1

UML - Introdução

Modelagem II

Modelagem II

Conceitos de orientação a objetos e UML

UML - Introdução

A modelagem é uma das principais atividades que levam à implementação de um bom software. Construímos modelos para comunicar a estrutura e o comportamento desejados do sistema, visualizar e controlar a arquitetura do mesmo e compreender melhor o sistema que estamos elaborando.

A modelagem de software utiliza vários modelos para projetar um determinado sistema. Um modelo é uma simplificação da realidade, criado para facilitar o entendimento de sistemas complexos. Estes modelos podem abranger planos detalhados, assim como planos mais gerais com uma visão panorâmica do sistema.

Modelagem II

Conceitos de orientação a objetos e UML

UML - Introdução

Todos os sistemas podem ser descritos sob diferentes aspectos, com a utilização de modelos distintos, onde cada modelo será, portanto, uma abstração específica do sistema. Os modelos podem ser estruturais, dando ênfase à organização do sistema, ou podem ser comportamentais, dando ênfase à dinâmica do sistema.
De acordo com Booch, Rumbaugh e Jacobson [1], há quatro objetivos principais para se criar modelos:

1. Eles ajudam a visualizar o sistema como ele é ou como desejamos que ele seja;
2. Eles permitem especificar a estrutura ou o comportamento de um sistema;
3. Eles proporcionam um guia para a construção do sistema;
4. Eles documentam as decisões tomadas no projeto.

Modelagem II

Conceitos de orientação a objetos e UML

UML - Introdução

Através dos modelos, conseguimos obter múltiplas visões do sistema, particionando a complexidade do sistema para facilitar sua compreensão, e atuando como meio de comunicação entre os participantes do projeto. Portanto, uma linguagem de modelagem padronizada, tal como a UML, é fundamental para a construção e o entendimento de bons modelos.

Se você quiser construir grandes softwares, o problema não se restringirá a uma questão de escrever grandes quantidades de código – de fato, o segredo está em elaborar o modelo correto e pensar em como será possível elaborar menos código, com maior confiabilidade e qualidade. Isso faz com que o desenvolvimento de software de qualidade se torne uma questão de arquitetura, processo e ferramentas, reduzindo um pouco a responsabilidade da implementação.

Modelagem II

Conceitos de orientação a objetos e UML

Princípios da Modelagem

Segundo Booch, Rumbaugh e Jacobson [1] há quatro princípios de modelagem, os quais são citados a seguir.

A escolha dos modelos a serem criados tem profunda influência sobre a maneira como um determinado problema é atacado e como uma solução é definida.

Em relação aos softwares, a escolha de modelos poderá ser modificada, de maneira significativa, de acordo com a visão de mundo do projetista. Projetistas distintos podem criar modelos bastante variados entre si, dados os mesmos requisitos do software. Um ponto importante é que cada visão de mundo conduz a um tipo diferente de sistema, com custos e benefícios diversos. A visão de mundo, no caso do desenvolvimento de software, refere-se à experiência dos desenvolvedores e às tecnologias que eles conhecem.

Modelagem II

Conceitos de orientação a objetos e UML

Princípios da Modelagem

Cada visão de mundo pode gerar modelos de sistema diferentes (as tecnologias e técnicas utilizadas nos sistemas podem ser diferentes). O modelo do sistema deve adequar-se não só aos requisitos do mesmo, como também aos conhecimentos da equipe de desenvolvimento. Criar projetos que utilizem tecnologias desconhecidas pela equipe pode atrapalhar o andamento do projeto, pois os desenvolvedores teriam que aprender essas novas tecnologias e como utilizar seus recursos.

Há várias técnicas de modelagem existentes e cada uma delas é adequada para um determinado problema. Uma técnica pode ser mais eficiente que outra em um caso, o que não significa que ela será a melhor em todos os casos. Dessa forma, saber qual técnica utilizar num determinado momento é um fator crucial para o sucesso do desenvolvimento de software de boa qualidade.

Modelagem II

Conceitos de orientação a objetos e UML

Princípios da Modelagem

Cada modelo poderá ser expresso em diferentes níveis de precisão.

Os melhores tipos de modelos são aqueles que permitem a escolha do grau de detalhamento das informações, dependendo de quem esteja fazendo a visualização e por que deseja fazê-la. O **diagrama de casos de uso**, permite escolher o grau de detalhes que será mostrado. Com esse diagrama, podemos ilustrar as funcionalidades do software numa visão de alto nível. E a partir de cada caso de uso elaborado, podemos subdividi-lo em outros casos de uso mais específicos, o que aumenta o nível de precisão do diagrama.

Um usuário final dirigirá a atenção para questões referentes ao que será visualizado pela interface gráfica do sistema e para as funcionalidades do mesmo, enquanto o desenvolvedor moverá o foco para a maneira como o software deve funcionar.

Modelagem II

Conceitos de orientação a objetos e UML

Princípios da Modelagem

Os melhores modelos estão relacionados à realidade.

Todos os modelos simplificam a realidade. Por isso, certifique-se de que sua simplificação não ocultará detalhes importantes.

De acordo com Booch, Rumbaugh e Jacobson [1], o tendão de Aquiles das técnicas de análise estruturada está no fato de não haver um relacionamento entre o modelo de análise e o modelo de projeto do sistema. Dessa forma, ao longo do tempo, aparecerá uma divergência entre o sistema concebido e o sistema projetado. Nos sistemas orientados a objetos, é possível estabelecer uma ligação entre várias partes do sistema, o que facilita bastante o desenvolvimento de sistemas complexos e o bom entendimento dos mesmos.

Nenhum modelo único é suficiente. Qualquer sistema não trivial será melhor investigado por meio de um pequeno conjunto de modelos quase independentes.

Modelagem II

Conceitos de orientação a objetos e UML

Princípios da Modelagem

Nesse contexto, a expressão “quase independentes” significa modelos que possam ser criados e estudados separadamente, mas que continuam inter-relacionados. Podemos criar vários modelos distintos a partir de um modelo qualquer. Por exemplo, suponha que tenhamos criado um diagrama de casos de uso. A partir dele, podemos criar diagramas de atividades, de classes, de sequência, etc. Cada diagrama pode ser estudado independentemente, mas eles continuam inter-relacionados.

Segundo Booch, Rumbaugh e Jacobson [1], para compreender a arquitetura de sistemas é necessário recorrer a várias visões complementares e inter-relacionadas, tais como: a visão dos casos de uso (expondo os requisitos do sistema); a visão de projeto (capturando o vocabulário do problema a ser resolvido); a visão do processo (modelando a distribuição dos processos e das threads do sistema); e a visão da implementação (com o foco voltado para questões de engenharia de sistemas). Cada uma dessas visões poderá conter aspectos estruturais como também aspectos comportamentais. Em conjunto, elas representam a base do projeto de software.

Modelagem II

Conceitos de orientação a objetos e UML

Introdução à linguagem UML

A UML (*Unified Modeling Language*) foi desenvolvida por Grady Booch, James Rumbaugh e Ivar Jacobson. Cada um deles possuía sua própria sistemática de criar modelos e a UML é a junção dos pontos fortes dessas três abordagens, adicionando novos conceitos e visões de linguagem.

Antes da elaboração da *Unified Modeling Language*, existiam diversos padrões para se criar modelos de software, o que dificultava a elaboração de software por equipes que utilizassem padrões diferentes. Esta linguagem é uma maneira de padronizar a modelagem orientada a objetos de forma que qualquer sistema possa ser modelado da maneira apropriada, simples de ser atualizado e compreendido. UML é uma linguagem muito expressiva, abrangendo todas as visões necessárias ao desenvolvimento e implantação de sistemas de software em geral.

Modelagem II

Conceitos de orientação a objetos e UML

Introdução à linguagem UML

UML é uma linguagem padrão para a elaboração da arquitetura de projetos de software. Ela pode ser empregada para visualização, especificação, construção e documentação de artefatos de software.

Ela aborda o caráter estático e dinâmico do sistema a ser analisado. Além disso, UML leva em consideração, já no período de modelagem, todas as futuras características do sistema. Tais características estão relacionadas a trocas de mensagens entre as diversas partes do sistema, o padrão arquitetural adotado e os padrões de projeto utilizados no mesmo.

A linguagem UML é usada no desenvolvimento dos mais diversos sistemas, variando desde sistemas de pequeno porte, tais como um sistema de comércio eletrônico para uma livraria, a sistemas de grande porte, como um sistema de transações bancárias. Ela pode abranger várias características de um sistema em um de seus diagramas, sendo aplicada nas diferentes atividades do desenvolvimento de software, desde a especificação de requisitos até a implementação e os testes.

Modelagem II

Conceitos de orientação a objetos e UML

Atividades gerais do desenvolvimento de software

Desenvolver software envolve diversas atividades que são realizadas em momentos distintos, de acordo com o processo de desenvolvimento de software utilizado. A seguir, destacaremos seis dessas atividades – a especificação de requisitos, a análise, o projeto, programação, testes e implantação, bem como a importância da UML para as suas realizações.

Modelagem II

Conceitos de orientação a objetos e UML

Atividades gerais do desenvolvimento de software

Especificação de requisitos

A atividade de especificação de requisitos envolve alta comunicação e colaboração com o cliente e todos os outros interessados no sistema, conhecidos como *stakeholders*, e abrange o levantamento de requisitos e outras atividades relacionadas, como por exemplo, a validação e a gestão de requisitos.

Segundo McLaughlin, Pollice e West [5], um requisito é uma necessidade única que detalha o que um produto ou serviço em particular deve ser ou fazer. É mais comumente utilizado na engenharia de sistemas ou engenharia de software.

Esta atividade captura as intenções e necessidades dos usuários do sistema a ser desenvolvido através do uso de modelos conhecidos como casos de uso. Cada caso de uso modelado é descrito através de um texto e/ou diagrama, os quais especificam os requerimentos do ator externo (usuário ou outro sistema) que o utilizará.

Modelagem II

Conceitos de orientação a objetos e UML

Atividades gerais do desenvolvimento de software

Especificação de requisitos

O diagrama de casos de uso mostrará o que os usuários finais deverão esperar do aplicativo, conhecendo toda sua funcionalidade sem se importar como ela será implementada. Os diagramas de casos de uso são uma forma de representar os requisitos funcionais de um sistema em um alto nível de abstração, sem considerar os objetos que fazem parte do sistema, a estrutura de classes e detalhes de implementação.

Eles indicam um caminho para a análise e compreensão do sistema como um todo sem a preocupação com o seu funcionamento interno. Esses diagramas criam uma forma clara e simples de comunicação e especificação entre usuários (pessoas que utilizarão o sistema) e membros da equipe de desenvolvimento.

Modelagem II

Conceitos de orientação a objetos e UML

Atividades gerais do desenvolvimento de software

Especificação de requisitos

Segundo Cockburn [6], um caso de uso se refere a um contrato que descreve o comportamento do sistema sob várias condições em que o sistema responde a uma solicitação de um dos seus interessados. Em essência, um caso de uso conta uma história sobre a maneira como um usuário final interage com o sistema sob um conjunto específico de circunstâncias. A história pode ser um texto narrativo, um delineamento das tarefas e/ou interações ou uma representação em diagrama. Independentemente de sua forma, um caso de uso descreve o sistema do ponto de vista do usuário.

Muitos problemas e falhas podem surgir durante a especificação dos requisitos em virtude das dificuldades desta fase inicial do desenvolvimento. Incertezas e erros podem ocorrer por causa da má comunicação ou falta de especialistas do domínio do problema em questão.

Modelagem II

Conceitos de orientação a objetos e UML

Atividades gerais do desenvolvimento de software

Especificação de requisitos

Além disso, a linguagem natural não é precisa, o que pode acarretar interpretações conflitantes. Há ainda a própria dificuldade do usuário em explicitar de forma completa, clara e estável seus requisitos. O uso de uma notação precisa que atue como base para evolução, documentação e comunicação, facilita a organização, o entendimento e a visualização padronizada do problema. O uso de modelos consistentes desde esta etapa, portanto, mostra-se determinante para evitar a propagação de erros para as atividades posteriores e assim diminuir os esforços para corrigi-los.

Segundo Booch [4], nossa habilidade em imaginar novas aplicações complexas sempre será superior à nossa habilidade de desenvolvê-las, e a construção de coisas erradas é um dos motivos da maioria das falhas dos projetos de software.

Modelagem II

Conceitos de orientação a objetos e UML

Atividades gerais do desenvolvimento de software

Análise

A atividade de análise, também conhecida como planejamento, estabelece um plano para o trabalho de engenharia de software que se segue. Ela descreve os riscos prováveis, os recursos que serão necessários, os produtos de trabalho a serem produzidos (documentações e diagramas), um cronograma do trabalho e as tarefas a serem conduzidas, tais como o refinamento dos casos de uso.

O objetivo da análise de requisitos é a compreensão e representação da informação sobre o domínio da aplicação, funcionalidades e comportamentos esperados e a construção de modelos que particionem estas informações de forma a constituir uma especificação básica do sistema sendo desenvolvido.

Modelagem II

Conceitos de orientação a objetos e UML

Atividades gerais do desenvolvimento de software

Análise

Esta atividade está preocupada com as primeiras abstrações (classes e objetos) e mecanismos que estarão presentes no domínio da aplicação. Tais mecanismos podem ser entendidos como algo intrínseco ao problema. Para facilitar o entendimento, os mecanismos do domínio da aplicação serão explicados tomando como exemplo um sistema de frenagem de um metrô.

Se estivermos desenvolvendo um sistema de frenagem de um metrô, teremos que estudar o domínio do problema para criar uma arquitetura que atenda aos requisitos específicos do metrô. Independente do processo de software que seja adotado, o metrô está suscetível a algumas regras físicas que devem ser entendidas.

Modelagem II

Conceitos de orientação a objetos e UML

Atividades gerais do desenvolvimento de software

Análise

Sabendo que o sistema de frenagem deve impor uma aceleração constante ao metrô, o sistema deve ser capaz de medir a velocidade atual do metrô e a distância que falta até o ponto de chegada. Com essas duas informações, ele será capaz de calcular a aceleração necessária para fazer o metrô parar no ponto de chegada. Nesse exemplo, a velocidade, a aceleração e a distância são variáveis presentes no domínio da aplicação, enquanto que o sistema de frenagem utiliza um mecanismo do domínio da aplicação (o cálculo da aceleração de acordo com a velocidade atual e a distância até o ponto de chegada).

Durante a atividade de análise, as classes são modeladas e ligadas através de relacionamentos com outras classes, e são descritas no Diagrama de Classes. Na análise só serão modeladas classes que pertençam ao domínio principal do software, ou seja, classes que gerenciam bancos de dados, interface, comunicação com outros sistemas, concorrência, entre outros, não estarão presentes nesta etapa.

Modelagem II

Conceitos de orientação a objetos e UML

Atividades gerais do desenvolvimento de software

Projeto

Nesta atividade, o resultado da análise é expandido em soluções técnicas. Novas classes serão adicionadas para prover uma arquitetura, como interface gráfica de usuários (GUI), gerenciamento de banco de dados e a comunicação com outros sistemas. As classes de domínio do problema, modeladas na atividade de análise, são utilizadas na etapa de projeto para criar uma arquitetura que atenda aos requisitos do sistema.

No exemplo do metrô, vimos algumas variáveis da aplicação, tais como a velocidade e a distância. Com elas é possível montar uma arquitetura para desenvolver o sistema de frenagem do metrô. Vimos um caso simples, mas e se houver atrito entre o metrô e a pista? E se houver resistência do ar? E se a trajetória for uma curva?

Modelagem II

Conceitos de orientação a objetos e UML

Atividades gerais do desenvolvimento de software

Projeto

Para resolver esses novos problemas, teremos que entender um pouco mais do domínio da aplicação. Se houver resistência do ar ou atrito, a maneira de se calcular a aceleração mudaria, mas a arquitetura do sistema seria a mesma. Nesse caso, apenas a implementação dos métodos mudaria, o que não acarretaria mudança na arquitetura do software.

Caso a trajetória seja curvilínea, seria necessário calcular a velocidade máxima com que o metrô poderia trafegar num determinado trecho para que não haja descarrilamento. Assim, teríamos que alterar a arquitetura do sistema para que ele seja capaz de calcular essa velocidade e manter o metrô com velocidade abaixo dela.

Modelagem II

Conceitos de orientação a objetos e UML

Atividades gerais do desenvolvimento de software

Projeto

Além de poder calcular essa velocidade, teríamos que garantir que a velocidade do metrô esteja sempre abaixo do valor limite. Em outras palavras, se o maquinista acionar um evento para acelerar o metrô e este possuir uma velocidade muito próxima da velocidade máxima, o evento precisa ser ignorado para garantir que não haja o descarrilamento do metrô. Nesse caso, teríamos uma nova restrição para o sistema, o que acarretaria numa necessidade de mudança da sua arquitetura.

Durante o Projeto, os projetistas devem ser capazes de analisar qual arquitetura, padrões de projeto e metodologias a serem utilizados para se construir um software de qualidade. As respostas a essas variáveis dependem da experiência dos projetistas sobre o domínio da aplicação e do conhecimento da equipe de desenvolvimento sobre as tecnologias e técnicas a serem utilizadas.

Modelagem II

Conceitos de orientação a objetos e UML

Atividades gerais do desenvolvimento de software

Projeto

Como podemos notar, a atividade de projeto resulta no detalhamento das especificações para a atividade de programação do sistema. Há várias metodologias para se criar projetos consistentes, por exemplo, *Scrum*[1], *Extreme Programming*, o modelo cascata, os modelos evolucionários, entre muitas outras.

Um desenvolvimento de software organizado tem como premissa uma metodologia de trabalho. Esta deve ter como base conceitos que visem a construção de um software de forma eficaz e os passos necessários para se chegar ao produto final esperado.

Modelagem II

Conceitos de orientação a objetos e UML

Atividades gerais do desenvolvimento de software

Projeto

Assim, quando se segue uma metodologia para o desenvolvimento de software, espera-se desenvolver um sistema que satisfaça todos os requisitos dos clientes de forma eficiente. Observando este aspecto, não faz sentido iniciar a construção de um software sem ter uma metodologia de trabalho bem definida e que seja do conhecimento de todos os envolvidos no processo. Porém, além de uma crescente demanda por softwares de qualidade, as empresas de desenvolvimento de software sofrem cada vez mais pressão por parte dos clientes para que o produto seja entregue com prazos menores. Este fato pode fazer com que uma metodologia de trabalho que já foi utilizada com sucesso em diferentes projetos não alcance o resultado esperado.

Modelagem II

Conceitos de orientação a objetos e UML

Atividades gerais do desenvolvimento de software

Programação

Na atividade de programação, as classes provenientes do projeto são implementadas na linguagem escolhida. Dependendo dos recursos providos pela linguagem utilizada, essa implementação pode ser uma tarefa fácil ou bastante complicada.
Nas atividades anteriores, os modelos criados são o significado do entendimento da estrutura do sistema, enquanto que na atividade de programação os modelos criados são convertidos em código.

Linguagens que não utilizam o paradigma orientado a objetos ou orientado a aspectos não podem usufruir de todos os diagramas disponíveis pela UML. Por exemplo, se estivermos utilizando uma linguagem procedural para desenvolvermos um sistema, não poderemos utilizar o diagrama de sequência para modelar seu comportamento. Nesse caso, teríamos que utilizar diagramas de fluxo de dados para modelar o comportamento do sistema.

Modelagem II

Conceitos de orientação a objetos e UML

Atividades gerais do desenvolvimento de software

Testes

Segundo Dijkstra, “testes podem mostrar a presença de defeitos, mas nunca a sua ausência”. De acordo com Pressman [3], aproximadamente 40% dos custos de software são de teste.

Um sistema geralmente é testado a partir dos testes de unidade, de integração, de aceitação e de sistema. Os testes de unidade são para classes individuais ou grupos de classes. Eles verificam se cada módulo funciona apropriadamente. Os testes de integração são aplicados para verificar se as classes estão cooperando umas com as outras como especificado nos modelos. Os testes de aceitação servem para assegurar que o software satisfaz os requisitos do usuário. Os testes de sistema verificam se a integração do software com outros sistemas é feita de forma apropriada.

Modelagem II

Conceitos de orientação a objetos e UML

Atividades gerais do desenvolvimento de software

Testes

Há dois tipos de teste principais: funcional ou caixa branca; estrutural ou caixa preta. O teste de caixa branca baseia-se apenas nos requisitos. Ele verifica se o comportamento do sistema é compatível com os requisitos do mesmo. O teste de caixa preta baseia-se no código-fonte, ou seja, os casos de teste são criados a partir da análise do código-fonte.

A importância dos testes para a qualidade do software é tão expressiva que até mesmo uma metodologia de desenvolvimento de software baseada em testes foi criada, TDD ou *Test Driven Development*.

Modelagem II

Conceitos de orientação a objetos e UML

Desenvolvimento de software e orientação a objetos

A orientação a objetos está intimamente ligada ao desenvolvimento de software. Segundo McLaughlin, Pollice e West [5], devemos seguir algumas etapas durante o desenvolvimento de software:

1. Verifique se o seu software faz o que o cliente deseja que ele faça;
2. Aplique os princípios básicos da orientação a objetos para adicionar flexibilidade;
3. Empenhe-se para ter um projeto reutilizável e que possa ser mantido.

A análise e projeto orientados a objetos têm como meta identificar o melhor conjunto de objetos para descrever um sistema de software. O funcionamento deste sistema se dá através do relacionamento e troca de mensagens entre estes objetos.

Modelagem II

Conceitos de orientação a objetos e UML

Desenvolvimento de software e orientação a objetos

Segundo Pressman [3], uma série de conceitos determina a qualidade de um projeto, como coesão, acoplamento, abstração, particionamento hierárquico, entre outros. Esses conceitos serão vistos nos próximos artigos dessa série. O uso de modelos ruins pode comprometer a codificação, a flexibilidade e a manutenção do sistema.

A orientação a objetos apresenta-se favorável à modularização e reutilização de componentes, facilita a transição entre análise e projeto, exibe melhores resultados em qualidade, produtividade e apresenta-se mais flexível a mudanças e adaptações.

A análise e projeto orientados a objetos são voltados para a construção de modelos melhores, mais próximos da realidade através do uso dos principais conceitos da orientação a objetos, como abstração, encapsulamento, herança e polimorfismo, criando um vocabulário e entendimento comuns entre os usuários do sistema e os desenvolvedores.

Modelagem II

Conceitos de orientação a objetos e UML

Conclusão

Vimos a importância da UML, os princípios de modelagem e como a UML pode ser utilizada nas atividades de desenvolvimento de software.

A orientação a objetos está intimamente ligada ao desenvolvimento de software, por isso também vimos alguns dos pontos fortes desse paradigma, enfatizando a flexibilidade e a manutenção do sistema, a modularização e a reutilização de componentes.

Referências

- Dev Media

Modelagem de Software com UML - DevMedia