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.

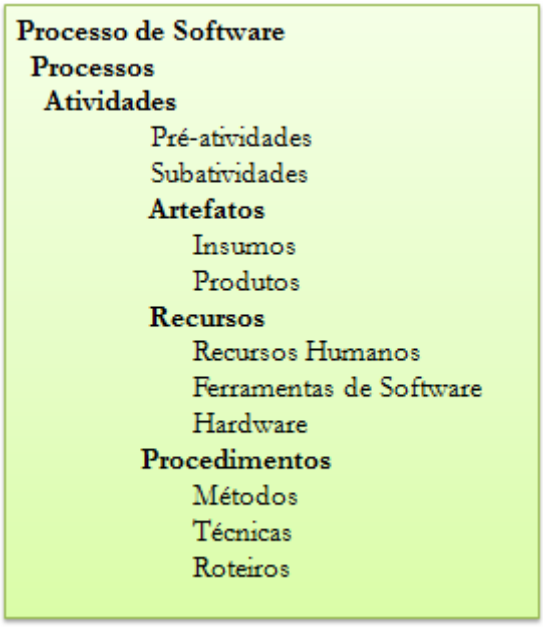

Figura 2.1 – Elementos que compõem um Processo de Software.

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.

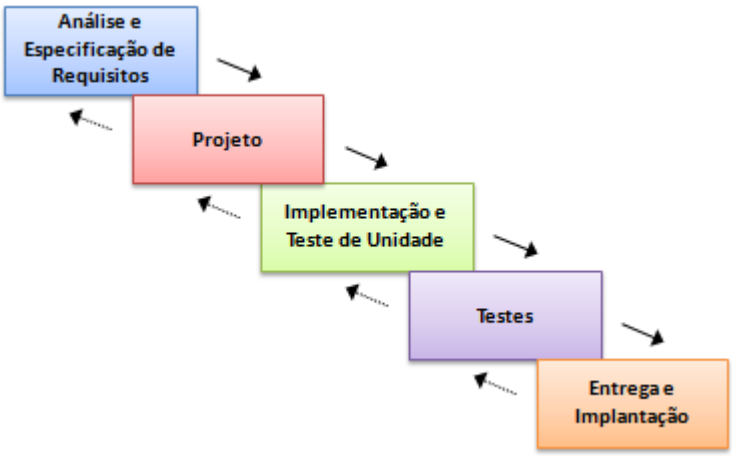

Figura 2.2 - O Modelo em Cascata.

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.

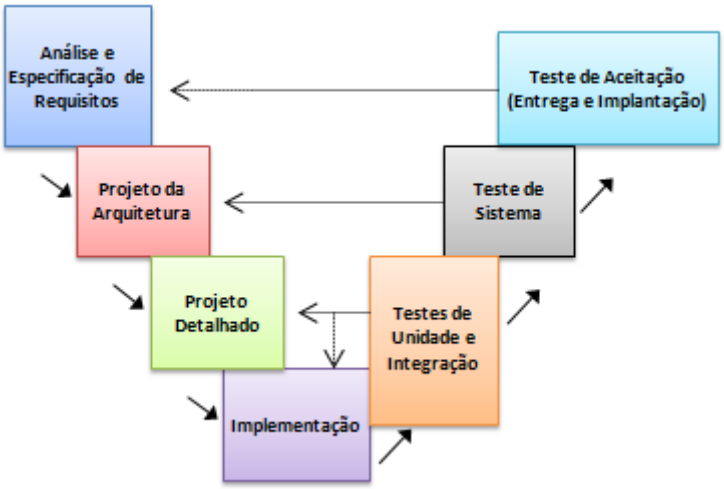

Figura 2.3 - O Modelo em V.

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.

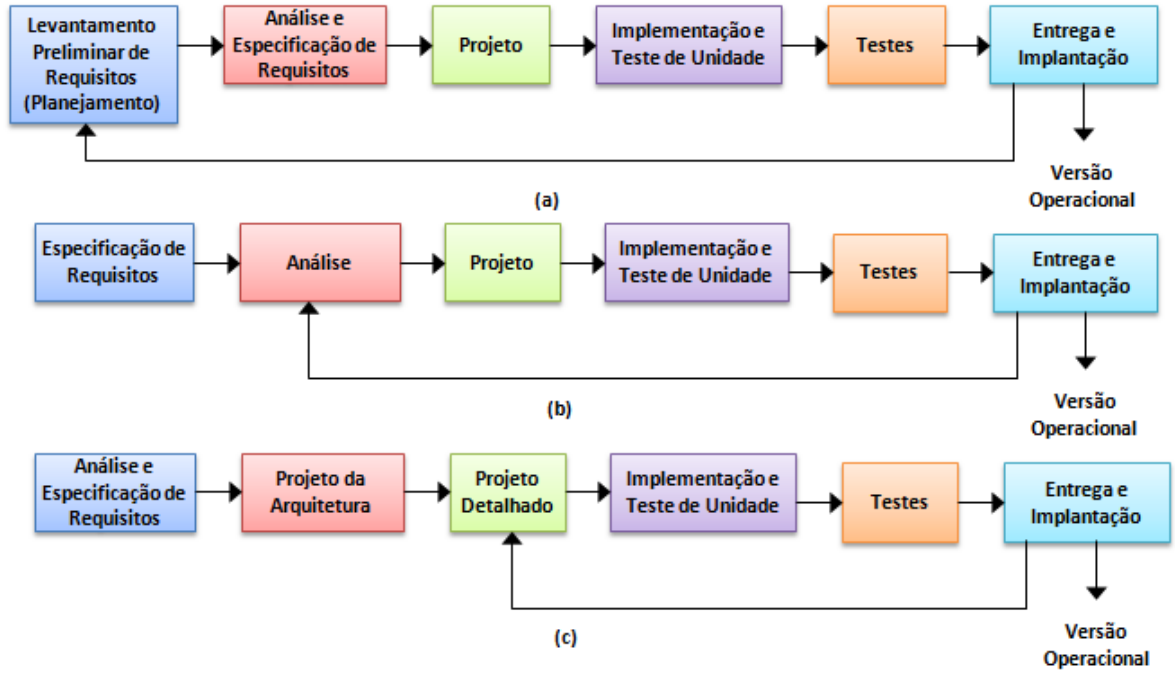

Figura 2.4 – Variações do Modelo Incremental.

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.

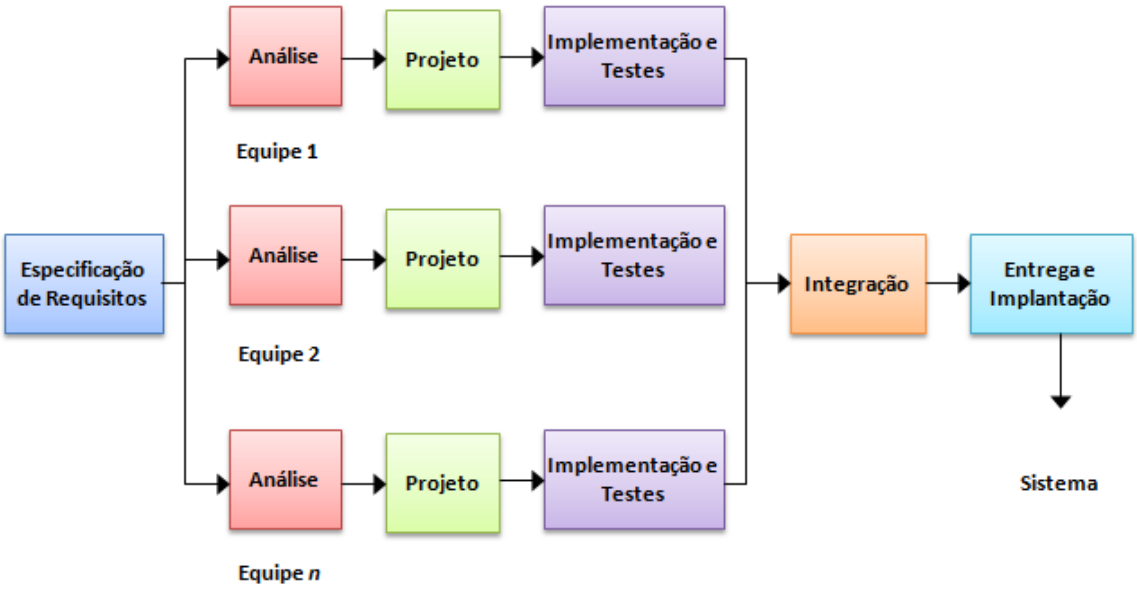

Figura 2.5 – O Modelo RAD.

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.

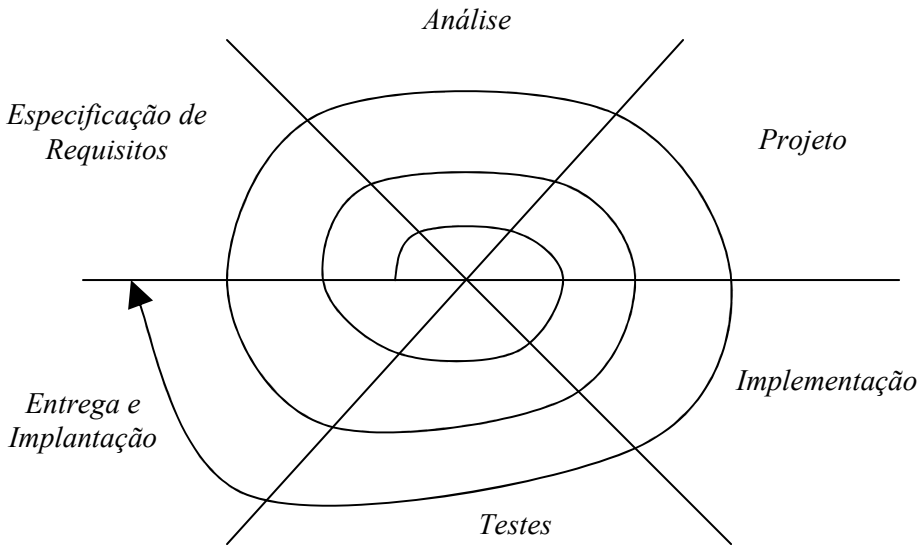

Figura 2.6 – O Modelo Espiral.

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].

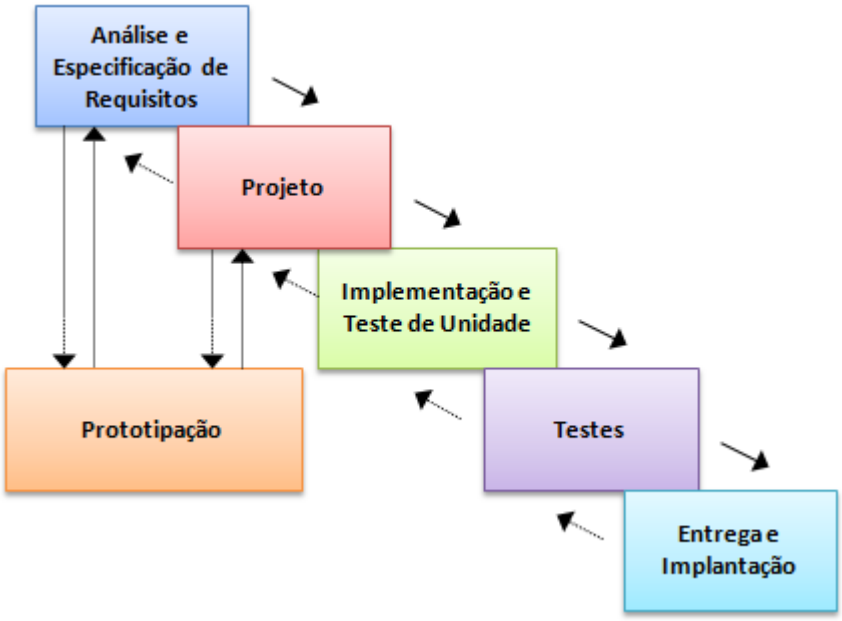

Figura 2.7 - O Modelo em Cascata com Prototipação.

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].

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

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 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).

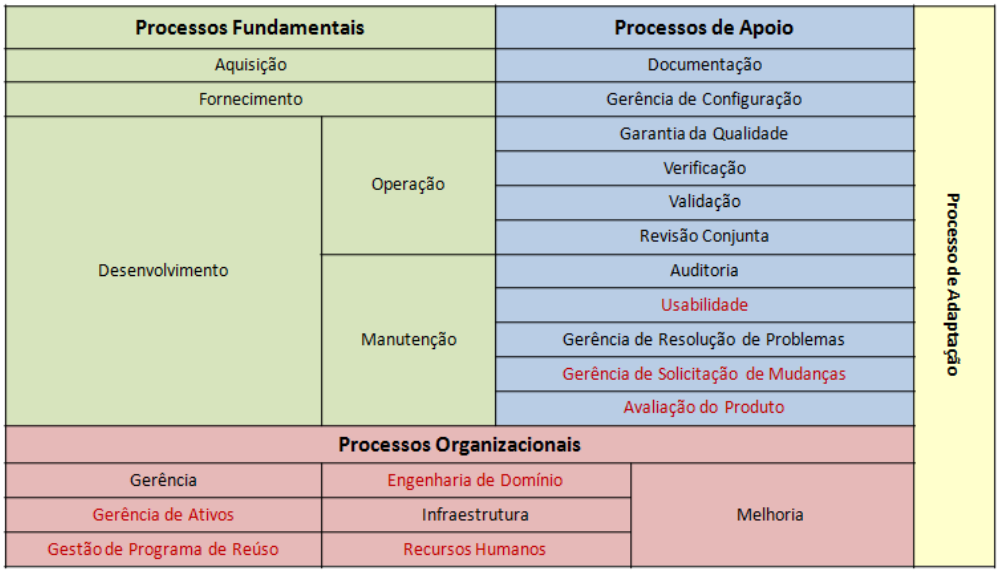

Figura 2.8 - Processos presentes na ISO/IEC 12207 – antes da versão de 2008.

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].

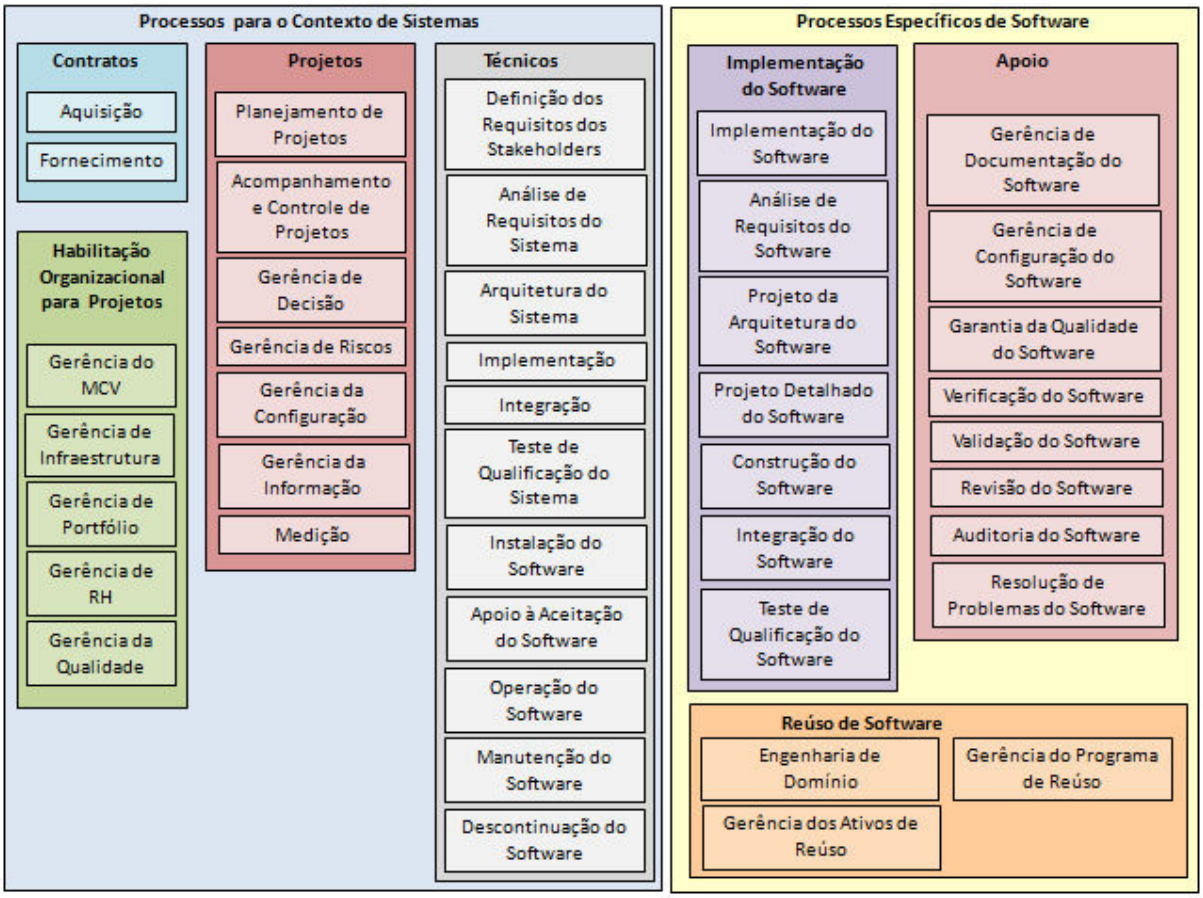

Figura 2.9 - Processos da ISO/IEC 12207:2008.

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.

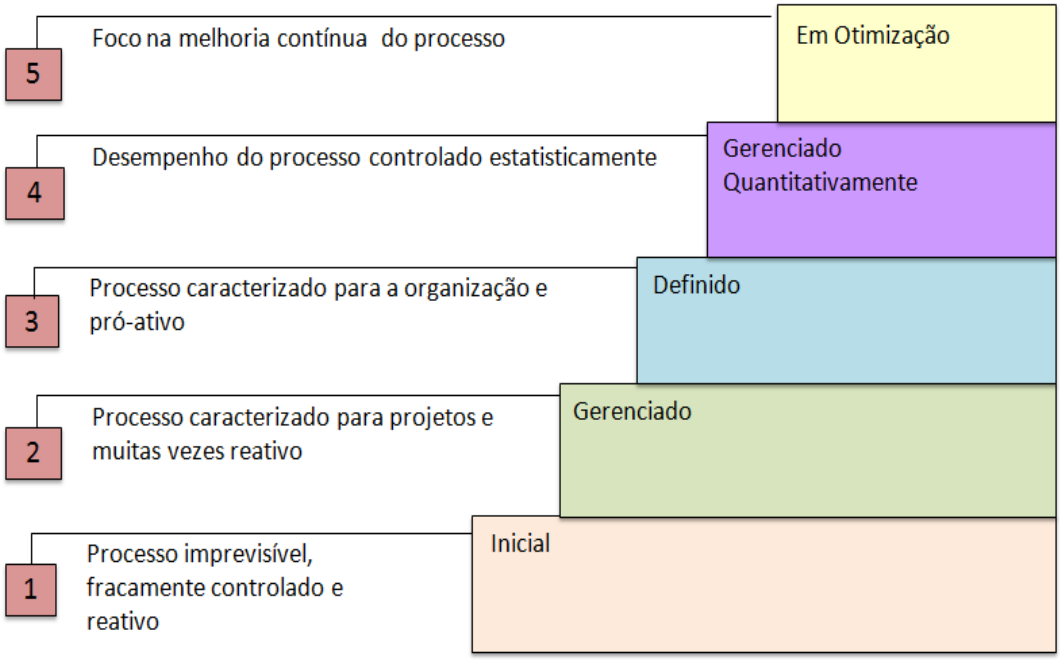

Figura 2.10 - Os níveis de maturidade do CMMI.

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 2.9 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.

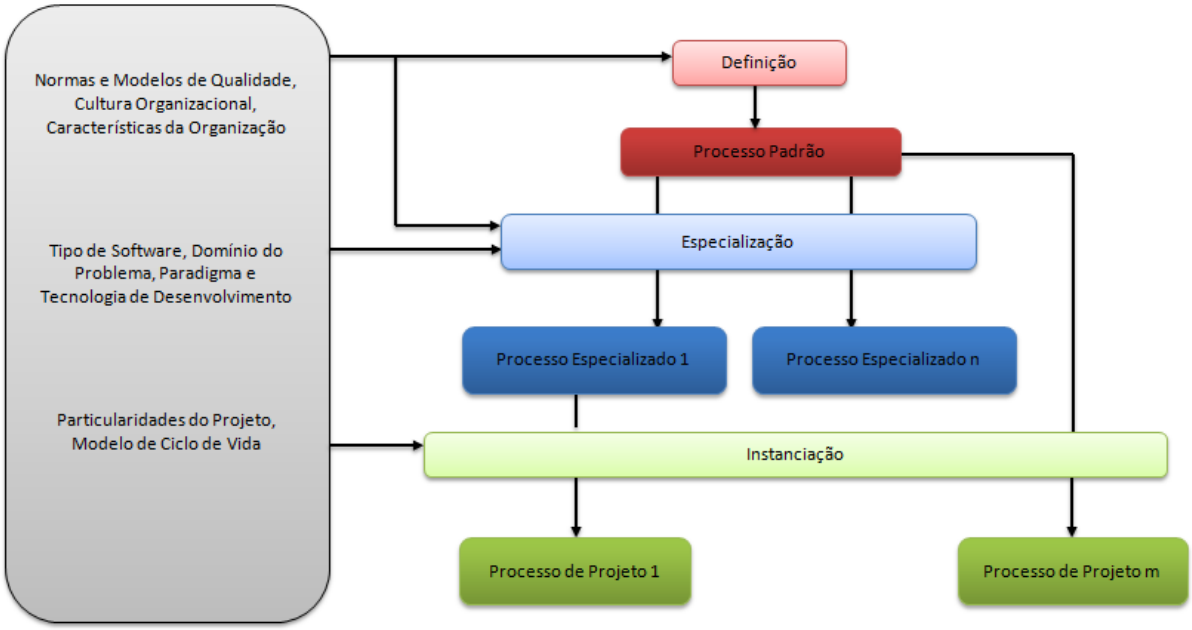

Figura 2.9 – Modelo para Definição de Processos em Níveis.

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, 2000.
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. M.B. Chrissis, M. Konrad, S. Shrum, *CMMI: Guidelines for Process Integration and Product Improvement*, 2nd Ed., Addison Wesley, 2006.
11. SOFTEX, MPS.BR – Melhoria de Processo do Software Brasileiro – Guia Geral, 2009.