ACRÔNIMOS

3GL Linguagem de Terceira Geração

BNF Forma de Backus-Naur

FDD Desenvolvimento Orientado a Funcionalidades

IDE Ambiente de Desenvolvimento Integrado

PBI Item do Backlog do Produto

RAD Desenvolvimento Rápido de Aplicações

UML Linguagem de Modelagem Unificada

XP Programação Extrema

MODELOS E MÉTODOS DE ENGENHARIA DE SOFTWARE

INTRODUÇÃO

Modelos e métodos de engenharia de software impõem estrutura à engenharia de software com o objetivo de tornar essa atividade sistemática, repetível e, em última análise, mais orientada para o sucesso. O uso de modelos fornece uma abordagem para a resolução de problemas, uma notação e procedimentos para construção e análise de modelos. Métodos fornecem uma abordagem para a especificação sistemática, design, construção, teste e verificação do software final e dos produtos de trabalho associados. Modelos e métodos de engenharia de software variam amplamente em escopo - desde abordar uma única fase do ciclo de vida do software até cobrir o ciclo de vida completo do software. O foco desta área de conhecimento (AC) está nos modelos e métodos de engenharia de software que abrangem múltiplas fases do ciclo de vida do software, uma vez que métodos específicos para fases individuais do ciclo de vida são abordados por outras ACs.

DIVISÃO DE TÓPICOS PARA MODELOS E MÉTODOS DE ENGENHARIA DE SOFTWARE

Este capítulo sobre modelos e métodos de engenharia de software é dividido em quatro áreas principais de tópicos:

- Modelagem: discute a prática geral de modelagem e apresenta tópicos em princípios de modelagem; propriedades e expressão de modelos; sintaxe, semântica e pragmática de modelagem; e precondições, pós-condições e invariantes.
- Tipos de Modelos: discute brevemente modelos e agregação de submodelos e fornece algumas características gerais de tipos de modelo comumente encontrados na prática de engenharia de software.
- Análise de Modelos: apresenta algumas das técnicas de análise comuns usadas na modelagem para verificar completude, consistência, correção, rastreabilidade e interação.

- Métodos de Engenharia de Software: apresenta um resumo breve dos métodos de engenharia de software comumente usados. A discussão guia o leitor por um resumo de métodos heurísticos, métodos formais, prototipagem e métodos ágeis.

A divisão de tópicos para a AC de Modelos e Métodos de Engenharia de Software é mostrada na Figura 9.1.

1. Modelagem

A modelagem de software está se tornando uma técnica difundida para ajudar os engenheiros de software a entender, projetar e comunicar aspectos do software aos interessados apropriados. Os interessados são aquelas pessoas ou partes que têm um interesse declarado ou implícito no software (por exemplo, usuário, comprador, fornecedor, arquiteto, autoridade certificadora, avaliador, desenvolvedor, engenheiro de software e talvez outros).

Embora existam muitas linguagens, notações, técnicas e ferramentas de modelagem na literatura e na prática, existem conceitos gerais unificadores que se aplicam de alguma forma a todas elas. As seções a seguir fornecem informações sobre esses conceitos gerais.

1.1. Princípios de Modelagem

[1*, c2s2, c5s1, c5s2] [2*, c2s2] [3*, c5s0]

A modelagem fornece ao engenheiro de software uma abordagem organizada e sistemática para representar aspectos significativos do software em estudo, facilitando a tomada de decisões sobre o software ou elementos dele e comunicando essas decisões significativas a outros nas comunidades interessadas. Existem três princípios gerais que orientam essas atividades de modelagem:

- Modelar o Essencial: bons modelos geralmente não representam todos os aspectos ou características do software em todas as condições possíveis. A modelagem geralmente envolve o desenvolvimento apenas dos aspectos ou características do software que precisam de respostas específicas, abstraindo qualquer informação não essencial. Esta abordagem mantém os modelos gerenciáveis e úteis.
- Fornecer Perspectiva: a modelagem fornece visões do software em estudo usando um conjunto definido de regras para expressão do modelo dentro de cada visão. Esta abordagem orientada pela perspectiva fornece dimensionalidade ao modelo (por exemplo, uma visão estrutural, uma visão comportamental, uma visão temporal, uma visão organizacional e outras visões relevantes). Organizar informações em visões concentra os esforços de modelagem de software em preocupações específicas relevantes para essa visão usando a notação, vocabulário, métodos e ferramentas apropriados.
- Possibilitar Comunicações Efetivas: a modelagem emprega o vocabulário do domínio de aplicação do software, uma linguagem de modelagem e expressão semântica (ou seja, significado dentro do contexto). Quando usada rigorosa e sistematicamente, esta modelagem resulta em uma abordagem de relatório que facilita a comunicação eficaz das informações do software aos interessados no projeto.

Um modelo é uma abstração ou simplificação de um componente de software. Uma consequência do uso de abstração é que nenhuma abstração única descreve completamente um componente de software. Em vez disso, o modelo do software é representado como uma agregação de abstrações, que - quando combinadas - descrevem apenas aspectos, perspectivas ou visões selecionados - apenas aqueles que são necessários para tomar decisões informadas e responder aos motivos de criação do modelo em primeiro lugar. Esta simplificação leva a um conjunto de pressupostos sobre o contexto no qual o modelo é colocado, que também devem ser capturados no modelo. Em seguida, ao reutilizar o modelo, esses pressupostos podem ser validados primeiro para estabelecer a relevância do modelo reutilizado dentro de seu novo uso e contexto.

1.2. Propriedades e Expressão de Modelos

As propriedades dos modelos são aquelas características distintivas de um modelo específico usadas para caracterizar sua completude, consistência e correção dentro da notação de modelagem e ferramentas utilizadas. As propriedades dos modelos incluem o seguinte:

- Completude: o grau em que todos os requisitos foram implementados e verificados dentro do modelo.
- Consistência: o grau em que o modelo não contém requisitos, assertivas, restrições, funções ou descrições de componentes conflitantes.
- Correção: o grau em que o modelo satisfaz seus requisitos e especificações de design e está livre de defeitos.

Os modelos são construídos para representar objetos do mundo real e seus comportamentos para responder a questões específicas sobre como o software deve operar. A interrogação dos modelos - seja por exploração, simulação ou revisão - pode expor áreas de incerteza dentro do modelo e do software ao qual o modelo se refere. Essas incertezas ou perguntas não respondidas sobre os requisitos, design e/ou implementação podem então ser tratadas adequadamente.

O principal elemento de expressão de um modelo é uma entidade. Uma entidade pode representar artefatos concretos (por exemplo, processadores, sensores ou robôs) ou artefatos abstratos (por exemplo, módulos de software ou protocolos de comunicação). As entidades do modelo são conectadas a outras entidades usando relações (ou seja, linhas ou operadores textuais em entidades de destino). A expressão das entidades do modelo pode ser realizada usando linguagens de modelagem textuais ou gráficas; ambos os tipos de linguagem de modelagem conectam entidades do modelo por meio de construções de linguagem específicas. O significado de uma entidade pode ser representado por sua forma, atributos textuais ou ambos. Geralmente, as informações textuais aderem à estrutura

sintática específica da linguagem. Os significados precisos relacionados à modelagem de contexto, estrutura ou comportamento usando essas entidades e relações dependem da linguagem de modelagem utilizada, do rigor de design aplicado ao esforço de modelagem, da visualização específica sendo construída e da entidade à qual o elemento de notação específico pode ser anexado. Múltiplas visualizações do modelo podem ser necessárias para capturar a semântica necessária do software.

Ao utilizar modelos com suporte de automação, os modelos podem ser verificados quanto à completude e consistência. A utilidade dessas verificações depende muito do nível de rigor semântico e sintático aplicado ao esforço de modelagem, além do suporte explícito da ferramenta. A correção geralmente é verificada por meio de simulação e/ou revisão.

1.3. Sintaxe, Semântica e Pragmatismo

Os modelos podem ser surpreendentemente enganosos. O fato de um modelo ser uma abstração com informações faltantes pode levar alguém a uma falsa sensação de entendimento completo do software a partir de um único modelo. Um modelo completo ("completo" sendo relativo ao esforço de modelagem) pode ser uma união de múltiplos submodelos e quaisquer modelos de funções especiais. A análise e tomada de decisão relativas a um único modelo dentro dessa coleção de submodelos pode ser problemática.

Compreender os significados precisos das construções de modelagem também pode ser difícil. As linguagens de modelagem são definidas por regras sintáticas e semânticas. Para linguagens textuais, a sintaxe é definida usando uma gramática de notação que define construções de linguagem válidas (por exemplo, Forma de Backus-Naur (BNF)). Para linguagens gráficas, a sintaxe é definida usando modelos gráficos chamados metamodelos. Assim como a BNF, os metamodelos definem as construções sintáticas válidas de uma linguagem de modelagem gráfica; o metamodelo define como essas construções podem ser compostas para produzir modelos válidos.

A semântica para linguagens de modelagem especifica o significado atribuído às entidades e relações capturadas dentro do modelo. Por exemplo, um diagrama simples de duas caixas conectadas por uma linha está aberto a uma variedade de interpretações. Saber que o diagrama no qual as caixas estão colocadas e conectadas é um diagrama de objeto ou um diagrama de atividade pode ajudar na interpretação desse modelo.

Como questão prática, geralmente há um bom entendimento da semântica de um modelo de software específico devido à linguagem de modelagem selecionada, como essa linguagem de modelagem é usada para expressar entidades e relações dentro desse modelo, à base de experiência do(s)

modelador(es) e ao contexto dentro do qual a modelagem foi realizada e representada. O significado é comunicado através do modelo mesmo na presença de informações incompletas por meio da abstração; a pragmática explica como o significado é incorporado ao modelo e seu contexto e comunicado efetivamente a outros engenheiros de software.

Ainda existem casos, no entanto, em que é necessária cautela em relação à modelagem e semântica. Por exemplo, partes do modelo importadas de outro modelo ou biblioteca devem ser examinadas quanto a suposições semânticas que conflitam no novo ambiente de modelagem; isso pode não ser óbvio. O modelo deve ser verificado quanto a suposições documentadas. Embora a sintaxe de modelagem possa ser idêntica, o modelo pode significar algo completamente diferente no novo ambiente, que é um contexto diferente. Além disso, considere que, à medida que o software amadurece e são feitas alterações, pode ser introduzido um desacordo semântico, levando a erros. Com muitos engenheiros de software trabalhando em uma parte do modelo ao longo do tempo, juntamente com atualizações de ferramentas e talvez novos requisitos, existem oportunidades para que partes do modelo representem algo diferente da intenção original do autor e do contexto inicial do modelo.

1.4. Pré-condições, Pós-condições e Invariáveis

Ao modelar funções ou métodos, o engenheiro de software geralmente começa com um conjunto de pressupostos sobre o estado do software antes, durante e após a execução da função ou método. Esses pressupostos são essenciais para a operação correta da função ou método e são agrupados, para discussão, como um conjunto de pré-condições, pós-condições e invariantes.

- Pré-condições: um conjunto de condições que devem ser satisfeitas antes da execução da função ou método. Se essas pré-condições não forem atendidas antes da execução da função ou método, a função ou método pode produzir resultados errôneos.
- Pós-condições: um conjunto de condições que é garantido de ser verdadeiro após a execução bem-sucedida da função ou método. Tipicamente, as pós-condições representam como o estado do software mudou, como os parâmetros passados para a função ou método mudaram, como os valores dos dados mudaram ou como o valor de retorno foi afetado.
- Invariáveis: um conjunto de condições dentro do ambiente operacional que persistem (ou seja, não mudam) antes e após a execução da função ou método. Essas invariáveis são relevantes e necessárias para o software e para a operação correta da função ou método.

2. Tipos de Modelos

Um modelo típico consiste em uma agregação de submodelos. Cada submodelo é uma descrição parcial e é criado para um propósito específico; ele pode ser composto por um ou mais diagramas. A coleção de submodelos pode empregar múltiplas linguagens de modelagem de software ou uma única linguagem de modelagem. A Linguagem de Modelagem Unificada (UML) reconhece uma rica coleção de diagramas de modelagem. O uso desses diagramas, juntamente com as construções da linguagem de modelagem, resulta em três tipos amplos de modelos comumente usados: modelos de informação, modelos comportamentais e modelos de estrutura (veja a seção 1.1).

2.1. Modelagem de Informação

Os modelos de informação têm um foco central em dados e informações. Um modelo de informação é uma representação abstrata que identifica e define um conjunto de conceitos, propriedades, relações e restrições em entidades de dados. O modelo de informação semântico ou conceitual é frequentemente usado para fornecer algum formalismo e contexto ao software que está sendo modelado, visto da perspectiva do problema, sem se preocupar com como esse modelo é realmente mapeado para a implementação do software. O modelo de informação semântico ou conceitual é uma abstração e, como tal, inclui apenas os conceitos, propriedades, relações e restrições necessários para conceituar a visão do mundo real das informações. Transformações subsequentes do modelo de informação semântico ou conceitual levam à elaboração de modelos de dados lógicos e físicos, conforme implementados no software.

2.2. Modelagem Comportamental

Os modelos comportamentais identificam e definem as funções do software que está sendo modelado. Os modelos comportamentais geralmente têm três formas básicas: máquinas de estados, modelos de fluxo de controle e modelos de fluxo de dados. As máquinas de estados fornecem um modelo do software como uma coleção de estados definidos, eventos e transições. O software transita de um estado para o próximo por meio de um evento de acionamento guardado ou não guardado que ocorre no ambiente modelado. Os modelos de fluxo de controle mostram como uma sequência de eventos faz com que processos sejam ativados ou desativados. O comportamento de fluxo de dados é tipificado como uma sequência de etapas onde os dados se movem por processos em direção a repositórios de dados ou destinos de dados.

2.3. Modelagem de Estrutura

Os modelos de estrutura ilustram a composição física ou lógica do software a partir de suas várias partes componentes. A modelagem de estrutura estabelece a fronteira definida entre o software que está sendo implementado ou modelado e o ambiente no qual ele deve operar. Alguns construtos estruturais comuns usados na modelagem de estrutura são composição, decomposição, generalização e especialização de entidades; identificação de relações relevantes e cardinalidade entre entidades; e a definição de interfaces de processo ou funcional. Os diagramas de estrutura fornecidos pela UML para modelagem de estrutura incluem diagramas de classes, componentes, objetos, implantação e empacotamento.

3. Análise de Modelos

O desenvolvimento de modelos proporciona ao engenheiro de software a oportunidade de estudar, raciocinar e entender a estrutura, função, uso operacional e considerações de montagem associadas ao software. A análise dos modelos construídos é necessária para garantir que esses modelos sejam completos, consistentes e corretos o suficiente para servir ao seu propósito pretendido para as partes interessadas.

As seções a seguir descrevem brevemente as técnicas de análise geralmente usadas com modelos de software para garantir que o engenheiro de software e outras partes interessadas relevantes obtenham o valor apropriado do desenvolvimento e uso de modelos.

3.1. Análise de integridade

Para ter um software que atenda plenamente às necessidades das partes interessadas, a completude é fundamental - desde o processo de elicitação de requisitos até a implementação do código. Completeness é o grau em que todos os requisitos especificados foram implementados e verificados. Os modelos podem ser verificados quanto à completude por uma ferramenta de modelagem que utiliza técnicas como análise estrutural e análise de alcançabilidade do espaço de estados (que garantem que todos os caminhos nos modelos de estado sejam alcançados por algum conjunto de entradas corretas); os modelos também podem ser verificados quanto à completude manualmente usando inspeções ou outras técnicas de revisão (consulte a AC de Qualidade de Software). Erros e avisos gerados por essas ferramentas de análise e encontrados por inspeção ou revisão indicam ações corretivas provavelmente necessárias para garantir a completude dos modelos.

3.2. Análise de Consistência

Consistência é o grau em que os modelos não contêm requisitos, assertivas, restrições, funções ou descrições de componentes conflitantes. Tipicamente, a verificação de consistência é realizada com a ferramenta de modelagem usando uma função de análise automatizada; os modelos também podem ser verificados quanto à consistência manualmente usando inspeções ou outras técnicas de revisão (consulte a AC de Qualidade de Software). Assim como com a completude, erros e avisos gerados por essas ferramentas de análise e encontrados por inspeção ou revisão indicam a necessidade de ação corretiva.

3.3. Análise de Correção

Correção é o grau em que um modelo satisfaz seus requisitos de software e especificações de design de software, é livre de defeitos e atende às necessidades das partes interessadas. A análise de correção inclui verificar a correção sintática do modelo (ou seja, o uso correto da gramática e construções da linguagem de modelagem) e verificar a correção semântica do modelo (ou seja, o uso das construções da linguagem de modelagem para representar corretamente o significado do que está sendo modelado). Para analisar um modelo quanto à correção sintática e semântica, analisa-se-o - automaticamente (por exemplo, usando a ferramenta de modelagem para verificar a correção sintática do modelo) ou manualmente (usando inspeções ou outras técnicas de revisão) - procurando defeitos possíveis e, em seguida, removendo ou reparando os defeitos confirmados antes que o software seja liberado para uso.

3.4. Traçabilidade

O desenvolvimento de software geralmente envolve o uso, criação e modificação de muitos produtos de trabalho, como documentos de planejamento, especificações de processo, requisitos de software, diagramas, projetos e pseudo-código, código gerado manualmente e por ferramentas, casos e relatórios de teste manuais e automatizados e arquivos e dados. Esses produtos de trabalho podem estar relacionados por meio de vários relacionamentos de dependência (por exemplo, usa, implementa e testa). À medida que o software está sendo desenvolvido, gerenciado, mantido ou estendido, há uma necessidade de mapear e controlar esses relacionamentos de traçabilidade para demonstrar a consistência dos requisitos de software com o modelo de software (consulte o Rastreamento de Requisitos na AC de Requisitos de Software) e os muitos produtos de trabalho. O uso de traçabilidade geralmente melhora o gerenciamento dos produtos de trabalho de software e a qualidade do processo de software; também fornece garantias às partes interessadas de que todos os requisitos foram satisfeitos. A traçabilidade permite a análise de mudanças uma vez que o software é desenvolvido e liberado, uma vez que os relacionamentos com os produtos de trabalho de software podem ser facilmente percorridos para avaliar o impacto das mudanças. As ferramentas de modelagem

geralmente fornecem algum meio automatizado ou manual para especificar e gerenciar links de traçabilidade entre requisitos, design, código e/ou entidades de teste, conforme representado nos modelos e em outros produtos de trabalho de software. (Para mais informações sobre traçabilidade, consulte a AC de Gerenciamento de Configuração de Software).

3.5. Análise de Interação

A análise de interação se concentra nas relações de comunicação ou fluxo de controle entre entidades usadas para realizar uma tarefa ou função específica dentro do modelo de software. Esta análise examina o comportamento dinâmico das interações entre diferentes partes do modelo de software, incluindo outras camadas de software (como o sistema operacional, middleware e aplicativos). Também pode ser importante para algumas aplicações de software examinar as interações entre o aplicativo de software do computador e o software de interface do usuário. Alguns ambientes de modelagem de software fornecem facilidades de simulação para estudar aspectos do comportamento dinâmico do software modelado. Percorrer a simulação fornece uma opção de análise para o engenheiro de software revisar o design de interação e verificar se as diferentes partes do software funcionam juntas para fornecer as funções pretendidas.

4. Métodos de Engenharia de Software

Os métodos de engenharia de software fornecem uma abordagem organizada e sistemática para desenvolver software para um computador-alvo. Existem inúmeros métodos entre os quais escolher, e é importante para o engenheiro de software escolher um método ou métodos apropriados para a tarefa de desenvolvimento de software em questão; essa escolha pode ter um efeito dramático no sucesso do projeto de software. O uso desses métodos de engenharia de software, juntamente com pessoas com o conjunto de habilidades adequado e ferramentas, permite aos engenheiros de software visualizar os detalhes do software e, por fim, transformar a representação em um conjunto de código e dados funcionais.

Abaixo são discutidos métodos selecionados de engenharia de software. As áreas de tópicos estão organizadas em discussões de Métodos Heurísticos, Métodos Formais, Métodos de Prototipagem e Métodos Ágeis.

4.1. Métodos Heurísticos

Métodos heurísticos são métodos de engenharia de software baseados em experiência que foram e são amplamente praticados na indústria de software. Esta área de tópicos contém três amplas categorias de discussão: métodos de análise e design estruturados, métodos de modelagem de dados e métodos de análise e design orientados a objetos.

- Métodos de Análise e Design Estruturados:
O modelo de software é desenvolvido principalmente a partir de um ponto de vista funcional ou comportamental, partindo de uma visão de alto nível do software (incluindo elementos de dados e controle) e, em seguida, decompondo ou refinando progressivamente os componentes do modelo por meio de designs cada vez mais detalhados. O design detalhado eventualmente converge para detalhes ou especificações muito específicas do software que devem ser codificados (manualmente, gerados automaticamente ou ambos), construídos, testados e verificados.

- Métodos de Modelagem de Dados:
O modelo de dados é construído a partir do ponto de vista dos dados ou informações utilizadas. Tabelas de dados e relacionamentos definem os modelos de dados. Este método de modelagem de dados é usado principalmente para definir e analisar os requisitos de dados que suportam designs de banco de dados ou repositórios de dados normalmente encontrados em software empresarial, onde os dados são gerenciados ativamente como um recurso ou ativo dos sistemas de negócios.

- Métodos de Análise e Design Orientados a Objetos:
O modelo orientado a objetos é representado como uma coleção de objetos que encapsulam dados e relacionamentos e interagem com outros objetos por meio de métodos. Os objetos podem ser itens do mundo real ou virtuais. O modelo de software é construído usando diagramas para constituir visões selecionadas do software. O refinamento progressivo dos modelos de software leva a um design detalhado. O design detalhado é então evoluído por meio de iteração sucessiva ou transformado (usando algum mecanismo) na visão de implementação do modelo, onde o código e a abordagem de empacotamento para o eventual lançamento e implantação do produto de software são expressos.

4.2. Métodos Formais

Métodos formais são métodos de engenharia de software usados para especificar, desenvolver e verificar o software por meio da aplicação de uma notação e linguagem rigorosamente baseadas em matemática. Através do uso de uma linguagem de especificação, o modelo de software pode ser verificado quanto à consistência (ou seja, falta de ambiguidade), completude e correção de maneira

sistemática e automatizada ou semi-automatizada. Este tópico está relacionado à seção de Análise Formal na KA de Requisitos de Software.

Esta seção aborda linguagens de especificação, refinamento e derivação de programas, verificação formal e inferência lógica.

- Linguagens de Especificação: As linguagens de especificação fornecem a base matemática para um método formal; são linguagens de computador formais e de nível superior (ou seja, não uma linguagem de programação clássica de Terceira Geração (3GL)) usadas durante a especificação de software, análise de requisitos e/ou estágios de design para descrever comportamentos específicos de entrada/saída. As linguagens de especificação não são linguagens diretamente executáveis; elas geralmente consistem em uma notação e sintaxe, semântica para uso da notação e um conjunto de relações permitidas para objetos.

- Refinamento e Derivação de Programas: O refinamento de programas é o processo de criação de uma especificação de nível inferior (ou mais detalhada) usando uma série de transformações. É por meio de transformações sucessivas que o engenheiro de software deriva uma representação executável de um programa. As especificações podem ser refinadas, adicionando detalhes até que o modelo possa ser formulado em uma linguagem de programação 3GL ou em uma porção executável da linguagem de especificação escolhida. Este refinamento de especificação é possibilitado pela definição de especificações com propriedades semânticas precisas; as especificações devem definir não apenas as relações entre entidades, mas também os significados exatos em tempo de execução dessas relações e operações.

- Verificação Formal: A verificação de modelos é um método formal de verificação; envolve tipicamente realizar uma exploração do espaço de estados ou análise de alcançabilidade para demonstrar que o design de software representado possui ou preserva certas propriedades de modelo de interesse. Um exemplo de verificação de modelo é uma análise que verifica o comportamento correto do programa sob todas as possíveis intercalações de eventos ou chegadas de mensagens. O uso de verificação formal requer um modelo rigorosamente especificado do software e seu ambiente operacional; este modelo frequentemente assume a forma de uma máquina de estado finito ou outro autômato formalmente definido.

- Inferência Lógica: A inferência lógica é um método de design de software que envolve especificar pré-condições e pós-condições em torno de cada bloco significativo do design e - usando lógica matemática - desenvolver a prova de que essas pré-condições e pós-condições devem ser mantidas sob todas as entradas. Isso fornece uma maneira para o engenheiro de software prever o comportamento

do software sem precisar executá-lo. Alguns Ambientes de Desenvolvimento Integrado (IDEs) incluem formas de representar essas provas junto com o design ou código.

4.3. Métodos de Prototipagem

A prototipagem de software é uma atividade que geralmente cria versões incompletas ou minimamente funcionais de um aplicativo de software, geralmente para experimentar novos recursos específicos, obter feedback sobre requisitos de software ou interfaces de usuário, explorar mais a fundo requisitos de software, design de software ou opções de implementação e/ou obter algum outro insight útil sobre o software. O engenheiro de software seleciona um método de prototipagem para entender os aspectos ou componentes menos compreendidos do software primeiro; esta abordagem é em contraste com outros métodos de engenharia de software que geralmente iniciam o desenvolvimento com as porções mais compreendidas primeiro. Tipicamente, o produto prototipado não se torna o produto final de software sem extenso retrabalho ou refatoração de desenvolvimento.

Esta seção discute estilos de prototipagem, alvos e técnicas de avaliação de forma breve.

- Estilo de Prototipagem: Isso aborda as várias abordagens para desenvolver protótipos. Protótipos podem ser desenvolvidos como código descartável ou produtos de papel, como uma evolução de um design de trabalho ou como uma especificação executável. Diferentes processos de ciclo de vida de prototipagem são tipicamente usados para cada estilo. O estilo escolhido é baseado no tipo de resultados que o projeto precisa, na qualidade dos resultados necessários e na urgência dos resultados.

- Alvo de Prototipagem: O alvo da atividade de prototipagem é o produto específico atendido pelo esforço de prototipagem. Exemplos de alvos de prototipagem incluem uma especificação de requisitos, um elemento ou componente de design arquitetônico, um algoritmo ou uma interface usuário-máquina.

- Técnicas de Avaliação de Prototipagem: Um protótipo pode ser utilizado ou avaliado de várias maneiras pelo engenheiro de software ou outros stakeholders do projeto, principalmente impulsionado pelos motivos subjacentes que levaram ao desenvolvimento do protótipo em primeiro lugar. Protótipos podem ser avaliados ou testados em relação ao software realmente implementado ou em relação a um conjunto alvo de requisitos (por exemplo, um protótipo de requisitos); o protótipo também pode servir como modelo para um esforço futuro de desenvolvimento de software (por exemplo, como em uma especificação de interface do usuário).

4.4 Métodos Ágeis

Os métodos ágeis surgiram na década de 1990 da necessidade de reduzir o grande overhead aparente associado aos métodos pesados e baseados em planos usados em projetos de desenvolvimento de software em grande escala. Os métodos ágeis são considerados métodos leves, caracterizados por ciclos de desenvolvimento curtos e iterativos, equipes auto-organizadas, designs mais simples, refatoração de código, desenvolvimento orientado a testes, envolvimento frequente do cliente e ênfase na criação de um produto funcional demonstrável a cada ciclo de desenvolvimento.

Muitos métodos ágeis estão disponíveis na literatura; alguns dos abordados aqui de forma resumida incluem Desenvolvimento Rápido de Aplicações (RAD), Programação Extrema (XP), Scrum e Desenvolvimento Dirigido por Funcionalidades (FDD).

- RAD: Métodos rápidos de desenvolvimento de software são usados principalmente em desenvolvimento de aplicativos de sistemas de informação intensivos em dados. O método RAD é viabilizado com ferramentas de desenvolvimento de banco de dados específicas usadas por engenheiros de software para desenvolver, testar e implantar rapidamente aplicativos empresariais novos ou modificados.

- XP: Esta abordagem usa histórias ou cenários para requisitos, desenvolve testes primeiro, tem envolvimento direto do cliente na equipe (tipicamente definindo testes de aceitação), utiliza programação em par, e prevê refatoração e integração contínuas de código. Histórias são decompostas em tarefas, priorizadas, estimadas, desenvolvidas e testadas. Cada incremento de software é testado com testes automatizados e manuais; um incremento pode ser lançado frequentemente, como a cada duas semanas aproximadamente.

- Scrum: Esta abordagem ágil é mais amigável ao gerenciamento de projetos do que as outras. O scrum master gerencia as atividades dentro do incremento do projeto; cada incremento é chamado de sprint e dura no máximo 30 dias. Uma lista de Itens do Backlog do Produto (PBI) é desenvolvida, a partir da qual as tarefas são identificadas, definidas, priorizadas e estimadas. Uma versão funcional do software é testada e lançada em cada incremento. Reuniões diárias do scrum garantem que o trabalho seja gerenciado conforme o planejado.

- FDD: Este é um método de desenvolvimento de software curto e iterativo, orientado a modelo, usando um processo de cinco fases: (1) desenvolver um modelo de produto para delimitar a amplitude do domínio, (2) criar a lista de necessidades ou funcionalidades, (3) construir o plano de desenvolvimento de funcionalidades, (4) desenvolver projetos para funcionalidades específicas da iteração e (5) codificar, testar e depois integrar as funcionalidades. FDD é semelhante a uma

abordagem incremental de desenvolvimento de software; também é semelhante ao XP, exceto que a propriedade do código é atribuída a indivíduos em vez da equipe. FDD enfatiza uma abordagem arquitetural geral para o software, que promove a construção correta da funcionalidade na primeira vez, em vez de enfatizar a refatoração contínua.

Há muitas variações de métodos ágeis na literatura e na prática. Note que sempre haverá espaço para métodos de engenharia de software baseados em planos, assim como lugares onde os métodos ágeis se destacam. Há novos métodos surgindo a partir de combinações de métodos ágeis e baseados em planos, onde os praticantes estão definindo novos métodos que equilibram as características necessárias em ambos os métodos, baseados principalmente nas necessidades organizacionais prevalecentes. Essas necessidades de negócios, conforme tipicamente representadas por alguns dos stakeholders do projeto, devem e influenciam a escolha no uso de um método de engenharia de software sobre outro ou na construção de um novo método a partir das melhores características de uma combinação de métodos de engenharia de software.