

AULA 2 – Processos de Software

Os processos de software são complexos e, como todos os processos intelectuais e criativos, dependem de pessoas para tomar decisões e fazer julgamentos. Não existe um processo ideal, a maioria das organizações desenvolve os próprios processos de desenvolvimento de software.

Embora não exista um processo 'ideal' de software, há espaço, em muitas organizações, para melhorias no processo de software. Os processos podem incluir técnicas ultrapassadas ou não aproveitar as melhores práticas de engenharia de software da indústria. De fato, muitas empresas ainda não se aproveitam dos métodos da engenharia de software em seu desenvolvimento de software.

1. Introdução

Um processo de software é um conjunto de atividades relacionadas que levam à produção de um produto de software. Essas atividades podem envolver o desenvolvimento de software a partir do zero em uma linguagem padrão de programação como Java ou C, por exemplo. Atualmente, novos softwares de negócios são desenvolvidos por meio da extensão e modificação de sistemas existentes ou por meio da configuração e integração de prateleira ou componentes do sistema.

Existem muitos processos de software diferentes, mas todos devem incluir quatro atividades fundamentais para a engenharia de software:

- **Especificação de software:** A funcionalidade do software e as restrições a seu funcionamento devem ser definidas.
- **Projeto e implementação de software:** O software deve ser produzido para atender às especificações.
- **Validação de software:** O software deve ser validado para garantir que atenda às demandas do cliente.
- **Evolução de software:** O software deve evoluir para atender às necessidades de mudança dos clientes.

Essas são atividades complexas em si mesmas, que incluem subatividades como validação de requisitos, projeto de arquitetura, testes unitários etc. Existem também as atividades que dão apoio ao processo, como documentação e gerenciamento de configuração de software.

Ao descrever e discutir os processos, costumamos falar sobre suas atividades. As descrições do processo também podem incluir:

- Produtos, que são os resultados de uma das atividades do processo. Por exemplo, o resultado da atividade de projeto de arquitetura pode ser um modelo da arquitetura de software.
- Papéis, que refletem as responsabilidades das pessoas envolvidas no processo. Exemplos de papéis são: gerente de projeto, gerente de configuração, programador etc.

- Pré e pós-condições, que são declarações verdadeiras antes e depois de uma atividade do processo ou da produção de um produto. Por exemplo, antes do projeto de arquitetura ser iniciado, pode haver uma pré-condição de que todos os requisitos tenham sido aprovados pelo cliente e, após a conclusão dessa atividade, uma pós-condição poderia ser a de que os modelos UML que descrevem a arquitetura tenham sido revisados.

Os processos de software, às vezes, são categorizados como dirigidos a planos ou processos ágeis. Processos dirigidos a planos são aqueles em que todas as atividades são planejadas com antecedência, e o progresso é avaliado por comparação com o planejamento inicial.

Em processos ágeis o planejamento é gradativo, e é mais fácil alterar o processo de maneira a refletir as necessidades de mudança dos clientes. Cada abordagem é apropriada para diferentes tipos de software. Geralmente, é necessário encontrar um equilíbrio entre os processos dirigidos a planos e os processos ágeis.

1.1. Ciclo de Vida do Software

O ciclo de vida é a estrutura contendo processos, atividades e tarefas envolvidas no desenvolvimento, operação e manutenção de um produto de software, abrangendo a vida do sistema, desde a definição de seus requisitos até o término de seu uso.

O modelo de ciclo de vida é a primeira escolha a ser feita no processo de software. A partir desta escolha definir-se-á desde a maneira mais adequada de obter as necessidades do cliente, até quando e como o cliente receberá sua primeira versão operacional do sistema.

Com o crescimento do mercado de software, houve uma tendência a repetirem-se os passos e as práticas que deram certo. A etapa seguinte foi a formalização em modelos de ciclo de vida. Em outras palavras, os modelos de ciclo de vida são o esqueleto, ou as estruturas pré-definidas nas quais encaixamos as fases do processo.

A conformidade com a ABNT NBR ISO/IEC 29110, primeira norma ISO certificável para desenvolvimento de software, visa assegurar que mesmo uma pequena organização seja capaz de fornecer software de acordo com os requisitos do cliente, atendendo às suas necessidades, no prazo pretendido e com o custo acordado.

O modelo de ciclo de vida é a primeira escolha a ser feita no processo de software. A partir desta escolha definir-se-á desde a maneira mais adequada de obter as necessidades do cliente, até quando e como o cliente receberá sua primeira versão operacional do sistema. Não existe um modelo ideal. O perfil e complexidade do negócio do cliente, o tempo disponível, o custo, a equipe, o ambiente operacional são fatores que influenciarão diretamente na escolha do ciclo de vida de software a ser adotado. Da mesma forma, também é difícil uma empresa adotar um único ciclo de vida. Na maior parte dos casos, vê-se a presença de mais de um ciclo de vida no processo.

Os ciclos de vida se comportam de maneira sequencial (fases seguem determinada ordem) e/ou incremental (divisão de escopo) e/ou iterativa (retroalimentação de fases) e/ou evolutiva (software é aprimorado). Neste contexto, temos alguns modelos de ciclo de vida ou modelos de processo de software:

- Cascata
- Modelo em V

- Incremental
- Evolutivo
- RAD
- Prototipagem
- Modelo de Ciclo de Vida Associado ao RUP
- Engenharia de Software Orientada a Reuso

2. Modelos de Processo de Software

Um modelo de processo de software é uma representação simplificada de um processo de software. Cada modelo representa uma perspectiva particular do processo e, portanto, fornece informações parciais sobre ele. Temos uma série de modelos gerais de processos (também chamados 'paradigmas de processo') a partir de uma perspectiva de sua arquitetura. Ou seja, nós vemos um framework do processo, mas não vemos os detalhes de suas atividades específicas. Esses modelos genéricos não são descrições definitivas dos processos de software (Fig.1).

Esses modelos não são mutuamente exclusivos e muitas vezes são usados em conjunto, especialmente para o desenvolvimento de sistemas de grande porte. Por exemplo, para sistemas de grande porte, faz sentido combinar algumas das melhores características do modelo em cascata e dos modelos de desenvolvimento incremental. Os subsistemas dentro de um sistema maior podem ser desenvolvidos com diferentes abordagens. As partes do sistema que são bem compreendidas podem ser especificadas e desenvolvidas por meio de um processo baseado no modelo em cascata. As partes que são difíceis de especificar antecipadamente, como a interface com o usuário, devem sempre ser desenvolvidas por meio de uma abordagem incremental.

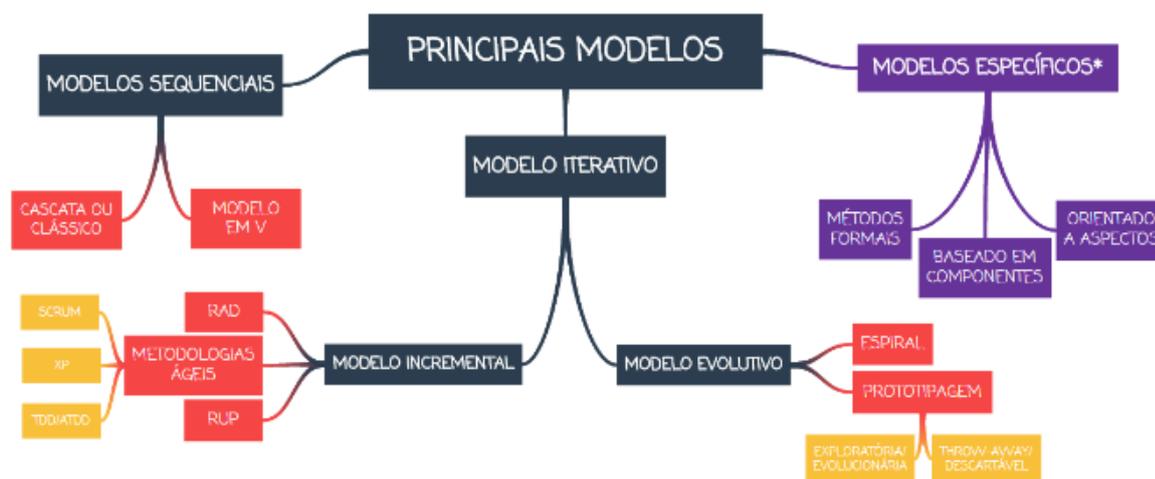


Figura 1 - Modelos de processo de desenvolvimento de software

Um modelo de processo prescritivo foi originalmente concebido para trazer ordem ao caos que existia na área de desenvolvimento de software. Durante todos esses anos que os modelos prescritivos têm sido estudados e utilizados conclui-se que esses modelos tradicionais proporcionaram uma grande contribuição quanto a sua estrutura utilizável e, além disso, eles forneceram um roteiro razoavelmente eficaz para as equipes que desenvolvem software. No entanto, esse modelo mostrou que

o trabalho de engenharia de software e o seu produto ainda permanecem instáveis ou parcialmente estruturados.

Esse modelo é denominado prescritivo, pois esses modelos prescrevem um conjunto de elementos de processo como atividades específicas do método, ações de engenharia de software, tarefas, produtos, garantia da qualidade, controles e mudanças para cada projeto. Cada modelo de processo também define um fluxo de processo, ou seja, como os elementos do processo estão inter-relacionados.

Sendo assim um modelo prescritivo é uma espécie de recomendação que pode ser adaptada ou melhorada pelo desenvolvedor de software que for adotá-la. Esses modelos abrangem três elementos principais:

- **Processos:** determinam quais são as tarefas necessárias e em que ordem elas devem ser executadas.
- **Métodos:** fornecem detalhes fundamentais de como fazer para executar as tarefas necessárias.
- **Ferramentas:** proporcionam apoio automatizado ou semiautomatizado aos processos e métodos.

A seguir, modelos prescritivos de desenvolvimento.

2.1. Modelo em Cascata

Formalizado em 1970, é o modelo mais antigo. O primeiro modelo do processo de desenvolvimento de software a ser publicado foi derivado de processos mais gerais da engenharia de sistemas.

O modelo em cascata tem o grande mérito de ser o primeiro a impor o planejamento e o gerenciamento ao processo de software, que antes era casual. O nome "cascata" foi atribuído em razão da sequência das fases, onde cada fase só começa quando a anterior termina; e da transmissão do resultado da fase anterior como entrada para a fase atual (o fim de cada fase resulta em um documento aprovado). Nesse modelo, portanto, é dada muita ênfase às fases de análise e projeto antes de partir para a programação, a fim de que o objetivo do software esteja bem definido e que sejam evitados retrabalhos.

Esse modelo considera as atividades fundamentais do processo de especificação, desenvolvimento, validação e evolução, e representa cada uma delas como fases distintas, como: especificação de requisitos, projeto de software, implementação, teste e assim por diante. Esse modelo é ilustrado na Figura 2.

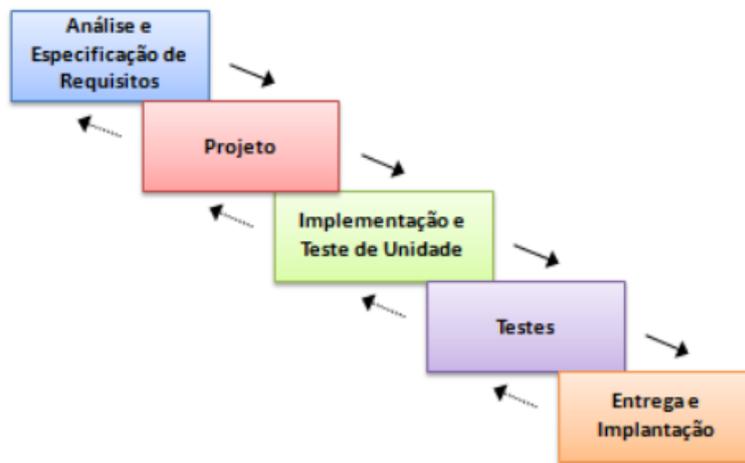


Figura 2 - Modelo em cascata

Por causa do encadeamento entre uma fase e outra, esse modelo é conhecido como 'modelo em cascata', ou ciclo de vida de software.

O modelo em cascata é um exemplo de um processo dirigido a planos — em princípio, você deve planejar e programar todas as atividades do processo antes de começar a trabalhar nelas. O modelo em cascata é consistente com outros modelos de processos de engenharia, e a documentação é produzida em cada fase do ciclo. Dessa forma, o processo torna-se visível, e os gerentes podem monitorar o progresso de acordo com o plano de desenvolvimento. Seu maior problema é a divisão inflexível do projeto em estágios distintos. Os compromissos devem ser assumidos em um estágio inicial do processo, o que dificulta que atendam às mudanças de requisitos dos clientes.

Os principais estágios do modelo em cascata refletem diretamente as atividades fundamentais do desenvolvimento:

- **Definição de requisitos:** Os serviços, restrições e metas do sistema são estabelecidos por meio de consulta aos usuários. Em seguida, são definidos em detalhes e funcionam como uma especificação do sistema.
- **Projeto do software:** O processo de projeto de sistemas aloca os requisitos tanto para sistemas de hardware como para sistemas de software, por meio da definição de uma arquitetura geral do sistema. O projeto de software envolve identificação e descrição das abstrações fundamentais do sistema de software e seus relacionamentos.
- **Implementação e teste de unidades:** Durante esse estágio, o projeto do software é desenvolvido como um conjunto de programas ou unidades de programa. O teste unitário envolve a verificação de que cada unidade atenda a sua especificação.
- **Integração e teste de sistema:** As unidades individuais do programa ou programas são integradas e testadas como um sistema completo para assegurar que os requisitos do software tenham sido atendidos. Após o teste, o sistema de software é entregue ao cliente.
- **Operação e manutenção:** Normalmente (embora não necessariamente), essa é a fase mais longa do ciclo de vida. O sistema é instalado e colocado em uso. A manutenção envolve a correção de erros que não foram descobertos em estágios iniciais do ciclo de vida, com melhora da

implementação das unidades do sistema e ampliação de seus serviços em resposta às descobertas de novos requisitos.

O resultado de cada estágio é a aprovação de um ou mais documentos. O estágio seguinte não deve ser iniciado até que a fase anterior seja concluída. Na prática, esses estágios se sobrepõem e alimentam uns aos outros de informações. Durante o projeto, os problemas com os requisitos são identificados. Durante a codificação, problemas de projeto são encontrados e assim por diante.

O processo de software não é um modelo linear simples, mas envolve o feedback de uma fase para outra. Assim, os documentos produzidos em cada fase podem ser modificados para refletirem as alterações feitas em cada um deles.

Durante o estágio final do ciclo de vida (operação e manutenção), o software é colocado em uso. Erros e omissões nos requisitos originais do software são descobertos. Os erros de programa e projeto aparecem e são identificadas novas necessidades funcionais. O sistema deve evoluir para permanecer útil. Fazer essas alterações (manutenção) pode implicar repetição de estágios anteriores do processo.

O modelo em cascata deve ser usado apenas quando os requisitos são bem compreendidos e pouco provavelmente venham a ser radicalmente alterados durante o desenvolvimento do sistema. Como é mais fácil usar um modelo de gerenciamento comum para todo o projeto, processos de software baseados no modelo em cascata ainda são comumente utilizados.

2.2. Modelo em V

Neste modelo, o modelo em cascata é colocado em forma de "V". Do lado esquerdo do V ficam da análise de requisitos até o projeto, a codificação fica no vértice e os testes, desenvolvimento, implantação e manutenção, à direita, conforme Figura 3.

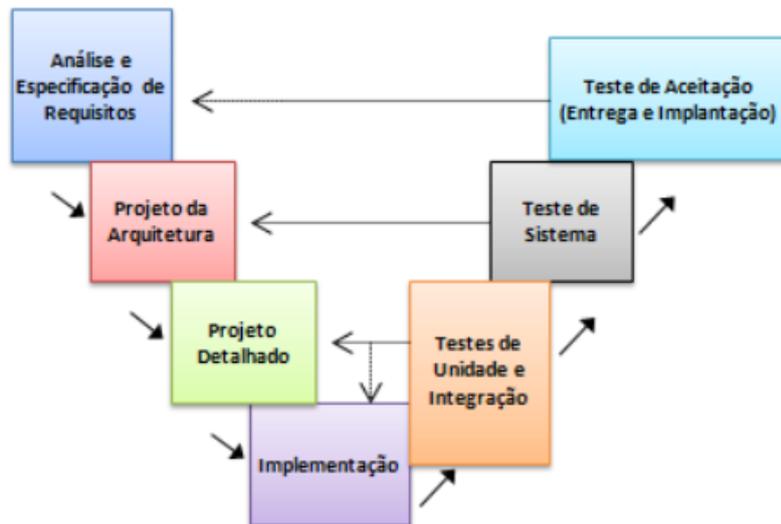


Figura 3 - Modelo em V

A característica principal desse modelo, que o diferencia do modelo em cascata, é a ênfase dada à verificação e validação: cada fase do lado esquerdo gera um plano de teste a ser executado no lado direito.

Mais tarde, o código fonte é testado, do mais baixo nível ao nível sistêmico para confirmar os resultados, seguindo os respectivos planos de teste: o teste de unidade valida o projeto do programa, o teste de sistema valida o projeto de sistema e o teste de aceitação do cliente valida a análise de requisitos.

Da mesma forma que o modelo em cascata, o cliente só recebe a primeira versão do software no final do ciclo, mas apresenta menos risco, devido ao planejamento prévio dos testes nas fases de análise e projeto.

2.3. Modelo Incremental

Neste modelo, de 1980, os requisitos do cliente são obtidos, e, de acordo com a funcionalidade, são agrupados em módulos. Após este agrupamento, a equipe, junto ao cliente, define a prioridade em que cada módulo será desenvolvido, escolha baseada na importância daquela funcionalidade ao negócio do cliente.

Cada módulo passará por todas as fases "cascata" de projeto e será entregue ao cliente um software operacional. Assim, o cliente receberá parte do produto final em menos tempo.

O modelo incremental pode ser considerado uma evolução do modelo em cascata. Este modelo é composto pelas atividades do modelo em cascata, porém estas atividades são realizadas repetidamente, ou seja, de forma iterativa. O modelo de desenvolvimento incremental é baseado na ideia de desenvolver uma implementação inicial, expô-la aos comentários dos usuários e continuar por meio da criação de várias versões até que um sistema adequado seja desenvolvido (Figura 4).

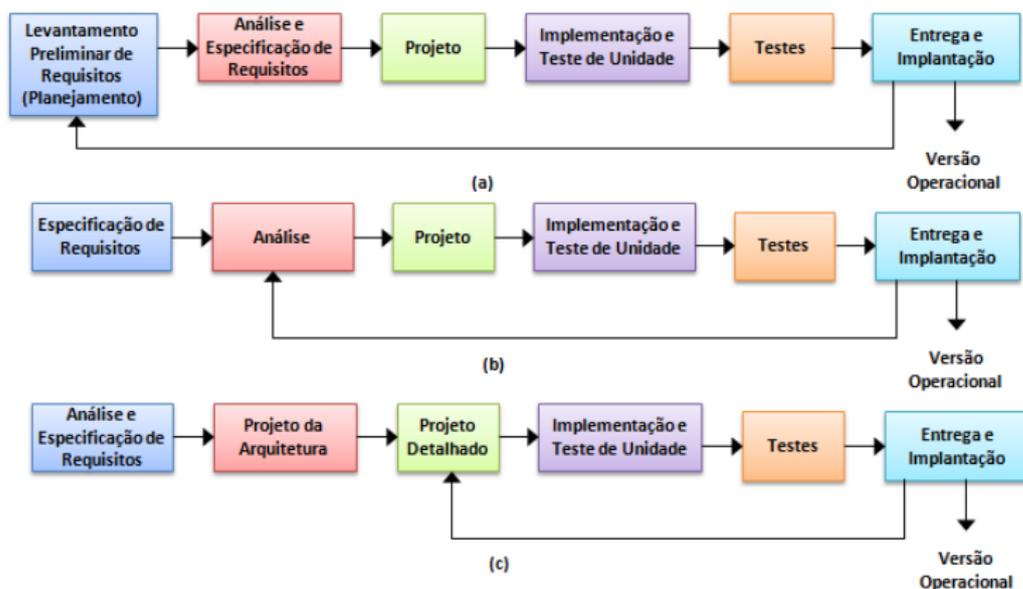


Figura 4 - Modelo incremental

Em (a), 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 em (b) e (c). Em (b), os requisitos são especificados para o sistema como um todo

e as iterações ocorrem a partir da fase de análise. Em (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.

Atividades de especificação, desenvolvimento e validação são intercaladas, e não separadas, com rápido feedback entre todas as atividades. Essa abordagem intercala as atividades de especificação, desenvolvimento e validação. O sistema é desenvolvido como uma série de versões (incrementos), de maneira que cada versão adiciona funcionalidade à anterior. Trata-se de uma parte fundamental das abordagens ágeis, é melhor do que uma abordagem em cascata para a maioria dos sistemas de negócios, e-commerce e sistemas pessoais.

Desenvolvimento incremental reflete a maneira como resolvemos os problemas. Raramente elaboramos uma completa solução do problema com antecedência; geralmente movemo-nos passo a passo em direção a uma solução, recuando quando percebemos que cometemos um erro. Ao desenvolver um software de forma incremental, é mais barato e mais fácil fazer mudanças no software durante seu desenvolvimento.

Cada incremento ou versão do sistema incorpora alguma funcionalidade necessária para o cliente. Os incrementos iniciais normalmente incluem a funcionalidade mais importante ou mais urgente. Isso significa que o cliente pode avaliar o sistema em um estágio relativamente inicial do desenvolvimento para ver se ele oferece o que foi requisitado. Em caso negativo, só o incremento que estiver em desenvolvimento no momento precisará ser alterado e, possivelmente, nova funcionalidade deverá ser definida para incrementos posteriores.

O desenvolvimento incremental tem três vantagens importantes quando comparado ao modelo em cascata:

- O custo de acomodar as mudanças nos requisitos do cliente é reduzido. A quantidade de análise e documentação a ser refeita é muito menor do que o necessário no modelo em cascata.
- É mais fácil obter feedback dos clientes sobre o desenvolvimento que foi feito. Os clientes podem fazer comentários sobre as demonstrações do software e ver o quanto foi implementado. Os clientes têm dificuldade em avaliar a evolução por meio de documentos de projeto de software.
- É possível obter entrega e implementação rápida de um software útil ao cliente, mesmo se toda a funcionalidade não for incluída. Os clientes podem usar e obter ganhos a partir do software inicial antes do que é possível com um processo em cascata.

O desenvolvimento incremental é a abordagem mais comum para o desenvolvimento de sistemas aplicativos na atualidade. Essa abordagem pode ser tanto dirigida a planos, ágil, ou uma mescla dessas abordagens. Em uma abordagem dirigida a planos, os incrementos do sistema são identificados previamente; se uma abordagem ágil for adotada, os incrementos iniciais são identificados, mas o desenvolvimento de incrementos posteriores depende do progresso e das prioridades dos clientes.

Do ponto de vista do gerenciamento, a abordagem incremental tem dois problemas:

- O processo não é visível: Os gerentes precisam de entregas regulares para mensurar o progresso. Se os sistemas são desenvolvidos com rapidez, não é economicamente viável produzir documentos que reflitam cada uma das versões do sistema.
- A estrutura do sistema tende a se degradar com a adição dos novos incrementos: A menos que tempo e dinheiro sejam dispendidos em refatoração para melhoria do software, as constantes mudanças tendem a corromper sua estrutura. Incorporar futuras mudanças do software torna-se cada vez mais difícil e oneroso.

Os problemas do desenvolvimento incremental são particularmente críticos para os sistemas de vida-longa, grandes e complexos, nos quais várias equipes desenvolvem diferentes partes do sistema. Sistemas de grande porte necessitam de um framework ou arquitetura estável, e as responsabilidades das diferentes equipes de trabalho do sistema precisam ser claramente definidas, respeitando essa arquitetura. Isso deve ser planejado com antecedência, e não desenvolvido de forma incremental.

2.4. Modelo Evolutivo

Neste modelo, os requisitos são adquiridos em paralelo à evolução do sistema. O modelo evolutivo parte do princípio de que o cliente não expõe todos os requisitos, ou os requisitos não são tão bem conhecidos, ou os requisitos ainda estão sofrendo mudanças. Desta forma, a análise é feita em cima dos requisitos conseguidos até então, e a primeira versão é entregue ao cliente. O cliente usa o software no seu ambiente operacional, e como feedback, esclarece o que não foi bem entendido e dá mais informações sobre o que precisa e sobre o que deseja (ou seja, mais requisitos).

A partir deste feedback, nova análise, projeto e desenvolvimento são realizados, e uma segunda versão do software é entregue ao cliente que, novamente, retorna com mais feedbacks. Assim, o software vai evoluindo, se tornando mais completo, até atender todas as necessidades do cliente dentro do escopo estabelecido. Tem-se assim a versão final, pelo menos até novos requisitos aparecerem (Fig. 5).

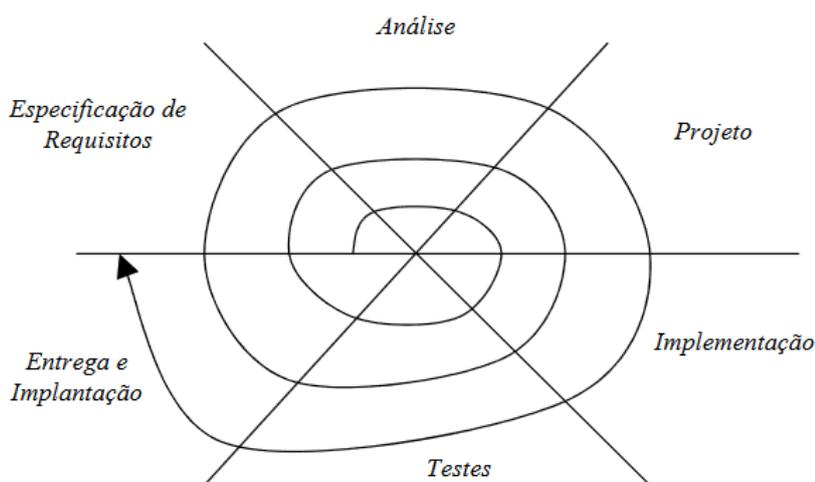


Figura 5 - Modelo evolutivo em espiral

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.

Este é um modelo guiado por risco, suporta sistemas complexos e/ou de grande porte, onde falhas não são toleráveis. Para isso, a cada iteração há uma atividade dedicada à análise de riscos e apoiada através de geração de protótipos, não necessariamente operacionais (desenhos de tela, por exemplo) para que haja um envolvimento constante do cliente nas decisões. A participação constante do cliente é uma grande vantagem desse modelo, o que diminui o risco de má interpretação de requisitos dos modelos que só oferecem a primeira versão do software no final do processo. Da mesma forma, o software já atende algumas necessidades do cliente muito mais cedo no processo.

Não é dada muita ênfase à documentação, pois a geração de versões torna este trabalho muito árduo. Além disso, como a análise de requisitos e desenvolvimento estão sempre acontecendo, a preocupação em documentar todo o processo pode fazer com que haja atrasos na entrega.

Há uma alta necessidade de gerenciamento nesse tipo de modelo, pois a falta de documentação adequada, o escopo de requisitos não determinado, o software crescendo e estando ao mesmo tempo em produção podem ter consequências negativas. Seguem alguns exemplos: o sistema nunca terminar, pois o cliente sempre pede uma alteração; o sistema não ter uma estrutura robusta a falhas nem propícia a uma fácil manutenção, pelas constantes alterações; o cliente mudar de ideia radicalmente entre uma versão e outra ou revelar um requisito que exija uma versão bem diferente da anterior, fazendo com que toda a base (de dados ou de programação) precise ser revista. Os citados problemas podem implicar em um grande ônus financeiro e de tempo.

É muito importante que o cliente esteja ciente do que se trata este ciclo de vida e que sejam esclarecidos os limites de escopo e de tempo, para que não haja frustrações de expectativas.

2.5. Modelo RAD (Rapid Application Development)

Este modelo formalizado em 1991, como uma evolução da “prototipagem rápida”, destaca-se pelo desenvolvimento rápido da aplicação. O ciclo de vida é extremamente comprimido, de forma a encontrarem-se exemplos, na literatura, de duração de 60 e 90 dias. É ideal para clientes buscando lançar soluções pioneiras no mercado.

O RAD é um método para desenvolvimento de software que permite uma prototipagem mais rápida e entrega iterativa do produto final. Trata-se de um modelo alternativo ao tradicional modelo cascata que, em geral, foca em um processo de desenvolvimento sequencial e pouco flexível.

Apresentado pela primeira vez ainda no início da década de 1990, o RAD popularizou-se rapidamente como uma metodologia ágil e poderosa. Se comparado a outros modelos de desenvolvimento, sua característica mais marcante é a potente maleabilidade de adaptação dos processos e a capacidade de manter-se em constante evolução.

No caso de a equipe ser grande, um modelo possível é o modelo RAD. Ele é recomendável quando uma aplicação pode ser modularizada.

É um ciclo de vida incremental, iterativo, onde é preferível que os requisitos tenham escopo restrito. A diferença principal do ciclo anterior é o forte paralelismo das atividades, requerendo, assim, módulos bastante independentes. Aqui os incrementos são desenvolvidos ao mesmo tempo, por equipes diferentes.

Além do paralelismo, a conquista do baixo tempo se dá graças à compressão da fase de requisitos e da fase de implantação. Isso significa que, na obtenção dos requisitos, costumam-se optar por metodologias mais dinâmicas e rápidas, como workshops ao invés de entrevistas. Permite-se também um desenvolvimento inicial no nível mais alto de abstração dos requisitos visto o envolvimento maior do usuário e visibilidade mais cedo dos protótipos (Fig. 6).

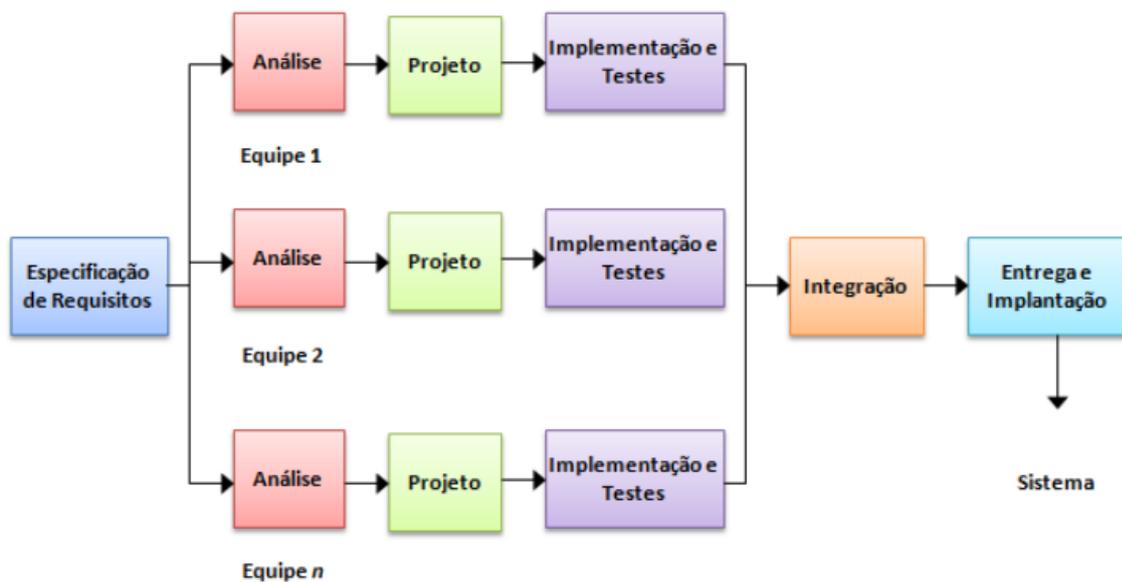


Figura 6 - Modelo RAD

As fábricas de software que resolvem por adotar este modelo devem ter uma estrutura prévia diferencial de pessoas e ferramentas, tais como:

- **Pessoas:**

- Profissionais experientes (funcional e gerência);
- Profissionais de rápida adaptação;
- Equipes de colaboração mútua;
- Maior quantidade de pessoas;

- **Gerenciamento:**

- Empresas pouco burocráticas que encorajem a eliminação de obstáculos;
- Alto controle do tempo;

- **Uso de Ferramentas:**

- CASE (Computer-Aided Software Engineering);
- Muita diagramação;

- Prévia biblioteca de componentes reutilizáveis (APIs, wizards, templates etc.);
- Fácil manutenção (ex.: linguagens de programação que suportem Orientação a Objetos, tratamento de exceção, ponteiros);
- Adoção de ferramentas maduras, pois não há tempo de atualizar versões e tratar erros inesperados;

Os sistemas desenvolvidos no ciclo RAD tendem a ter uma padronização de telas muito forte, devido a bibliotecas reutilizáveis e templates, porém tendem a perder em desempenho do sistema e na análise de risco (atividades estas que demandam tempo em qualquer projeto). Assim, é preferível seu uso para softwares de distribuição pequena.

2.6. Prototipagem

Um protótipo é uma versão inicial de um sistema de software, usado para demonstrar conceitos, experimentar opções de projeto e descobrir mais sobre o problema e suas possíveis soluções. O desenvolvimento rápido e iterativo do protótipo é essencial para que os custos sejam controlados e os contratantes do sistema possam experimentá-lo no início do processo de software.

Um protótipo pode ser usado em um processo de desenvolvimento de software para ajudar a antecipar as mudanças que podem ser requisitadas:

- No processo de engenharia de requisitos, um protótipo pode ajudar na elicitacão e validacão de requisitos de sistema.
- No processo de projeto de sistema, um protótipo pode ser usado para estudar soluçoes específicas do software e para apoiar o projeto de interface de usuário.

Um protótipo ser considerado um ciclo de vida ou pode ser usado como ferramenta em outros ciclos de vida. Um protótipo em engenharia de software pode ser o desenho de uma tela, um software contendo algumas funcionalidades do sistema. São considerados operacionais (quando já podem ser utilizados pelo cliente no ambiente real, ou seja, em produçao), ou não operacionais (não estão aptos para serem utilizados em produçao). Os protótipos podem ser descartados, ou reaproveitados para evoluírem até a versão final.

No ciclo de vida de prototipagem, não é exigido um conhecimento aprofundado dos requisitos num primeiro momento. Isso é bastante útil quando os requisitos não são totalmente conhecidos, são muitos complexos ou confusos. Desta forma, se o cliente não sabe expressar o que deseja (o que ocorre bastante quando não é um sistema legado), a melhor maneira de evitar que se perca tempo e recursos com uma má interpretação é a construçao de modelos, ou seja, de protótipos do que o software faria.

Assim, o cliente experimentará, na prática, como o sistema ou parte dele funcionará. A partir desse primeiro contato, o cliente esclarece o que não foi bem interpretado, aprofunda alguns conceitos e até descobre um pouco mais sobre o que realmente precisa. A partir deste feedback, novos requisitos são colhidos e o projeto ganha maior profundidade. Outro protótipo é gerado e apresentado ao cliente, que retorna com mais feedbacks. Ou seja, o cliente participa ativamente do início ao fim do processo (Fig. 7).

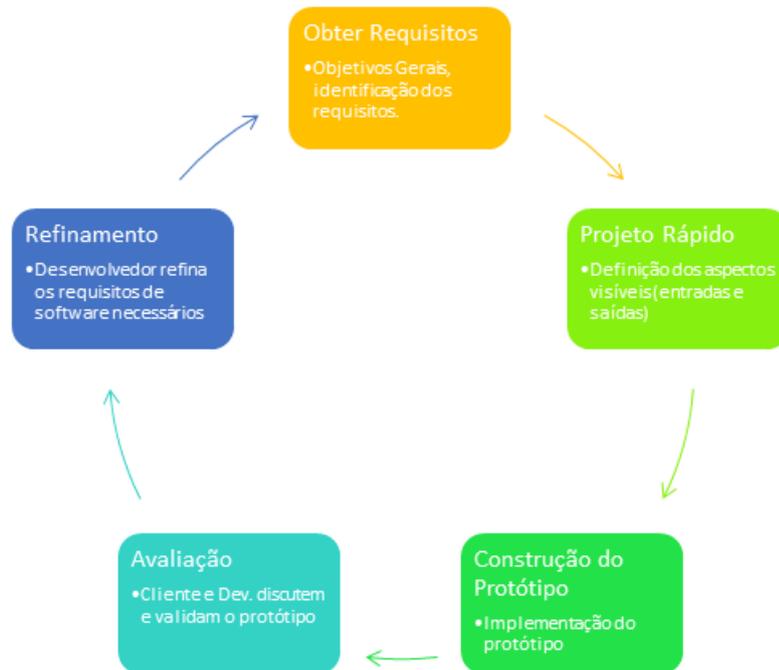


Figura 7 - Prototipagem

A geração de protótipos pode ser facilitada por ferramentas geradoras de telas, de relatórios, poupando esforço de programação e diminuindo o tempo de entrega. Cada protótipo tem uma finalidade diferente. Um protótipo pode servir para esclarecer dúvidas sobre uma rotina, demonstrar a aparência das telas, conteúdo de tabelas, formato de relatórios. Os protótipos podem também ser utilizados para apresentar opções ao cliente para que ele escolha a que mais lhe agrada, como opções de navegação, de fluxo de telas, entre outras.

É importante explicar previamente ao cliente que protótipos são apenas modelos para melhorar a comunicação. Caso contrário, pode causar uma frustração por não funcionar corretamente, ter funções limitadas, ter resposta lenta, ou a aparência ruim. Certamente um protótipo construído para esclarecer uma rotina provavelmente terá uma “cara feia”; para demonstrar a aparência das telas, não terá funcionalidade; para apresentar o formato dos relatórios, os dados não serão coerentes.

O cliente fará comparações entre o sistema final e o que foi “prometido” através do protótipo e pode ficar insatisfeito. Por exemplo, geralmente o protótipo não acessa rede ou banco de dados, pois as informações são “desenhadas” com a tela, fazendo com que tudo fique muito rápido. Já no ambiente operacional haverá uma degradação de desempenho e o cliente pode se decepcionar.

Faz parte de um bom gerenciamento no modelo de prototipagem planejar se, quais e que funções dos protótipos não operacionais serão reaproveitadas na versão operacional, para que sua confecção siga as boas práticas de engenharia de software. Os protótipos não operacionais são construídos com pouca qualidade em prol da velocidade. Ou seja, não há preocupação na programação, em refinar o código, em usar comentários, em aproveitar eficientemente os recursos de hardware e software, na manutenção, no reuso de componentes e na integração com outras funções ou sistemas. Com certeza será um problema se a equipe sucumbir à pressão do cliente, cada vez mais ansioso para ver a versão final daquele trabalho, e transformar à revelia, protótipos não operacionais em operacionais.

O gerente também deve se preocupar com o escopo do projeto versus a quantidade de protótipos, para que não se perca muito tempo nesse processo, tampouco se transforme num processo de “tentativa e erro”. Não é uma tarefa fácil documentar o modelo de ciclo de vida baseado na prototipagem devido aos requisitos não serem totalmente conhecidos no primeiro momento e a consequente quantidade de mudanças ocorridas.

2.7. Modelo de Ciclo de Vida Associado ao RUP

O objetivo do RUP é produzir software com qualidade (melhores práticas de engenharia de software) que satisfaça as necessidades dos clientes dentro de um prazo e orçamento estabelecidos. Este modelo foi desenvolvido pela Rational Software Corporation e adquirido pela IBM, que o define da seguinte maneira: “IBM Rational Unified Process®, ou RUP, é uma plataforma de processo de desenvolvimento de software configurável que oferece melhores práticas comprovadas e uma arquitetura configurável” (Figuras 8 e 9).

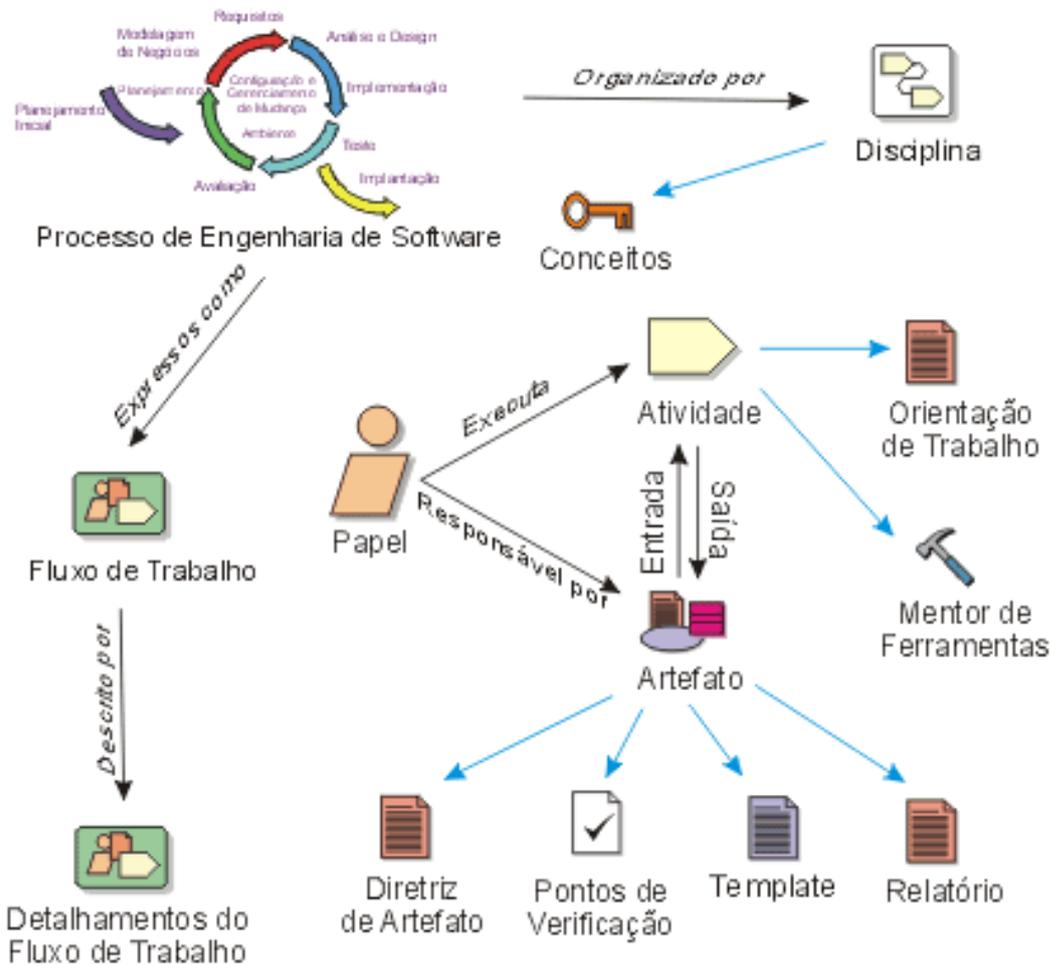


Figura 8 - Conceitos chave RUP



Figura 9 - Processo de Engenharia de Software na visão do RUP (detalhe)

Derivado da UML e do Processo Unificado de Desenvolvimento de Software, o RUP, Rational Unified Process, é um modelo de processo iterativo e incremental, dividido em fases, orientado a casos de uso. Possui framework (esqueleto) de processo e manuais que guiam na utilização das melhores práticas de especificação de projeto. Ao contrário do modelo em cascata, no qual as fases são equalizadas com as atividades do processo, as fases do RUP são estreitamente relacionadas ao negócio, e não a assuntos técnicos.

O RUP é composto por 4 fases (concepção, elaboração, construção e transição) cada uma com objetivos específicos (Figura 10). No RUP, a iteração é apoiada de duas maneiras. Cada fase pode ser executada de forma iterativa com os resultados desenvolvidos de forma incremental. Além disso, todo o conjunto de fases também pode ser executado de forma incremental, como indicado pela seta curva de 'transição' para concepção.

Todas as fases têm ao final um marco (milestone) de verificação de quais objetivos da fase foram alcançados. As disciplinas representam o que deve ser feito pelos responsáveis em termos de atividades e artefatos. O RUP fornece modelos (templates) para cada artefato e roteiros (guidelines) para a execução de suas atividades.

- **Fase 1: Concepção** - O objetivo da fase de concepção é estabelecer um business case para o sistema. Você deve identificar todas as entidades externas (pessoas e sistemas) que vão interagir com o sistema e definir as interações. Então, você deve usar essas informações para avaliar a contribuição do sistema para o negócio. Se essa contribuição for pequena, então o projeto poderá ser cancelado depois dessa fase.
- **Fase 2: Elaboração** - As metas da fase de elaboração são desenvolver uma compreensão do problema dominante, estabelecer um framework da arquitetura para o sistema, desenvolver o plano do projeto e identificar os maiores riscos do projeto. No fim dessa fase, você deve ter um modelo de requisitos para o sistema, que pode ser um conjunto de casos de uso da UML, uma descrição da arquitetura ou um plano de desenvolvimento do software.
- **Fase 3: Construção** - A fase de construção envolve projeto, programação e testes do sistema. Durante essa fase, as partes do sistema são desenvolvidas em paralelo e integradas. Na conclusão dessa fase, você deve ter um sistema de software já funcionando, bem como a documentação associada pronta para ser entregue aos usuários.

- **Fase 4: Transição** - A fase final do RUP implica transferência do sistema da comunidade de desenvolvimento para a comunidade de usuários e em seu funcionamento em um ambiente real. Isso é ignorado na maioria dos modelos de processo de software, mas é, de fato, uma atividade cara e, às vezes, problemática. Na conclusão dessa fase, você deve ter um sistema de software documentado e funcionando corretamente em seu ambiente operacional.

Iniciação	Elaboração	Construção	Transição
Objetivos do Ciclo de Vida	Arquitetura do Ciclo de Vida	Recurso Operacional Inicial	Liberação do Produto
<ul style="list-style-type: none"> -Escopo do sistema -Requisitos do sistema -Custo geral do sistema -Riscos em potencial 	<ul style="list-style-type: none"> -Baseline da Arquitetura -Riscos em potencial -Componentes do Sistema -Reusabilidade 	<ul style="list-style-type: none"> -Qualidade do sistema -Versões Alfa e Beta -Release do Sistema 	<ul style="list-style-type: none"> -Teste Beta -Conversão do BD -Treinamentos -Distribuição
<ul style="list-style-type: none"> -Documento de Visão -Lista de Riscos -Plano de Iteração -Glossário -Modelo de Caso de Uso -Protótipos 	<ul style="list-style-type: none"> -Protótipo -Modelo de Design -Modelo de Dados -Modelo de Implantação 	<ul style="list-style-type: none"> -Release do Sistema -Casos de Testes -Material de Suporte 	<ul style="list-style-type: none"> -Release -Material de Suporte -Casos de Testes -Pacote de Distribuição
FASES	MARCOS	OBJETIVOS	ARTEFATOS

Figura 10 - Fases RUP

Em resumo, na fase de concepção, deve-se estabelecer o escopo e a viabilidade econômica do projeto. Na elaboração, o objetivo é eliminar os principais riscos e estabelecer uma arquitetura estável a partir da qual o sistema poderá evoluir. Na fase de construção, um produto completo é desenvolvido de maneira iterativa até que esteja pronto para ser passado aos usuários, o que ocorre na fase de transição, onde uma versão beta do sistema é disponibilizada. No final da transição pode ser iniciado um novo ciclo de desenvolvimento para a evolução do produto, o que envolveria todas as fases novamente.

A visão estática do RUP prioriza as atividades que ocorrem durante o processo de desenvolvimento. Na descrição do RUP, essas são chamadas workflows. Existem seis workflows centrais, identificadas no processo, e três workflows de apoio. O RUP foi projetado em conjunto com a UML, assim, a descrição do workflow é orientada em torno de modelos associados à UML, como modelos de sequência, modelos de objetos etc. Os workflows centrais de engenharia e de apoio estão descritos na Tabela 1.

Tabela 1 - Workflows estáticos no Rational Unified Process

WORKFLOW	DESCRIÇÃO
Modelagem de negócios	Os processos de negócio são modelados por meio de casos de uso de negócios.
Requisitos	Atores que interagem com o sistema são identificados e casos de uso são desenvolvidos para modelar os requisitos do sistema.
Análise e projeto	Um modelo de projeto é criado e documentado com modelos de arquitetura, modelos de componentes, modelos de objetos e modelos de sequência.
Implementação	Os componentes do sistema são implementados e estruturados em subsistemas de implementação. A geração automática de código a partir de modelos de projeto ajuda a acelerar esse processo.
Teste	O teste é um processo iterativo que é feito em conjunto com a implementação. O teste do sistema segue a conclusão da implementação.
Implantação	Um <i>release</i> do produto é criado, distribuído aos usuários e instalado em seu local de trabalho.
Gerenciamento de configuração e mudanças	Esse <i>workflow</i> de apoio gerencia as mudanças do sistema (veja o Capítulo 25).
Gerenciamento de projeto	Esse <i>workflow</i> de apoio gerencia o desenvolvimento do sistema (veja os capítulos 22 e 23).
Meio ambiente	Esse <i>workflow</i> está relacionado com a disponibilização de ferramentas apropriadas para a equipe de desenvolvimento de software.

2.8. Engenharia de Software Orientada a Reuso

Na maioria dos projetos de software, há algum reuso de software. Isso acontece muitas vezes informalmente, quando as pessoas envolvidas no projeto sabem de projetos ou códigos semelhantes ao que é exigido. Elas os buscam, fazem as modificações necessárias e incorporam-nos a seus sistemas. Essa abordagem é baseada na existência de um número significativo de componentes reusáveis. O processo de desenvolvimento do sistema concentra-se na integração desses componentes em um sistema já existente em vez de desenvolver um sistema a partir do zero.

Abordagens orientadas a reuso dependem de uma ampla base de componentes reusáveis de software e de um framework de integração para a composição desses componentes. Em alguns casos, esses componentes são sistemas completos, capazes de fornecer uma funcionalidade específica, como processamento de texto ou planilha.

Um modelo de processo geral de desenvolvimento baseado no reuso é mostrado na Figura 11.

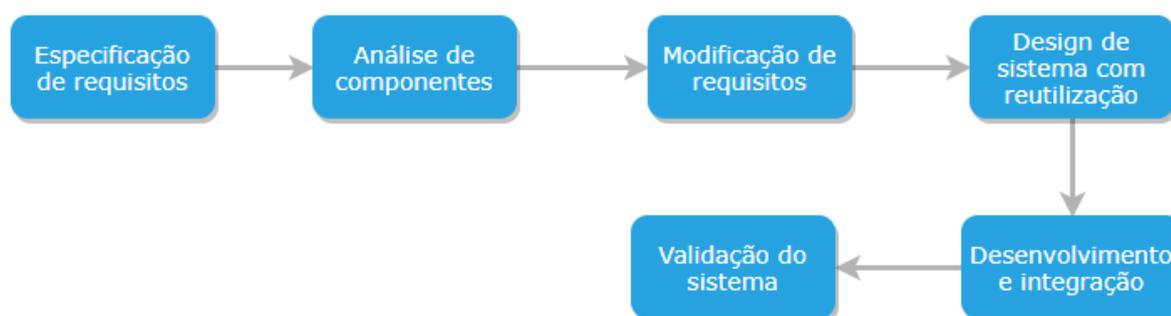


Figura 11 – Engenharia de software orientada a reuso

Embora o estágio de especificação de requisitos iniciais e o estágio de validação sejam comparáveis a outros processos de software, os estágios intermediários em um processo orientado a reuso são diferentes:

- **Análise de componentes:** Dada a especificação de requisitos, é feita uma busca por componentes para implementar essa especificação. Em geral, não há correspondência exata, e os componentes que podem ser usados apenas fornecem alguma funcionalidade necessária.
- **Modificação de requisitos:** Durante esse estágio, os requisitos são analisados usando-se informações sobre os componentes que foram descobertos. Em seguida, estes serão modificados para refletir os componentes disponíveis. No caso de modificações impossíveis, a atividade de análise dos componentes pode ser reinserida na busca por soluções alternativas.
- **Projeto do sistema com reuso:** Durante esse estágio, o framework do sistema é projetado ou algo existente é reusado. Os projetistas têm em mente os componentes que serão reusados e organizam o framework para reuso. Alguns softwares novos podem ser necessários, se componentes reusáveis não estiverem disponíveis.
- **Desenvolvimento e integração:** Softwares que não podem ser adquiridos externamente são desenvolvidos, e os componentes e sistemas COTS (Commercial Off-The-Shelf), Software que existe, está disponível para o público e pode ser comprado ou licenciado, são integrados para criar o novo sistema. A integração de sistemas, nesse modelo, pode ser parte do processo de desenvolvimento, em vez de uma atividade separada.

Existem três tipos de componentes de software que podem ser usados em um processo orientado a reuso:

1. Web services desenvolvidos de acordo com os padrões de serviço e que estão disponíveis para invocação remota.
2. Coleções de objetos que são desenvolvidas como um pacote a ser integrado com um framework de componentes, como .NET ou J2EE.
3. Sistemas de software stand-alone configurados para uso em um ambiente particular.

Engenharia de software orientada a reuso tem a vantagem óbvia de reduzir a quantidade de software a ser desenvolvido e, assim, reduzir os custos e riscos. Geralmente, também proporciona a entrega mais rápida do software. No entanto, compromissos com os requisitos são inevitáveis, e isso pode levar a um sistema que não atende às reais necessidades dos usuários. Além disso, algum controle sobre a evolução do sistema é perdido, pois as novas versões dos componentes reusáveis não estão sob o controle da organização que os está utilizando.

Segue uma tabela comparativa das principais características que devem ser observadas antes de escolher o ciclo ou os ciclos de vida a serem adotados no projeto (Tabela 2).

Tabela 2 - Comparativo das características dos ciclos de vida apresentados

Modelo	Foco	Requisitos	1ª versão p/ cliente	Gerenciamento (1=mais simples)	Sistemas (tamanho/complexidade máximos)
Cascata	Documento e artefato	Bem conhecido/congelado	Fim do ciclo	1	Simple
V	Planejamento de testes	Bem conhecido/congelado	Fim do ciclo	2	Simple
Incremental	Incrementos operacionais	Maior abstração / Tratado em módulos	Protótipos operacionais	3	Médio
Evolutivo	Evolução dos requisitos	Pouco conhecidos	Protótipos operacionais	4	Médio
RAD	Rapidez	Escopo restrito / Maior abstração / Tratado em módulos	Protótipos operacionais	4	Médio
Prototipagem	Dúvidas nos Requisitos	Abstratos	Protótipos não operacionais	5	Médio
Espiral	Análise de risco	Maior abstração / evoluídos com o tempo	Protótipos operacionais ou não operacionais	5	Complexos
RUP	Frameworks e boas práticas	Maior abstração / evoluídos com o tempo	Protótipos operacionais ou não operacionais	5	Complexos

3. Atividades do Processo

Processo de software é como uma metodologia para as atividades, ações e tarefas necessárias para desenvolver softwares de alta qualidade. Dessa forma, um processo de software é como uma série de passos previsíveis, ou um roteiro, que ajudará na criação de um produto ou sistema de alta qualidade e dentro do prazo estabelecido entre as partes.

Um processo de software pode ser diferente dependendo da organização ou do projeto, ou seja, ele é adaptável às necessidades. Esse processo conta com a ajuda de toda a equipe de desenvolvimento, equipe de testes, gerentes, entre outros, além também dos próprios solicitantes do software que devem colaborar com a definição, construção e teste do software.

Processos de software são intercalados com sequências de atividades técnicas, de colaboração e de gerência, com o intuito de especificar, projetar, implementar e testar um sistema de software. As quatro atividades básicas do processo — especificação, desenvolvimento, validação e evolução — são organizadas de forma diferente conforme o processo de desenvolvimento.

No modelo em cascata são organizadas em sequência, enquanto no desenvolvimento incremental são intercaladas. A maneira como essas atividades serão feitas depende do tipo de software, das pessoas e das estruturas organizacionais envolvidas.

3.1. Especificação de Software

Especificação de software ou engenharia de requisitos é o processo de compreensão e definição dos serviços requisitados do sistema e identificação de restrições relativas à operação e ao desenvolvimento do sistema. A engenharia de requisitos é um estágio particularmente crítico do processo de software, pois erros nessa fase inevitavelmente geram problemas no projeto e na implementação do sistema Fig. 12).

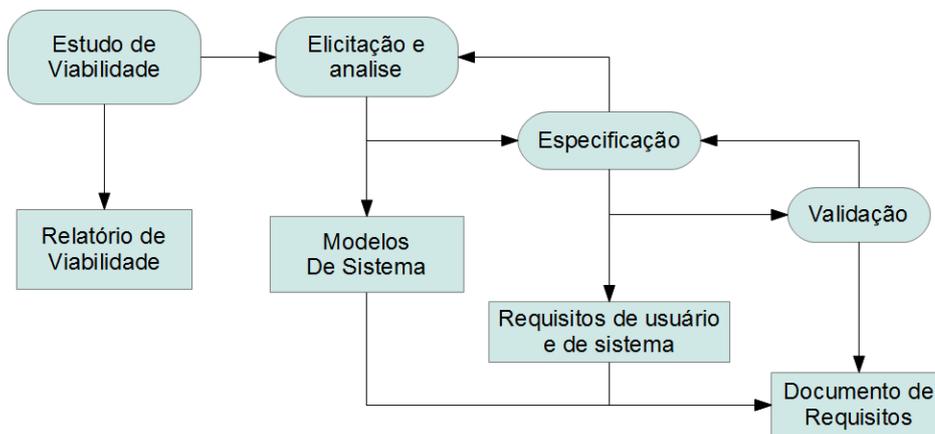


Figura 12 - Requisitos da engenharia de processos

O processo de engenharia de requisitos tem como objetivo produzir um documento de requisitos acordados que especifica um sistema que satisfaz os requisitos dos stakeholders (as partes interessadas e envolvidas com o projeto). Requisitos são geralmente apresentados em dois níveis de detalhe. Os usuários finais e os clientes precisam de uma declaração de requisitos em alto nível; desenvolvedores de sistemas precisam de uma especificação mais detalhada do sistema.

Existem quatro atividades principais do processo de engenharia de requisitos:

1. Estudo de viabilidade: É feita uma estimativa acerca da possibilidade de se satisfazerem as necessidades do usuário identificado usando-se tecnologias atuais de software e hardware. O estudo considera se o sistema proposto será rentável a partir de um ponto de vista de negócio e se ele pode ser desenvolvido no âmbito das atuais restrições orçamentais. Um estudo de viabilidade deve ser relativamente barato e rápido. O resultado deve informar a decisão de avançar ou não, com uma análise mais detalhada.
2. Elicitação (extração) e análise de requisitos: Esse é o processo de derivação dos requisitos do sistema por meio da observação dos sistemas existentes, além de discussões com os potenciais usuários e compradores, análise de tarefas, entre outras etapas. Essa parte do processo pode envolver o desenvolvimento de um ou mais modelos de sistemas e protótipos, os quais nos ajudam a entender o sistema a ser especificado.
3. Especificação de requisitos: É a atividade de traduzir as informações obtidas durante a atividade de análise em um documento que defina um conjunto de requisitos. Dois tipos de requisitos podem ser incluídos nesse documento. Requisitos do usuário são declarações abstratas dos requisitos do sistema para o cliente e usuário final do sistema; requisitos de sistema são uma descrição mais detalhada da funcionalidade a ser provida.

4. A validação de requisitos: Essa atividade verifica os requisitos quanto a realismo, consistência e completude. Durante esse processo, os erros no documento de requisitos são inevitavelmente descobertos. Em seguida, o documento deve ser modificado para correção desses problemas.

A análise de requisitos continua durante a definição e especificação, e novos requisitos emergem durante o processo. Portanto, as atividades de análise, definição e especificação são intercaladas. Nos métodos ágeis, os requisitos são desenvolvidos de forma incremental, de acordo com as prioridades do usuário, e a elicitación de requisitos é feita pelos usuários que integram equipe de desenvolvimento.

3.2. Projeto e Implementação de Software

O estágio de implementação do desenvolvimento de software é o processo de conversão de uma especificação do sistema em um sistema executável. Sempre envolve processos de projeto e programação de software, mas, se for usada uma abordagem incremental para o desenvolvimento, também pode envolver o refinamento da especificação do software.

Um projeto de software é uma descrição da estrutura do software a ser implementado, dos modelos e estruturas de dados usados pelo sistema, das interfaces entre os componentes do sistema e, às vezes, dos algoritmos usados. Os projetistas não chegam a um projeto final imediatamente, mas desenvolvem-no de forma iterativa. Eles acrescentam formalidade e detalhes, enquanto desenvolvem seu projeto por meio de revisões constantes para correção de projetos anteriores.

A maioria dos softwares interage com outros sistemas de software, incluindo o sistema operacional, o banco de dados, o middleware e outros aplicativos. Estes formam a 'plataforma de software', o ambiente em que o software será executado. Informações sobre essa plataforma são entradas para o processo de projeto, pois os projetistas devem decidir a melhor forma de integrá-la ao ambiente do software (Fig. 13).

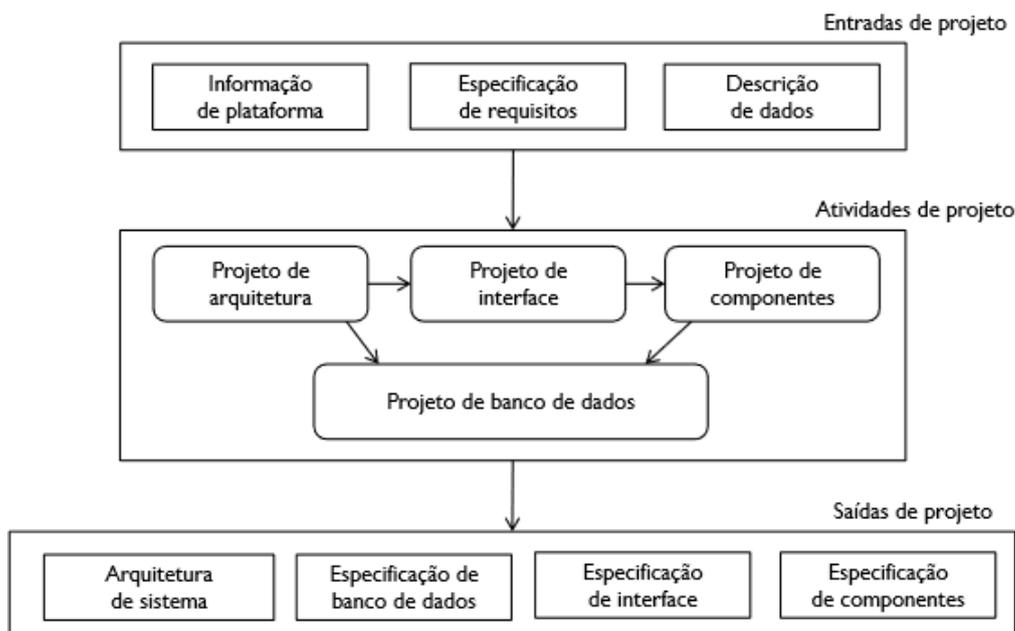


Figura 13 - Modelo geral do processo de projeto de software

A especificação de requisitos é uma descrição da funcionalidade que o software deve oferecer, e seus requisitos de desempenho e confiança. Se o sistema for para processamento de dados existentes, a descrição desses dados poderia ser incluída na especificação da plataforma. Caso contrário, a descrição dos dados deve ser uma entrada para o processo de projeto, para que a organização dos dados do sistema seja definida.

As atividades no processo de projeto podem variar, dependendo do tipo de sistema a ser desenvolvido. Por exemplo, sistemas de tempo real demandam projeto de timing, mas podem não incluir uma base de dados. Nesse caso, não há um projeto de banco de dados envolvido. O modelo anterior mostra quatro atividades que podem ser parte do processo de projeto de sistemas de informação:

1. Projeto de arquitetura, no qual você pode identificar a estrutura geral do sistema, os componentes principais (algumas vezes, chamados subsistemas ou módulos), seus relacionamentos e como eles são distribuídos.
2. Projeto de banco de dados, no qual você projeta as estruturas de dados do sistema e como eles devem ser representados em um banco de dados. Novamente, o trabalho aqui depende da existência de um banco de dados a ser reusado ou da criação de um novo banco de dados.
3. Projeto de interface, no qual você define as interfaces entre os componentes do sistema. Essa especificação de interface deve ser inequívoca. Com uma interface precisa, um componente pode ser usado de maneira que outros componentes não precisam saber como ele é implementado. Uma vez que as especificações de interface são acordadas, os componentes podem ser projetados e desenvolvidos simultaneamente.
4. Projeto de componentes, no qual você toma cada componente do sistema e projeta seu funcionamento. Pode-se tratar de uma simples declaração da funcionalidade que se espera implementar, com o projeto específico para cada programador. Pode, também, ser uma lista de alterações a serem feitas em um componente reusável ou um modelo de projeto detalhado. O modelo de projeto pode ser usado para gerar automaticamente uma implementação.

Métodos estruturados para projeto foram desenvolvidos nas décadas de 1970 e 1980, e foram os precursores da UML e do projeto orientado a objetos. Eles estão relacionados com a produção de modelos gráficos do sistema e, em muitos casos, geram códigos automaticamente a partir desses modelos. Desenvolvimento dirigido a modelos – MDD (Model-Driven Development) ou engenharia dirigida a modelos em que os modelos de software são criados em diferentes níveis de abstração, é uma evolução dos métodos estruturados. Em MDD, há uma ênfase maior nos modelos de arquitetura com uma separação entre os modelos abstratos independentes de implementação e específicos de implementação. Os modelos são desenvolvidos em detalhes suficientes para que o sistema executável possa ser gerado a partir deles.

Ferramentas de desenvolvimento de software podem ser usadas para gerar um esqueleto de um programa a partir do projeto. Isso inclui o código para definir e implementar interfaces e, em muitos casos, o desenvolvedor precisa apenas acrescentar detalhes da operação de cada componente do programa.

3.3. Validação de Software

Validação de software ou, mais genericamente, verificação e validação (V&V), tem a intenção de mostrar que um software se adequa a suas especificações ao mesmo tempo que satisfaz as especificações do cliente do sistema. Teste de programa, em que o sistema é executado com dados de testes simulados, é a principal técnica de validação. A validação também pode envolver processos de verificação, como inspeções e revisões, em cada estágio do processo de software, desde a definição dos requisitos de usuários até o desenvolvimento do programa. Devido à predominância dos testes, a maior parte dos custos de validação incorre durante e após a implementação.

Idealmente, os defeitos de componentes são descobertos no início do processo, e os problemas de interface são encontrados quando o sistema é integrado. No entanto, quando os defeitos são descobertos, o programa deve ser depurado, e isso pode requerer que outros estágios do processo de testes sejam repetidos. Erros em componentes de programa podem vir à luz durante os testes de sistema. O processo é, portanto, iterativo, com informações realimentadas de estágios posteriores para partes anteriores do processo. Destacam-se os seguintes estágios:

1. Testes de desenvolvimento - Os componentes do sistema são testados pelas pessoas que o desenvolveram. Cada componente é testado de forma independente, separado dos outros. Os componentes podem ser entidades simples, como funções ou classes de objetos, ou podem ser agrupamentos coerentes dessas entidades. Ferramentas de automação de teste que podem reexecutar testes de componentes quando as novas versões dos componentes são criadas, são comumente usadas.
2. Testes de sistema - Componentes do sistema são integrados para criar um sistema completo. Esse processo se preocupa em encontrar os erros resultantes das interações inesperadas entre componentes e problemas de interface do componente. Também visa mostrar que o sistema satisfaz seus requisitos funcionais e não funcionais, bem como testar as propriedades emergentes do sistema. Para sistemas de grande porte, esse pode ser um processo multi-estágios, no qual os componentes são integrados para formar subsistemas individualmente testados antes de serem integrados para formar o sistema final.
3. Testes de aceitação - Esse é o estágio final do processo de testes, antes que o sistema seja aceito para uso operacional. O sistema é testado com dados fornecidos pelo cliente, e não com dados advindos de testes simulados. O teste de aceitação pode revelar erros e omissões na definição dos requisitos do sistema, pois dados reais exercitam o sistema de formas diferentes dos dados de teste. Os testes de aceitação também podem revelar problemas de requisitos em que os recursos do sistema não atendam às necessidades do usuário ou o desempenho do sistema seja inaceitável.

Quando um processo de software dirigido a planos é usado (por exemplo, para o desenvolvimento de sistemas críticos), o teste é impulsionado por um conjunto de planos de testes. Uma equipe trabalha a partir desses planos de teste pré-formulados, que foram desenvolvidos a partir das especificações e do projeto do sistema. A Figura 14, ilustra como os planos de teste são o elo entre as atividades de teste e de desenvolvimento. Esse modelo é, às vezes, chamado modelo V de desenvolvimento.

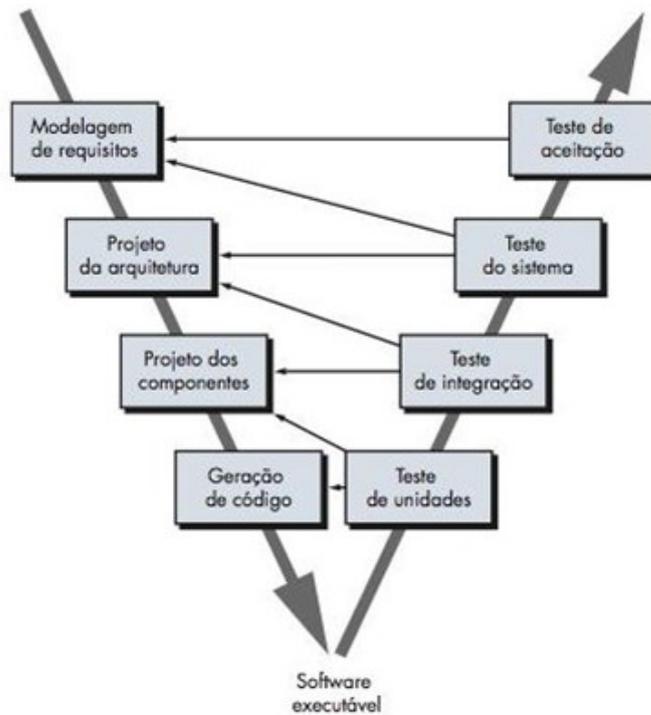


Figura 14 - Fases de testes de um processo de software dirigido a planos

O teste de aceitação também pode ser chamado 'teste alfa'. Sistemas sob encomenda são desenvolvidos para um único cliente. O processo de testes-alfa continua até que o desenvolvedor do sistema e o cliente concordem que o sistema entregue é uma implementação aceitável dos requisitos.

Quando um sistema está pronto para ser comercializado como um produto de software, costuma-se usar um processo de testes denominado 'teste beta'. Este teste envolve a entrega de um sistema a um número de potenciais clientes que concordaram em usá-lo. Eles relatam problemas para os desenvolvedores dos sistemas. O produto é exposto para uso real, e erros que podem não ter sido antecipados pelos construtores do sistema são detectados. Após esse feedback, o sistema é modificado e liberado para outros testes-beta ou para venda em geral.