonte, cores etc) e o conteúdo esperado para documentos de um tipo específico. Documentos tais como Plano de Projeto, Especificação de Requisitos e Especificação de Projeto devem ter modelos de documentos específicos associados. Documentos padronizados são importantes, pois facilitam a leitura e a compreensão, uma vez que os profissionais envolvidos estão familiarizados com seu formato.

Quando não é possível ou desejável estabelecer uma estrutura rígida como um modelo de documento, roteiros dando diretrizes gerais para a elaboração de um artefato devem ser providos. Em situações em que não é possível definir uma estrutura, tal como no código-fonte, normas devem ser providas. Assim, tomando o exemplo do código-fonte, é extremamente pertinente a definição de um padrão de codificação, indicando nomes de variáveis válidos, estilos de identação, regras para comentários etc.

Padrões organizacionais, sejam de processo sejam de produto, são muito importantes, pois eles fornecem um meio de capturar as melhores práticas de uma organização e facilitam a realização de atividades de avaliação da qualidade, que podem ser dirigidas pela avaliação da conformidade em relação ao padrão. Além disso, sendo organizacionais, todos os membros da organização tendem a estar familiarizados com os mesmos, facilitando a manutenção dos artefatos ou a substituição de pessoas no decorrer do projeto. Para aqueles artefatos tidos como os mais importantes no planejamento da documentação, é saudável que haja um padrão organizacional associado.

Dada a importância de padrões organizacionais, eles devem ser elaborados com bastante cuidado. Uma boa prática, conforme já sugerido para a definição de processos padrão, consiste em usar como base padrões gerais propostos por instituições nacionais ou internacionais voltadas para a área de qualidade de software, tal como a ISO.

4.2.2 - Revisões

Para se controlar a qualidade em um projeto de software, uma abordagem bastante usada consiste em se realizar revisões. Nas revisões, processos, documentos e outros artefatos são revisados por um grupo de pessoas, com o objetivo de verificar se os mesmos estão em conformidade com os padrões organizacionais estabelecidos e se o propósito de cada um deles está sendo atingido, incluindo o atendimento a requisitos do cliente e dos usuários. Assim, o objetivo de uma revisão é detectar erros e inconsistências em artefatos e processos, sejam eles relacionados à forma, sejam em relação ao conteúdo, e apontá-los aos responsáveis pela sua elaboração.

O processo de revisão começa com o planejamento da revisão, quando uma equipe de revisão é formada, tendo à frente um líder. A equipe de revisão deve incluir os membros da equipe que possam efetivamente úteis para atingir o objetivo da revisão. Muitas vezes, a pessoa responsável pela elaboração do artefato a ser revisado integra a equipe de revisão.

Dando início ao processo de revisão propriamente dito, normalmente, o autor do artefato apresenta o mesmo e descreve a perspectiva utilizada para a sua construção. Além disso, o propósito da revisão deve ser previamente informado e o material a ser revisado deve ser entregue com antecedência para que cada membro da equipe de revisão possa avaliá-lo. Uma vez que todos estejam preparados, uma reunião é convocada pelo líder. Essa reunião deverá ser relativamente breve (duas horas, no máximo), uma vez que todos já estão preparados para a mesma. Durante a reunião, o líder orientará o processo de revisão, passando por todos os aspectos relevantes a serem revistos. Todas as considerações dos demais membros da equipe de revisão devem ser discutidas e as decisões registradas, dando origem a uma ata de reunião de revisão, contendo uma lista de defeitos encontrados.

4.3 – Gerência de Configuração de Software

Referências: [1] Cap. 9; [2] Cap. 29; [3] Cap. 3, seção 3.3.

Durante o processo de desenvolvimento de software, vários artefatos são produzidos e alterados constantemente, evoluindo até que seus propósitos fundamentais sejam atendidos. Ferramentas de software, tais como compiladores e editores de texto, e processos também podem ser substituídos por versões mais recentes ou mesmo por outras, no caso de ferramentas. Porém, caso essas mudanças não sejam devidamente documentadas e comunicadas, poderão acarretar diversos problemas, tais como: dois ou mais desenvolvedores podem estar alterando um mesmo artefato ao mesmo tempo; não se saber qual a versão mais atual de um artefato; não se refletir alterações nos artefatos impactados por um artefato em alteração. Esses problemas podem gerar vários transtornos como incompatibilidade entre os grupos de desenvolvimento, inconsistências, retrabalho, atraso na entrega e insatisfação do cliente.

Assim, para que esses transtornos sejam evitados, é de suma importância o acompanhamento e o controle de artefatos, processos e ferramentas, através de um processo de gerência de configuração de software, durante todo o ciclo de vida do software [5].

A Gerência de Configuração de Software (GCS) visa estabelecer e manter a integridade dos itens de software ao longo de todo o ciclo de vida do software, garantindo a completeza, a consistência e a correção de tais itens, e controlando o armazenamento, a manipulação e a distribuição dos mesmos. Para tal, tem de identificar e documentar os produtos de trabalho que podem ser modificados, estabelecer as relações entre eles e os mecanismos para administrar suas diferentes versões, controlar modificações e permitir auditoria e a elaboração de relatórios sobre o estado de configuração.

Pelos objetivos da GCS, pode-se notar que ela está diretamente relacionada com as atividades de garantia da qualidade de software.

As atividades da GCS podem ser resumidas em:

- Planejamento da GCS: Um plano deve ser elaborado descrevendo as atividades da gerência de configuração, procedimentos e responsáveis pela execução dessas atividades.
- Identificação da Configuração: refere-se à identificação dos itens de software e suas versões a serem controladas, estabelecendo linhas básicas.
- Controle de Versão: combina procedimentos e ferramentas para administrar diferentes versões dos itens de configuração criados durante o processo de software.
- Controle de Modificação: combina procedimentos humanos e ferramentas automatizadas para controlar as alterações feitas em itens de software. Para tal, o seguinte processo é normalmente realizado: solicitação de mudança, aprovação ou rejeição da solicitação, registro de retirada para alteração (*check-out*), análise, avaliação e realização das alterações, revisão e registro da realização das alterações (*check-in*).

- Auditoria de Configuração: visa avaliar um item de configuração quanto a características não consideradas nas revisões, tal como se os itens relacionados aos solicitados foram devidamente atualizados.
- Relato da situação da configuração: refere-se à preparação de relatórios que mostrem a situação e o histórico dos itens de software controlados. Tais relatórios podem incluir, dentre outros, o número de alterações nos itens, as últimas versões dos mesmos e identificadores de liberação.

4.3.1 - O Processo de GCS

O primeiro passo do processo de gerência de configuração de software é a confecção de um plano de gerência de configuração, que inicia com a identificação dos itens que serão colocados sob gerência de configuração, chamados itens de configuração. Os itens mais relevantes para serem submetidos à gerência de configuração são aqueles mais usados durante o ciclo de vida, os mais genéricos, os mais importantes para segurança, os projetados para reutilização e os que podem ser modificados por vários desenvolvedores ao mesmo tempo [6]. Os itens não colocados sob gerência de configuração podem ser alterados livremente.

Após a seleção dos itens, deve-se descrever como eles se relacionam. Isso será muito importante para as futuras manutenções, pois permite identificar de maneira eficaz os itens afetados em decorrência de uma alteração. Além disso, deve-se criar um esquema de identificação dos itens de configuração, com atribuição de nomes exclusivos, para que seja possível estabelecer a evolução de cada versão dos itens [1, 6].

Após a identificação dos itens de configuração, devem ser planejadas as linhas-base dentro do ciclo de vida do projeto. Uma linha base (ou *baseline*) é uma versão estável de um sistema contendo todos os componentes que constituem este sistema em um determinado momento. Nos pontos estabelecidos pelas linhas-base, os itens de configuração devem ser identificados, analisados, corrigidos, aprovados e armazenados em um local sob controle de acesso, denominado repositório central, base de dados de projeto ou biblioteca de projeto. Assim, quaisquer alterações nos itens daí em diante só poderão ser realizadas através de procedimentos formais de controle de modificação [1, 6].

O passo seguinte do processo de GCS é o controle de versão, que combina procedimentos e ferramentas para identificar, armazenar e administrar diferentes versões dos itens de configuração que são criados durante o processo de software [1, 6]. A ideia é que a cada modificação que ocorra em um item de configuração, uma nova versão ou variante seja criada. Versões de um item são geradas pelas diversas alterações, enquanto variantes são as diferentes formas de um item, que existem simultaneamente, e atendem a requisitos similares [6].

Talvez a mais importante atividade do processo de GCS é o controle de alterações, que combina procedimentos humanos e ferramentas automatizadas para o controle das alterações realizadas nos itens de configuração [1]. Assim que uma alteração é solicitada, o impacto em outros itens de configuração e o custo para a modificação devem ser avaliados. Um responsável deve decidir se a alteração poderá ou não ser realizada. Caso a alteração seja liberada, pessoas são indicadas para sua execução. Assim que não houver ninguém utilizando os itens de configuração envolvidos, cópias deles são retiradas do repositório central e colocadas em uma área de trabalho do desenvolvedor, através de um procedimento

denominado *check-out*. A partir deste momento, nenhum outro desenvolvedor poderá alterar esses itens. Os desenvolvedores designados fazem as alterações necessárias e, assim que essas forem concluídas, os itens são submetidos a uma revisão. Se as alterações forem aprovadas, os itens são devolvidos ao repositório central, estabelecendo uma nova linha base, em um procedimento chamado *check-in* [1, 6].

Porém, mesmo com os mecanismos de controle mais bem sucedidos, não é possível garantir que as modificações foram corretamente implementadas. Assim, revisões e auditorias de configuração de software são necessárias no processo de GCS [1]. Essas atividades de garantia da qualidade tentam descobrir omissões ou erros na configuração e se os procedimentos, padrões, regulamentações ou guias foram devidamente aplicados no processo e no produto [6].

Enfim, o último passo do processo de GCS é a preparação de relatórios, que é uma tarefa que tem como objetivo relatar a todas as pessoas envolvidas no desenvolvimento e manutenção do software as seguintes questões: (i) O que aconteceu? (ii) Quem fez? (iii) Quando aconteceu? (iv) O que mais será afetado? O acesso rápido às informações agiliza o processo de desenvolvimento e melhora a comunicação entre as pessoas, evitando, assim, muitos problemas de alterações do mesmo item de configuração, com intenções diferentes e, às vezes, conflitantes [6].

4.4 – Medição e Métricas

Referências: [1] Cap. 4, seções 4.1, 4.2 e 4.3; [2] Cap. 24, seção 24.4;

Para poder controlar a qualidade, medir é muito importante. Se não é possível medir, expressando em números, apenas uma análise qualitativa (e, portanto, subjetiva) pode ser feita, o que, na maioria das vezes, é insuficiente. Com medições, as tendências (boas ou más) podem ser detectadas, melhores estimativas podem ser feitas e melhorias reais podem ser conseguidas [1].

Quando se trata de avaliação quantitativa, quatro conceitos, muitas vezes usados equivocadamente com o mesmo sentido, são importantes: medida, medição, métrica e indicador. Ainda não há consenso na comunidade de Engenharia de Software sobre a utilização desses termos, o que leva à utilização heterogênea dos conceitos. Segundo Pressman [1], uma **medida** fornece uma indicação quantitativa da extensão, quantidade, dimensão, capacidade ou tamanho de um atributo de um produto ou de um processo. Quando os dados de um único ponto são coletados, uma medida é estabelecida (p.ex., quantidade de erros descobertos em uma revisão). **Medição** é o ato de medir, isto é, de determinar uma medida. Já uma **métrica** procura relacionar medidas individuais com o objetivo de se ter uma ideia da eficácia do processo, do projeto ou do produto sendo medido. Por fim, desenvolve-se métricas para se obter **indicadores**, isto é, para se ter uma compreensão do processo, produto ou projeto sendo medido.

Seja o seguinte exemplo: deseja-se saber se uma pessoa está com seu peso ideal ou não. Para tal, duas **medidas** são importantes: altura (H) e peso (P). Ao medir essas dimensões, está-se efetuando uma **medição**. A **métrica** "índice de massa corporal (IMC)" é calculada segundo a seguinte fórmula: $IMC = P / H^2$. A partir dessa métrica, a Organização Mundial de Saúde estabeleceu **indicadores** que apontam se um adulto está acima do peso, se está obeso ou abaixo do peso ideal considerado saudável, conforme a seguir:

Se IMC < 18,5, então o indivíduo está abaixo do peso ideal considerado saudável;

Se 18,5 <= IMC < 25, então o indivíduo está com o peso normal;

Se 25 <= IMC <= 30, então o indivíduo está acima do peso;

Se IMC > 30, então o indivíduo está obeso.

Realizando medições, uma pessoa pode obter suas medidas para altura e peso. A partir delas, pode calcular a métrica IMC e ter um indicador se está abaixo do peso, no peso ideal, acima do peso ou obeso. Conhecendo esse indicador, a pessoa pode ajustar seu processo de alimentação, obtendo melhorias reais para sua saúde.

Uma vez que a medição de software se preocupa em obter valores numéricos (medidas) para alguns atributos de um produto ou de um processo, uma questão importante passa a ser: Que atributos medir?

O modelo de qualidade definido na norma ISO/IEC 9126-1 trata dessa questão, estando subdividido em dois modelos: o modelo de qualidade para características externas e internas e o modelo de qualidade para qualidade em uso. O modelo de qualidade para características externas e internas classifica os atributos de qualidade de software em seis características (funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade) que são, por sua vez, desdobradas em subcaracterísticas. As subcaracterísticas podem ser desdobradas em mais níveis, até se ter subcaracterísticas diretamente mensuráveis, para as quais métricas são aplicadas. As normas ISO/IEC 9126-2 e 9126-3 apresentam, respectivamente, métricas externas e internas.

Esse modelo de qualidade que preconiza a análise de características de qualidade a partir de suas subcaracterísticas de forma recursiva até que se tenham métricas para coletar dados é aplicável não somente a produto, mas também para avaliar a qualidade de processos de software. Assim, de maneira geral, na avaliação quantitativa da qualidade, é necessário:

1. Definir características de qualidade relevantes para avaliar a qualidade do objeto em questão (produto ou processo).
2. Para cada característica de qualidade selecionada, definir subcaracterísticas de qualidade relevantes que tenham influência sobre a mesma, estabelecendo um modelo ou fórmula de computar a característica a partir das subcaracterísticas. Fórmulas baseadas em peso são bastante utilizadas: $cq = p_1 * scq_1 + ... + p_n * scq_n$.
3. Usar procedimento análogo ao anterior para as subcaracterísticas não passíveis de mensuração direta.
4. Para as subcaracterísticas diretamente mensuráveis, selecionar métricas, coletar medidas e computar as métricas segundo a fórmula ou modelo estabelecido.
5. Fazer o caminho inverso, agora computando subcaracterísticas não diretamente mensuráveis, até se chegar a um valor para a característica de qualidade.

Concluído o processo de medição, deve-se comparar os valores obtidos com padrões estabelecidos para a organização, de modo a se obter os indicadores da qualidade. A partir dos indicadores, ações devem ser tomadas visando à melhoria.

Vale destacar que, especialmente no caso da avaliação da qualidade de software, métricas relacionadas a defeitos são bastante úteis, tal como (número de erros / ponto de função).

O único modo racional de melhorar um processo é medir atributos específicos, obter um conjunto de métricas significativas baseadas nesses atributos e usar os valores das métricas para fornecer indicadores que conduzirão um processo de melhoria [1].

4.5 – Considerações Finais

Neste capítulo foram discutidos alguns aspectos que julgamos fundamentais para o desenvolvimento de software com qualidade. Vale destacar que, embora apresentados coletivamente como parte da gerência da garantia da qualidade, eles têm uma importância tão destacada que as principais normas e modelos de qualidade (vide capítulo 2) os tratam como processos individualmente. Assim, no modelo MPS.BR [7], por exemplo, garantia da qualidade, gerência de configuração e medição são processos do nível F.

Referências

1. R.S. Pressman, *Engenharia de Software*, Rio de Janeiro: McGraw Hill, 5ª edição, 2002.
2. I. Sommerville, *Engenharia de Software*, São Paulo: Addison-Wesley, 6ª edição, 2003.
3. S.L. Pfleeger, *Engenharia de Software: Teoria e Prática*, São Paulo: Prentice Hall, 2ª edição, 2004.
4. R. Sanches, "Documentação de Software". In: *Qualidade de Software: Teoria e Prática*, Eds. A.R.C. Rocha, J.C. Maldonado, K. Weber, Prentice Hall, 2001.
5. J.C. Maldonado, S.C.P.F. Fabbri, "Verificação e Validação de Software". In: *Qualidade de Software: Teoria e Prática*, Eds. A.R.C. Rocha, J.C. Maldonado, K. Weber, Prentice Hall, 2001.
6. R. Sanches, "Gerência de Configuração de Software". In: *Qualidade de Software: Teoria e Prática*, Eds. A.R.C. Rocha, J.C. Maldonado, K. Weber, Prentice Hall, 2001.
7. SOFTEX, MPS.BR – Melhoria de Processo do Software Brasileiro – Guia Geral, 2009.

Anexo A – Análise de Pontos de Função

A Análise de Pontos de Função (APF) é um método padrão para a medição do desenvolvimento de software, visando estabelecer uma medida de tamanho do software em Pontos de Função (PFs), com base na funcionalidade a ser implementada, sob o ponto de vista do usuário.

Os objetivos da APF são:

- Medir as funcionalidades do sistema requisitadas e recebidas pelo usuário;
- Medir projetos de desenvolvimento e manutenção de software, sem se preocupar com a tecnologia que será utilizada na implementação.

O processo para contagem de PFs compreende sete passos, mostrados na Figura A.1.

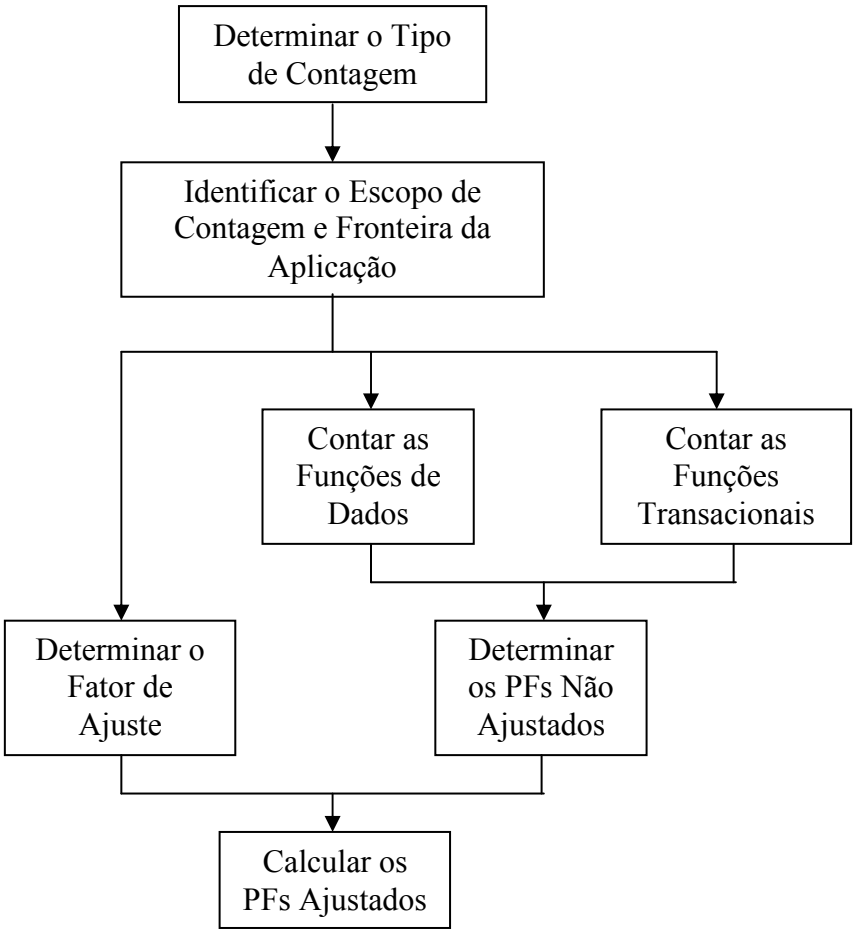

Figura A.1 – O Processo de Contagem de Pontos de Função.

- **Determinar o tipo de contagem de pontos de função**: este é o primeiro passo no processo de contagem, sendo que existem três tipos de contagem: contagem de PF de projeto de desenvolvimento, de aplicações instaladas e de projetos de manutenção.
- **Identificar o escopo de contagem e a fronteira da aplicação**: neste passo, definem-se as funcionalidades que serão incluídas em uma contagem de PFs específica. A fronteira da aplicação é definida estabelecendo um limite lógico entre a aplicação que está sendo medida, o usuário e outras aplicações. O escopo de contagem define a parte do sistema (funcionalidades) a ser contada.
- **Determinar a contagem de pontos de função não ajustados**: os pontos de função não ajustados (PFNA) refletem as funcionalidades fornecidas pelo sistema para o usuário. Essa contagem leva em conta dois tipos de função: de dados e transacionais, bem como sua complexidade (simples, média ou complexa).
- **Contagem das funções de dados**: as funções de dados representam as funcionalidades relativas aos requisitos de dados internos e externos à aplicação. São elas os arquivos lógicos internos e os arquivos de interface externa. Ambos são grupos de dados logicamente relacionados ou informações de controle que foram identificados pelo usuário. A diferença está no fato de um **Arquivo Lógico Interno (ALI)** ser mantido dentro da fronteira da aplicação, isto é, armazenar os dados mantidos através de um ou mais processos elementares da aplicação, enquanto que um **Arquivo de Interface Externa (AIE)** é apenas referenciado pela aplicação, ou seja, ele é mantido dentro da fronteira de outra aplicação. Assim, o objetivo de um AIE é armazenar os dados referenciados por um ou mais processos elementares da aplicação sendo contada, mas que são mantidos por outras aplicações.
- **Contagem das funções transacionais**: as funções transacionais representam as funcionalidades de processamento de dados do sistema fornecidas para o usuário. São elas: as entradas externas, as saídas externas e as consultas externas. As **Entradas Externas (EEs)** são processos elementares que processam dados (ou informações de controle) que entram pela fronteira da aplicação. O objetivo principal de uma EE é manter um ou mais ALIs ou alterar o comportamento do sistema. As **Saídas Externas (SEs)** são processos elementares que enviam dados (ou informações de controle) para fora da fronteira da aplicação. Seu objetivo é mostrar informações recuperadas através de um processamento lógico (isto é, que envolva cálculos ou criação de dados derivados) e não apenas uma simples recuperação de dados. Uma SE pode, também, manter um ALI ou alterar o comportamento do sistema. Por fim, uma **Consulta Externa (CE)**, assim como uma SE, é um processo elementar que envia dados (ou informações de controle) para fora da fronteira da aplicação, mas sem realização de nenhum cálculo nem a criação de dados derivados. Seu objetivo é apresentar informação para o usuário, por meio apenas de uma recuperação das informações. Nenhum ALI é mantido durante sua realização, nem o comportamento do sistema é alterado.
- **Determinar o valor do fator de ajuste**: o fator de ajuste é baseado em 14 características gerais de sistemas, que avaliam a funcionalidade geral da aplicação que está sendo contada, e seus níveis de influência. O nível de influência de uma característica é determinado com base em uma escala de 0 (nenhuma influência) a 5 (forte influência). Assim, o fator de ajuste visa a ajustar os pontos de função não ajustados em ± 35%. Esse passo tornou-se opcional em 2002 para que o método da APF passasse a ser um padrão internacional de medição funcional (ISO/IEC 20926).

As principais críticas são a grande variação na interpretação das 14 características gerais de sistemas e a constatação que algumas delas estão desatualizadas.

- **Calcular os pontos de função ajustados**: finalmente, os PFs ajustados são calculados, considerando-se o tipo de contagem definido no primeiro passo.

A Figura A.2 apresenta uma visão geral dos tipos de função que são considerados na contagem da APF.

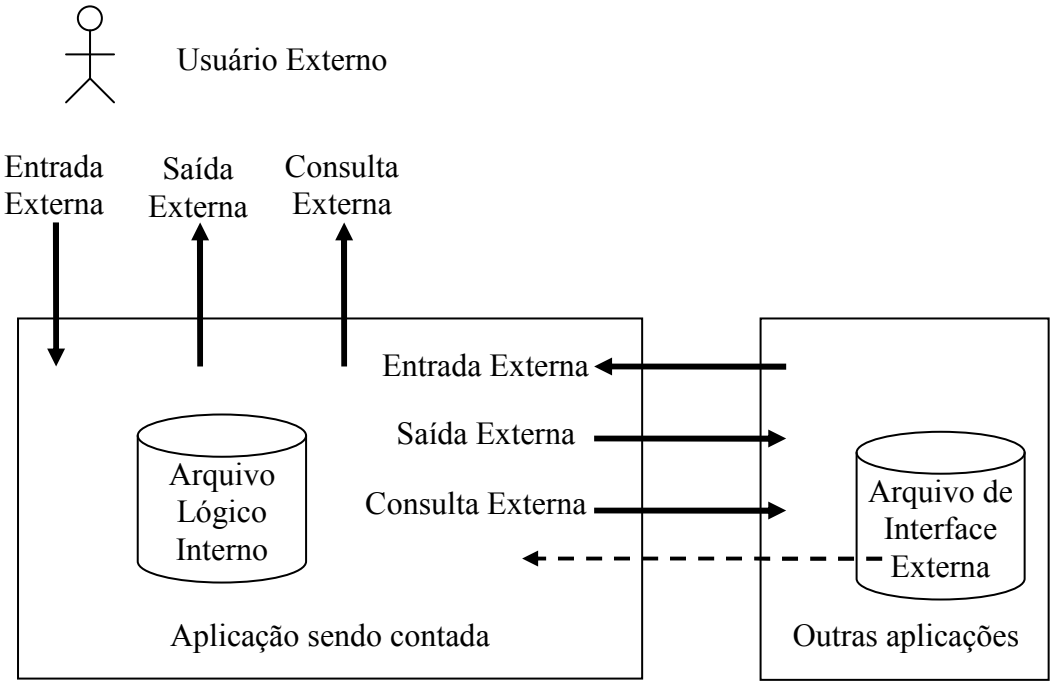

Figura A.2 – Visão Geral das Funções de uma Aplicação segundo a APF.

Contagem das Funções de Dados

Conforme discutido anteriormente, o primeiro passo para a contagem das funções de dados consiste em identificar arquivos lógicos internos (ALIs) e arquivos de interface externa (AIEs). Cada uma dessas funções de dados deve ser classificada segundo sua complexidade funcional. Essa complexidade é definida com base em dois conceitos: registros lógicos e itens de dados.

Registros Lógicos são subconjuntos de dados dentro de um ALI/AIE, que foram reconhecidos pelo usuário. Se o usuário não reconhecer subconjuntos de dados em um ALI/AIE, então se deve contar o ALI/AIE como um registro lógico.

Um Item de Dados, por sua vez, é um campo reconhecido pelo usuário como único e não repetido. Vale destacar que só devem ser contados os itens de dados utilizados pela aplicação em contagem.

Contando-se os registros lógicos e os itens de dados de um ALI/AIE, pode-se chegar à sua complexidade, utilizando a tabela A.1.

Tabela A.1 – Tabela de Identificação da Complexidade das Funções de Dados .

Número de Registros Lógicos	Número de Itens de Dados Referenciados		
	De 1 a 19	**De 20 a 50**	**51 ou mais**
Apenas 1	Simples	Simples	Média
De 2 a 5	Simples	Média	Complexa
6 ou mais	Média	Complexa	Complexa

Contagem das Funções Transacionais

De maneira análoga à contagem das funções de dados, a contagem das funções transacionais envolve a identificação de funções transacionais (entradas externas, saídas externas e consultas externas) e sua classificação de acordo com a complexidade funcional envolvida (simples, média ou complexa). A definição da complexidade funcional é feita com base no número de arquivos referenciados e dos itens de dados manipulados pela função, utilizando as tabelas A.2 para entradas externas e A.3 para saídas e consultas externas. Nessas tabelas, um arquivo referenciado pode ser um ALI lido ou mantido pela função transacional, ou um AIE lido pela função transacional. Já o número de itens de dados referenciados é calculado considerando apenas os itens de dados efetivamente referenciados pela função transacional em questão.

Tabela A.2 – Tabela de Identificação da Complexidade de Entradas Externas.

Número de Arquivos Referenciados	Número de Itens de Dados Referenciados		
	De 1 a 4	**De 5 a 15**	**16 ou mais**
0 ou 1	Simples	Simples	Média
2	Simples	Média	Complexa
3 ou mais	Média	Complexa	Complexa

Tabela A.3 – Tabela de Identificação da Complexidade de Saídas e Consultas Externas.

Número de Arquivos Referenciados	Número de Itens de Dados Referenciados		
	De 1 a 5	**De 6 a 19**	**20 ou mais**
0 ou 1	Simples	Simples	Média
2 ou 3	Simples	Média	Complexa
4 ou mais	Média	Complexa	Complexa

Cálculo dos Pontos de Função Não Ajustados

Uma vez contadas as funções de dados e as funções transacionais, é possível calcular os PFs não ajustados de uma aplicação. Esse cálculo é feito da seguinte forma:

1. Para cada um dos cinco tipos de função (ALI, AIE, EE, SE e CE), são computados os totais de pontos de função (NPF_i), segundo a seguinte expressão:

$$NPF_i = \sum_{j=1}^{3} NC_{i,j} * C_{i,j}$$

onde

- $NC_{i,j}$ = número funções do tipo *i* (*i* variando de 1 a 5, segundo os tipos de função existentes: ALI, AIE, EE, SE e CE) que foram classificados na complexidade *j* (*j* variando de 1 a 3, segundo os valores de complexidade: simples, média e complexa).
- $C_{i,j}$ = valor da contribuição da complexidade *j* no cálculo dos pontos da função *i*, dado pela tabela A.4.

Tabela A.4 – Contribuição das Funções na Contagem de PFs Não Ajustados.

Função	Complexidade		
	Simples	**Média**	**Complexa**
ALI	7	10	15
AIE	5	7	10
EE	3	4	6
SE	4	5	7
CE	3	4	6

2. O total de pontos de função não ajustados (PFNA) é dado pelo somatório dos pontos das tabelas de função:

$$PFNA = \sum_{i=1}^{5} NPF_i$$

sendo que *i* varia de 1 a 5, segundo os tipos de função existentes (AIL, AIE, EE, SE e CE).

Determinação do Fator de Ajuste

O fator de ajuste influencia os pontos de função não ajustados em ± 35%, obtendo-se o número de PFs ajustados. Para se calcular o fator de ajuste, são usadas 14 características gerais dos sistemas, a saber:

1. Comunicação de Dados
2. Processamento de Dados Distribuído
3. Desempenho
4. Utilização do Equipamento (Restrições de Recursos Computacionais)
5. Volume de Transações
6. Entrada de Dados On-line
7. Eficiência do Usuário Final (Usabilidade)
8. Atualização On-line
9. Processamento Complexo
10. Reusabilidade
11. Facilidade de Implantação
12. Facilidade Operacional (Processos Operacionais, tais como Inicialização, Cópia de Segurança, Recuperação etc)
13. Múltiplos Locais e Organizações do Usuário
14. Facilidade de Mudanças (Manutenibilidade)

Para cada uma dessas 14 características deve-se atribuir um valor de 0 (nenhuma influência) a 5 (forte influência), dito grau ou nível de influência, que indica o quanto determinada característica tem influência no sistema. Os 14 graus de influência (GIs) informados são somados, resultando no nível de influência total (NIT):

$$NIT = \sum_{i=1}^{14} GI_i$$

Finalmente, o valor do fator de ajuste (VFA) é determinado, então, pela fórmula:

$$VFA = (NIT * 0,01) + 0,65$$

Cálculo dos Pontos de Função Ajustados

Uma vez calculados os PF não ajustados e o fator de ajuste, é possível calcular os PFs ajustados. Esse cálculo é feito de formas diferentes para cada tipo de contagem (projeto de desenvolvimento, projeto de manutenção ou aplicações instaladas). Para projetos de desenvolvimento, o cálculo é dado por:

$$PF = PFNA * VFA$$

onde PFNA = Número de PFs não ajustados e
VFA = valor do fator de ajuste

Referência

C. Hazan. "Medição da Qualidade e Produtividade em Software", In: Qualidade e Produtividade em Software, 4ª edição, K.C. Weber, A.R.C. Rocha, C.J. Nascimento (organizadores), Makron Books, 2001, p. 25 – 41.

As 14 Características Gerais e seus Graus de Influência (Dias, 2004)

Grau	Descrição
0	Nenhuma influência
1	Influência mínima
2	Influência moderada
3	Influência média
4	Influência significante
5	Influência forte

1. **Comunicação de dados**: os aspectos relacionados aos recursos utilizados para a comunicação de dados do sistema deverão ser descritos de forma global. Descrever se a aplicação utiliza protocolos[4] diferentes para recebimento/envio das informações do sistema.

0. Aplicação *batch* ou funciona *stand-alone*;
1. Aplicação *batch*, mas utiliza entrada de dados ou impressão remota;
2. Aplicação *batch*, mas utiliza entrada de dados e impressão remota;
3. Aplicação com entrada de dados *on-line* para alimentar processamento *batch* ou sistema de consulta;
4. Aplicação com entrada de dados *on-line*, mas suporta apenas um tipo de protocolo de comunicação;
5. Aplicação com entrada de dados *on-line* e suporta mais de um tipo de protocolo de comunicação.

2. **Processamento de Dados Distribuído**: Esta característica refere-se a sistemas que utilizam dados ou processamento distribuído, valendo-se de diversas CPUs.

0. Aplicação não auxilia na transferência de dados ou funções entre os processadores da empresa;
1. Aplicação prepara dados para o usuário final utilizar em outro processador (do usuário final), tal como planilhas;
2. Aplicação prepara dados para transferência, transfere-os para serem processados em outro equipamento da empresa (não pelo usuário final);
3. Processamento é distribuído e a transferência de dados é *on-line* e apenas em uma direção;
4. Processamento é distribuído e a transferência de dados é *on-line* e em ambas as direções;
5. As funções de processamento são dinamicamente executadas no equipamento (CPU) mais apropriada;

[4] Protocolo é um conjunto de informações que reconhecem e traduzem para um determinado padrão, informações entre dois sistemas ou periféricos, permitindo intercâmbio das informações.

3. **Desempenho**: Trata-se de parâmetros estabelecidos pelo usuário como aceitáveis, relativos a tempo de resposta.

0. Nenhum requisito especial de desempenho foi solicitado pelo usuário;
1. Requisitos de desempenho foram estabelecidos e revistos, mas nenhuma ação especial foi requerida;
2. Tempo de resposta e volume de processamento são itens críticos durante horários de pico de processamento. Nenhuma determinação especial para a utilização do processador foi estabelecida. A data limite para a disponibilidade de processamento é sempre o próximo dia útil;
3. Tempo de resposta e volume de processamento são itens críticos durante todo o horário comercial. Nenhuma determinação especial para a utilização do processador foi estabelecida. A data-limite necessária para a comunicação com outros sistemas é limitante.
4. Os requisitos de desempenho estabelecidos requerem tarefas de análise de desempenho na fase de planejamento e análise da aplicação.
5. Além do descrito no item anterior, ferramentas de análise de desempenho foram usadas nas fases de planejamento, desenvolvimento e/ou implementação para atingir os requisitos de desempenho estabelecidos pelos usuários.

4. **Utilização do Equipamento**: Trata-se de observações quanto ao nível de utilização de equipamentos requerido para a execução do sistema. Este aspecto é observado com vista a planejamento de capacidades e custos.

0. Nenhuma restrição operacional explícita ou mesmo implícita foi incluída.
1. Existem restrições operacionais leves. Não é necessário esforço especial para atender às restrições.
2. Algumas considerações de ajuste de desempenho e segurança são necessárias.
3. São necessárias especificações especiais de processador para um módulo específico da aplicação.
4. Restrições operacionais requerem cuidados especiais no processador central ou no processador dedicado para executar a aplicação.
5. Além das características do item anterior, há considerações especiais que exigem utilização de ferramentas de análise de desempenho, para a distribuição do sistema e seus componentes, nas unidades processadoras.

5. **Volume de transações**: Consiste na avaliação do nível de influência do volume de transações no projeto, desenvolvimento, implantação e manutenção do sistema.

0. Não estão previstos períodos de picos de volume de transação.
1. Estão previstos picos de transações mensalmente, trimestralmente, anualmente ou em certo período do ano.
2. São previstos picos semanais.
3. São previstos picos diários.
4. Alto volume de transações foi estabelecido pelo usuário, ou o tempo de resposta necessário atinge nível alto o suficiente para requerer análise de desempenho na fase de projeto.
5. Além do descrito no item anterior, é necessário utilizar ferramentas de análise de desempenho nas fases de projeto, desenvolvimento e/ou implantação.

6. **Entrada de dados *on-line***: A análise desta característica permite quantificar o nível de influência exercida pela utilização de entrada de dados no modo *on-line* no sistema.

0. Todas as transações são processadas em modo *batch*.
1. De 1% a 7% das transações são entradas de dados *on-line*.
2. De 8% a 15% das transações são entradas de dados *on-line*.
3. De 16% a 23% das transações são entradas de dados *on-line*.
4. De 24% a 30% das transações são entradas de dados *on-line*.
5. Mais de 30% das transações são entradas de dados *on-line*.

7. **Usabilidade**: a análise desta característica permite quantificar o grau de influência relativo aos recursos implementados com vista a tornar o sistema amigável, permitindo incrementos na eficiência e satisfação do usuário final, tais como:

- Auxílio à navegação (teclas de função, acesso direto e menus dinâmicos)
- Menus Documentação e *help on-line*
- Movimento automático do cursor.
- Movimento horizontal e vertical de tela.
- Impressão remota (via transações *on-line*)
- Teclas de função preestabelecidas.
- Processos batch submetidos a partir de transações *on-line*
- Utilização intensa de campos com vídeo reverso, intensificados, sublinhados, coloridos e outros indicadores.
- Impressão da documentação das transações on-line através de *hard copy*
- Utilização de mouse
- Menus *pop-up*
- O menor número possível de telas para executar as funções de negócio.
- Suporte bilingue (contar como 4 itens)
- Suporte multilíngue. (contar como 6 itens)

Pontuação:

0. Nenhum dos itens descritos.
1. De um a três itens descritos.
2. De quatro a cinco dos itens descritos.
3. Mais de cinco dos itens descritos, mas não há requisitos específicos do usuário quanto à usabilidade do sistema.
4. Mais de cinco dos itens descritos e foram estabelecidos requisitos quanto à usabilidade fortes o suficiente para gerarem atividades específicas envolvendo fatores, tais como minimização da digitação, para mostrar inicialmente os valores utilizados com mais freqüência.
5. Mais de cinco dos itens descritos e foram estabelecidos requisitos quanto à usabilidade fortes o suficiente para requerer ferramentas e processos especiais para demonstrar antecipadamente que os objetivos foram alcançados.

8. **Atualizações *on-line***: Mede a influência no desenvolvimento do sistema face à utilização de recursos que visem a atualização dos Arquivos Lógicos Internos, no modo *on-line*.

0. Nenhuma.
1. Atualização *on-line* de um a três arquivos lógicos internos. O volume de atualização é baixo e a recuperação de dados é simples.
2. Atualização *on-line* de mais de três arquivos lógicos internos. O volume de atualização é baixo e a recuperação dos dados é simples.
3. Atualização *on-line* da maioria dos arquivos lógicos internos.
4. Em adição ao item anterior, é necessário proteção contra perdas de dados que foi projetada e programada no sistema.
5. Além do item anterior, altos volumes trazem considerações de custo no processo de recuperação. Processos para automatizar a recuperação foram incluídos minimizando a intervenção do operador.

9. **Processamento complexo**: a complexidade de processamento influencia no dimensionamento do sistema, e, portanto, deve ser quantificado o seu grau de influência, com base nas seguintes categorias:

- Processamento especial de auditoria e/ou processamento especial de segurança foram considerados na aplicação;
- Processamento lógico extensivo;
- Processamento matemático extensivo;
- Processamento gerando muitas exceções, resultando em transações incompletas que devem ser processadas novamente. Exemplo: transações de auto-atendimento bancário interrompidas por problemas de comunicação ou com dados incompletos;
- Processamento complexo para manusear múltiplas possibilidades de entrada/saída. Exemplo: multimídia.

Pontuação

0. Nenhum dos itens descritos.
1. Apenas um dos itens descritos.
2. Dois dos itens descritos.
3. Três dos itens descritos.
4. Quatro dos itens descritos.
5. Todos os cinco itens descritos.

10. **Reusabilidade**: a preocupação com o reaproveitamento de parte dos programas de uma aplicação em outras aplicações implica em cuidados com padronização. O grau de influência no dimensionamento do sistema é quantificado observando-se os seguintes aspectos:

0. Nenhuma preocupação com reutilização de código.
1. Código reutilizado foi usado somente dentro da aplicação.
2. Menos de 10% da aplicação foi projetada prevendo utilização posterior do código por outra aplicação.
3. 10% ou mais da aplicação foi projetada prevendo utilização posterior do código por outra aplicação.
4. A aplicação foi especificamente projetada e/ou documentada para ter seu código reutilizado por outra aplicação e a aplicação é customizada pelo usuário em nível de código fonte.
5. A aplicação foi especificamente projetada e/ou documentada para ter seu código facilmente reutilizado por outra aplicação e a aplicação é customizada para uso através de parâmetros que podem ser alterados pelo usuário.

11. **Facilidade de implantação**: a quantificação do grau de influência desta característica é feita, observando-se o plano de conversão e implantação e/ou ferramentas utilizadas durante a fase de testes do sistema.

0. Nenhuma consideração especial foi estabelecida pelo usuário e nenhum procedimento especial é requerido na implantação.
1. Nenhuma consideração especial foi estabelecida pelo usuário, mas procedimentos especiais são necessários na implementação.
2. Requisitos de conversão e implantação foram estabelecidos pelo usuário e roteiro de conversão e implantação foram providos e testados. O impacto da conversão no projeto não é considerado importante.
3. Requisitos de conversão e implantação foram estabelecidos pelo usuário e roteiro de conversão e implantação foram providos e testados. O impacto da conversão no projeto é considerado importante.
4. Além do item 2, conversão automática e ferramentas de implantação foram providas e testadas.
5. Além do item 3, conversão automática e ferramentas de implantação foram providas e testadas.

12. **Facilidade operacional**: a análise desta característica permite quantificar o nível de influência na aplicação, com relação a procedimentos operacionais automáticos que reduzem os procedimentos manuais, bem como mecanismos de inicialização, salvamento e recuperação, verificados durante os testes do sistema.

0. Nenhuma consideração especial de operação, além do processo normal de salvamento foi estabelecida pelo usuário.

1-4. Verifique quais das seguintes afirmativas podem ser identificadas na aplicação. Selecione as que forem aplicadas. Cada item vale um ponto, exceto se definido explicitamente:

- Foram desenvolvidos processos de inicialização, salvamento e recuperação, mas a intervenção do operador é necessária.
- Foram estabelecidos processos de inicialização, salvamento e recuperação, e nenhuma intervenção do operador é necessária (conte como dois itens)
- A aplicação minimiza a necessidade de montar fitas magnéticas.
- A aplicação minimiza a necessidade de manuseio de papel.

5. A aplicação foi desenhada para trabalhar sem operador, nenhuma intervenção do operador é necessária para operar o sistema além de executar e encerrar a aplicação. A aplicação possui rotinas automáticas para recuperação em caso de erro.

13. **Múltiplos Locais e Organizações do Usuário**: consiste na análise da arquitetura do projeto, observando-se a necessidade de instalação do sistema em diversos lugares.

0. Os requisitos do usuário não consideraram a necessidade de instalação em mais de um local.
1. A necessidade de múltiplos locais foi considerada no projeto e a aplicação foi desenhada para operar apenas em ambientes de software e hardware idênticos.
2. A necessidade de múltiplos locais foi considerada no projeto e a aplicação está preparada para trabalhar apenas em ambientes similares de software e hardware.
3. A necessidade de múltiplos locais foi considerada no projeto e a aplicação está preparada para trabalhar em diferentes ambientes de hardware e/ou software.
4. Plano de documentação e manutenção foram providos e testados para suportar a aplicação em múltiplos locais, além disso, os itens 1 ou 2 caracterizam a aplicação.
5. Plano de documentação e manutenção foram providos e testados para suportar a aplicação em múltiplos locais, além disso, o item 3 caracteriza a aplicação.

14. **Facilidade de mudanças**: focaliza a preocupação com a influência da manutenção no desenvolvimento do sistema. Esta influência deve ser quantificada baseando na observação de atributos, tais como:

- disponibilidade de facilidades como consultas e relatórios flexíveis para atender necessidades simples (conte como 1 item);
- disponibilidade de facilidades como consultas e relatórios flexíveis para atender necessidades de complexidade média (conte como 2 itens);
- disponibilidade de facilidades como consultas e relatórios flexíveis para atender necessidades complexas (conte 3 itens);
- se os dados de controle são armazenados em tabelas que são mantidas pelo usuário através de processos on-line, mas mudanças têm efeitos somente no dia seguinte;
- se os dados de controle são armazenados em tabelas que são mantidas pelo usuário através de processos on-line, as mudanças têm efeito imediatamente (conte como 2 itens).

Pontuação

0. Nenhum dos itens descritos.
1. Um dos itens descritos.
2. Dois dos itens descritos.
3. Três dos itens descritos.
4. Quatro dos itens descritos.
5. Todos os cinco itens descritos.

Referência:

R. Dias, "Análise por Pontos de Função: Uma Técnica para Dimensionamento de Sistemas de Informação", on-line. Disponível em: www.presidentekennedy.br/resi/edicao03/artigo02.pdf. Último acesso: 13.05.2004.