NÃO PODE FALTAR

O PROCESSO DE DESENVOLVIMENTO DE SOFTWARE COM UML

Maurício Acconcia Dias

PROCESSO UNIFICADO NO DESENVOLVIMENTO DE SOFTWARE

É preciso utilizar a UML em conjunto com um processo de desenvolvimento de software, como o UP, um método interativo e incremental que apresenta quatros fases: concepção, elaboração, construção e transição, divididas em fluxos de trabalhos.

Fonte: Shutterstock.

Deseja ouvir este material?

Áudio disponível no material digital.

PRATICAR PARA APRENDER

Agora que você já tem os conhecimentos básicos da linguagem UML, é necessário entender a relação da linguagem com os processos de desenvolvimento. A linguagem auxilia no processo de desenvolvimento de software, por isso é utilizada nesse processo. Nesta seção, você vai aprender a respeito do processo de desenvolvimento de software chamado UP (do inglês *Unified Process*). Esse processo de desenvolvimento, desenvolvido por Ivar Jacobson, um dos criadores da UML, foi escolhido por apresentar características que facilitam a utilização da UML como linguagem de modelagem.

Outro ponto importante que abordaremos nesta seção é a relação de consistência entre os diagramas UML gerados para o desenvolvimento de software. Você deve se lembrar de que esses diagramas proporcionam diferentes visões do mesmo software, porém, essas visões devem ser consistentes, ou seja, devem apresentar uma representação única de onde e como se deseja chegar no processo de elaboração de um sistema de informação. Assim, imagine que haja dois diagramas UML distintos e com informações/requisitos diferentes, sendo utilizados por diferentes membros de um time de desenvolvimento do produto de software. Possivelmente, o produto final apresentará erros ou a comunicação entre módulos do software pode ser impactada, devido à inconsistência de informações dos diagramas.

Por fim, serão abordados os mecanismos gerais da notação UML. Cada diagrama tem suas particularidades, porém existem algumas ferramentas que são aplicáveis a todos eles e que podem melhorar sua legibilidade e enriquecer a informação apresentada. Dessa forma, conhecer essas ferramentas é importante para um programador que pretende utilizar a UML em seu dia a dia profissional.

Analisando o que foi apresentado é possível perceber que esta seção situará você, caro aluno, em relação à utilização da ferramenta que está aprendendo: a linguagem UML. Feito isso, será ainda mais fácil entender o motivo da linguagem ser tão importante e como utilizá-la de forma a obter o melhor resultado de sua aplicação. Cada uma das fases do UP será relacionada a um diagrama, tornando o entendimento ainda mais completo.

Com o objetivo de colocar os conhecimentos a serem aprendidos em prática, vamos analisar a seguinte situação-problema: você foi contratado por uma startup de desenvolvimento de sistemas, e agora fará parte de um time de desenvolvimento. É importante saber que cada empresa tem seus métodos de desenvolvimento e seus padrões e que todos os times de desenvolvimento estão adaptados a esse universo, para atingir suas metas e entregar os softwares no tempo determinado, atendendo a todos os requisitos solicitados pelos clientes.

Apesar de todas as técnicas de desenvolvimento de software e modelos serem amplamente conhecidos, você percebeu que a startup ainda utiliza métodos clássicos de desenvolvimento e, em alguns projetos, ainda não existe um processo de desenvolvimento com uma documentação clara do que se deseja nos produtos de software. Agora que você iniciou seus estudos de UML, sabe que é necessário

desenvolver a modelagem dos sistemas utilizando métodos e linguagens específicas para essa finalidade. Sabe, ainda, que a linguagem mais utilizada é a UML, por apresentar todas as características principais desejáveis para o problema em questão.

Sendo assim, como seus conhecimentos acerca do histórico de UML, suas características e seus benefícios estão sólidos, você acredita poder melhorar o desenvolvimento de software na empresa em que trabalha. Assim, você terá o desafio de mostrar os conceitos, vantagens, desvantagens, motivações e justificativas para que sua empresa adote o UML e se beneficie da ferramenta em seu processo de desenvolvimento.

Ao iniciar seus dias de trabalho nessa empresa você percebeu que existem problemas no processo de desenvolvimento de software, e a maioria deles estão associados ao fato de que os programadores nem sempre entendem realmente a importância e a análise e modelagem de um sistema antes do desenvolvimento, muitas vezes gerando falhas na implementação.

Após algumas apresentações feitas, você já ganhou destaque entre os diretores, que agora gostariam de saber como implantar a UML no processo de desenvolvimento. Você foi informado que deverá seguir o processo unificado dividido em quatro etapas. O líder do projeto solicitou um relatório descrevendo a melhor maneira de incorporar a utilização da linguagem UML no processo de desenvolvimento. Seu trabalho será fazer um relatório de no máximo duas páginas relacionando o UP à linguagem UML. Lembre-se de que eles já conhecem o UP, mas não identificaram os diagramas que podem ser utilizados no processo unificado. Você acha que seria interessante utilizar recursos gráficos para a apresentação dos conceitos? Bom trabalho!

Preparado para conhecer melhor como aplicar a UML em um processo de desenvolvimento de software e utilizá-lo futuramente no desenvolvimento de um software? Então, bons estudos!

CONCEITO-CHAVE

Você agora já está entendendo melhor o poder da linguagem UML e como ela pode ser útil para o desenvolvimento de um sistema. Além disso, você já compreendeu os tipos de diagramas e o objetivo de cada um deles. Apesar de todas as vantagens que conhecemos da linguagem UML, é necessário

compreender como utilizar essa linguagem de modo a obter os resultados esperados. Vamos considerar três pontos e, a partir deles, desenvolver uma análise a respeito da utilização da linguagem. Os três pontos são:

1. A linguagem UML é uma ferramenta que pode ser aplicada para a modelagem do sistema em processos de desenvolvimento de software.

2. A aplicação da linguagem de forma incorreta pode prejudicar o desenvolvimento de software, sendo necessário saber como fazê-lo.

3. A existência de mecanismos comuns a todos os diagramas da linguagem, que devem ser utilizados para que esses diagramas fiquem mais completos e explicativos.

Para entender como a linguagem UML pode ser aplicada ao UP, é necessário entender seu funcionamento. Definimos inicialmente a linguagem UML como uma linguagem visual para criação de modelos.

Um modelo de um software consiste em uma descrição completa desse software a partir de uma determinada perspectiva.

Embora a UML tenha 14 diferentes perspectivas para o software, a UML não define um processo de software padrão, logo, tais modelos em si não são suficientes para suprir as necessidades do desenvolvimento como um todo, e é preciso utilizar a UML em conjunto com um processo de desenvolvimento de software.

PROCESSO UNIFICADO

Basicamente, processos descrevem **quem** é o responsável por fazer um determinado artefato (**o que**), **como** serão executadas as tarefas e **quando**. Uma forma de escrever tais processos é por meio do chamado UP (do inglês *Unified Process*) ou Processo Unificado (PU), criado pelos fundadores da UML (JACOBSON, I., BOOCH, G., RUMBAUGH, 1999). Uma evolução do UP é o RUP (do inglês *Rational Unified Process*). O RUP é um refinamento do PU desenvolvido pela *Rational Corporation* (daí a origem do nome), posteriormente adquirida pela empresa IBM. O processo foi melhorado e foram criadas ferramentas para auxiliá-lo.

O UP consiste em um processo de desenvolvimento de software interativo e incremental em que, a partir de um conjunto de atividades bem-definidas, os requisitos de clientes (usuários) são convertidos em um sistema de software. Dessa forma, o desenvolvimento do software no UP foi definido com base nas características a seguir (WAZLAWICK, 2014):

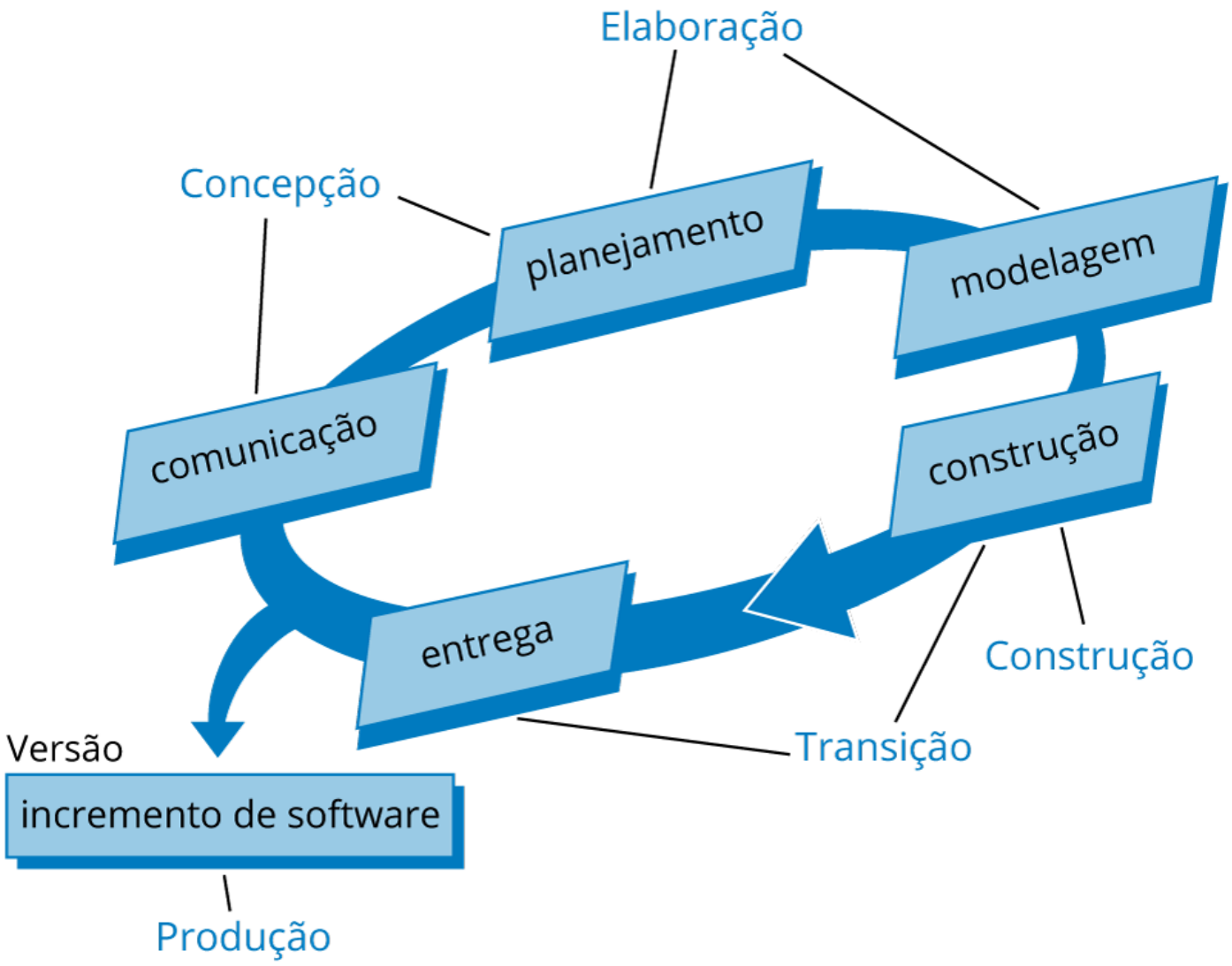
- O UP deve ser interativo e incremental, ou seja, a cada nova iteração são introduzidos incrementos de novas características à arquitetura do sistema. Pode-se também corrigir e refinar partes do projeto.
- O UP é dirigido por uma lista de casos de uso.
- O UP deve ser focado na arquitetura do sistema, a qual possibilita implementar os requisitos.
- O UP é orientado a riscos, e as partes do projeto com maior risco são priorizadas e tratadas primeiramente. Como exemplo, podemos citar os casos de uso que, identificados como mais críticos, são implementados e priorizados no momento do desenvolvimento.

Sendo assim, foi definido o **ciclo de vida do UP**. É importante destacar o que seria o ciclo de vida de um método de desenvolvimento, que nada mais é do que o conjunto de etapas que esse processo desempenhará e como essas etapas são divididas ao longo do tempo de desenvolvimento.

FASES DO UP

No caso do UP, o ciclo de vida apresenta quatro fases. A Figura 1.4 ilustra o processo unificado e como são organizadas as fases, segundo Pressman e Maxim (2016, p. 57).

Figura 1.4 | Organização do Processo unificado



Fonte: Pressman e Maxim (2016, p. 57).

As quatro fases do processo unificado são descritas a seguir:

- **Concepção:** é definido o escopo do projeto e são elaborados os casos de uso e de negócio. As definições ocorrem após uma conversa com o cliente, momento em que são definidos os casos de uso do sistema – e esse é um dos artefatos dessa parte do fluxo. Elabora-se uma proposta de arquitetura rudimentar que apresenta um estado inicial provisório do sistema e seus subsistemas, que seria outro artefato. Além disso, nessa fase identificam-se os possíveis riscos que podem surgir no decorrer do projeto e propõe-se o cronograma de desenvolvimento do sistema, estimando os custos do projeto considerando as restrições econômicas.
- **Elaboração:** nessa fase do projeto do software as características principais são especificadas com o detalhamento da arquitetura inicial realizada na fase concepção. Assim, na fase elaboração são definidos os requisitos funcionais e refinada a base da arquitetura do software. Nesse momento, os casos de uso desenvolvidos anteriormente são expandidos. A elaboração inclui cinco visões do software, também chamadas de fluxos de trabalho ou disciplinas, e os artefatos desenvolvidos nessa fase são: casos de uso, análise, projeto,

implementação e disponibilização. Nesse momento também ocorre uma revisão detalhada de riscos, escopo do projeto e datas de entrega.

- **Construção:** quando o desenvolvimento do software é realizado, tendo como entrada a arquitetura do software. São construídos ou adquiridos os componentes de software necessários para tornar o sistema funcional para o usuário final. Há uma remodelagem dos modelos de requisito e projeto, para considerar o incremento de software, e são esses os artefatos gerados.
- **Transição:** fase importante do desenvolvimento, quando o produto é entregue para os usuários. Na transição testes são feitos e os incrementos do sistema podem ser implantados. O software em fase beta é testado pelos usuários finais, que relatam os erros e mudanças necessárias. Esse também é o momento no qual a equipe de software elabora o material de apoio (como manuais e tutoriais).

Além dessas, após a transição, de acordo com Pressman e Maxim (2016, p. 57), há a **fase de produção**, em que há um contínuo monitoramento da utilização do software e é realizado o suporte para o ambiente operacional (infraestrutura). Relatórios com problemas e mudanças a serem realizadas são elaborados.

A questão de ser interativo e incremental está associada ao fato de que as fases do ciclo de vida ocorrem simultaneamente e de forma escalonada e não sequencial, como o clássico modelo cascata.

Como é possível perceber nas definições, os artefatos gerados em cada uma das fases do fluxo são as informações internas ou externas, que desempenham papéis no desenvolvimento do sistema.

FLUXOS DE TRABALHO (DISCIPLINAS)

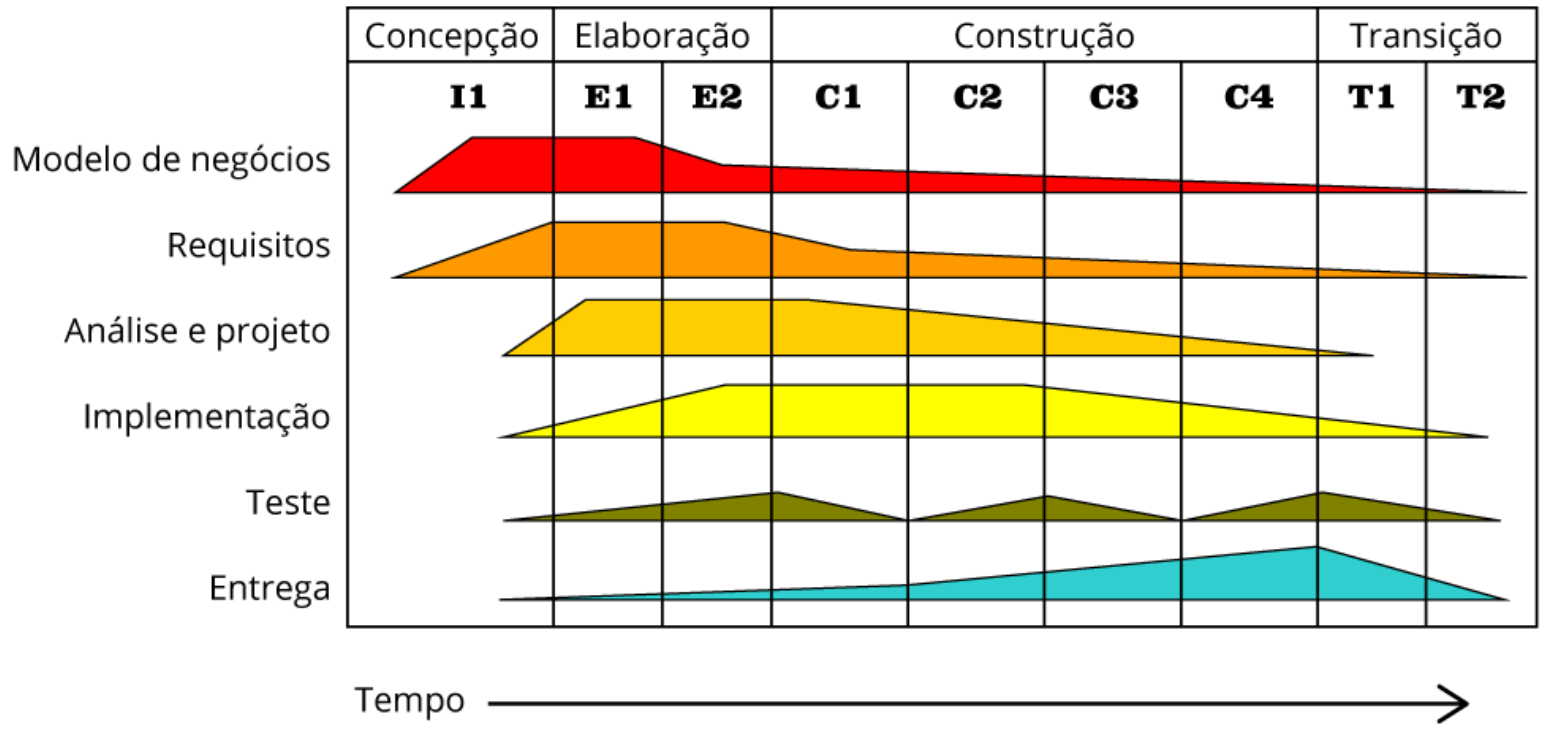
O desenvolvimento incremental é proposto da seguinte forma no UP: são feitos vários miniprojetos em tempos limitados, chamados de iterações, e o resultado de cada uma das iterações é um sistema testado, integrado e executável. Assim, as quatro fases do UP são divididas em fluxos de trabalho, também chamadas de disciplinas.

Os fluxos de trabalho correspondem ao conjunto de atividades (disciplinas) e os responsáveis por executar essas atividades. No processo unificado, cada fase pode ser dividida em disciplinas, as quais abrangem todas as fases do UP. Alguns

exemplos de disciplinas são o modelo de negócios, requisitos, análise e projeto, implementação, teste e entrega. Para cada disciplina pode-se produzir um ou mais artefatos, que são documentos, manuais, contratos, diagramas, código-fonte e plano de testes, entre outros.

Importa destacar que as fases do UP englobam todo o processo do desenvolvimento de software para cada uma das iterações. Esse comportamento é mostrado na Figura 1.5, uma imagem clássica sobre o UP que apresenta a concorrência entre as quatro fases iniciais.

Figura 1.5 | Desenvolvimento iterativo do UP



Fonte: adaptada de Wikimedia Commons.

Cada um dos produtos das iterações é indicado por uma das colunas da Figura 1.5, porém é possível notar que, dependendo da fase da UP, algumas das disciplinas podem aparecer mais concentradas em determinadas fases (apresentadas pelos gráficos coloridos da figura). Isso ocorre devido à evolução no desenvolvimento do software, por exemplo: a disciplina Modelo negócios (gráfico vermelho), que corresponde a I1 da Figura 1.5, requer uma maior atenção na fase Concepção, comparando à fase Transição T1 e T2. Nessa fase, o modelo de negócios já deve estar claro e consolidado. Outro exemplo é a disciplina requisitos, que tem uma carga mais concentrada nas fases de concepção e elaboração, quando comparada às fases Construção e Transição (WAZLAWICK, 2014).

REFLITA

Os métodos clássicos de desenvolvimento de software, como o modelo Cascata (PRESSMAN; MAXIM, 2011), apresentavam diversos problemas, principalmente relacionados à necessidade de finalizações de tarefas para iniciar outras – e tais situações podem ser contornadas com a aplicação de

> métodos iterativos como o UP. Considere o seguinte fato: quando se desenvolve um software em que só se pode executar a versão final, existem vários riscos e problemas, como alto número de erros e a possibilidade de o cliente não ficar satisfeito com o produto final – mudanças nos requisitos causam um grande retrabalho. No caso do UP não acontece isso justamente pelo fato de o desenvolvimento ser iterativo. Você consegue descrever por que a iteratividade no processo – ou seja, a criação de pequenos softwares seguidos pela integração – melhora os requisitos que apontamos como riscos e problemas?

RELAÇÃO ENTRE OS DIAGRAMAS E AS FASES

Agora que você conhece melhor o UP, é possível discutir a relação entre os diagramas UML e as fases apresentadas.

Iniciando pela parte da análise de requisitos e modelo de negócios na **fase de concepção**, é possível relacionar diretamente o diagrama de casos de uso que apresenta as principais relações de utilização do software. Este é acompanhado pelos diagramas de sequência, colaboração, atividade e máquinas de estado, que buscam apesentar o comportamento do software de acordo com a análise dos requisitos e do modelo de negócio.

Em seguida, a **fase de elaboração** (que envolve a análise e projeto) utiliza os diagramas de classe por estar mais próxima da implementação do sistema e, novamente, de sequência, colaboração, atividade e máquinas de estado, porém evoluídos de acordo com a evolução obtida pela construção do diagrama de classes.

Na **fase de construção** também é importante incluir o diagrama de instalação, já que conhecer o hardware e o ambiente de software que serão utilizados na implantação é importante durante o desenvolvimento.

Na **fase de implementação** os principais diagramas utilizados são os de classes, componentes, sequência e colaboração. Perceba que são diagramas que dispõem de informações diretamente relacionadas à implementação do sistema propriamente dito.

O fluxo de trabalho relacionado a testes utiliza a informação de todos os outros diagramas já desenvolvidos, para que seja possível testar todas as funcionalidades do sistema. Analisando a utilização dos diagramas em cada fase do ciclo de vida do

UP (apresentada na Figura 1.5), é possível perceber que são escolhidos os diagramas que apresentam a informação necessária e importante para cada etapa.

Ao construir os diagramas UML para cada fase do processo, é importante que se mantenha a consistência dos diagramas desenvolvidos. Lembre-se de que os 14 diagramas UML apresentam diferentes aspectos do mesmo software e, sendo assim, não devem conter informações conflitantes. Os trabalhos de análise de consistência de diagramas UML seguem o padrão de definição de regras para criação dos diagramas que, quando utilizadas, têm por objetivo evitar a inconsistência.

O trabalho de Alsammak, Sahib e Itwee (2018) apresenta uma abordagem interessante das regras de consistência de diagramas UML, com 11 regras básicas, e o trabalho de Harza e Dey (2014) apresenta outras 11 regras, sendo que algumas são referentes a relações entre diagramas diferentes dos casos apontados no primeiro trabalho. Então é possível compilar essas regras de forma sintética no seguinte conjunto:

1. O número de objetos no diagrama de sequência deve ser o mesmo do número de classes presente no diagrama de classes. Isso significa que há inconsistência caso um objeto no diagrama de sequência não tenha classe correspondente.
2. Deve se atentar para as atualizações do diagrama de classes e reproduzi-las corretamente no diagrama de sequência.
3. Se uma relação de dependência é expressa no diagrama de classes, ela deve estar representada por ao menos uma passagem de mensagem entre os objetos dessas classes no diagrama de sequência. Caso não exista relação expressa entre objetos no diagrama de sequência, a relação de dependência no diagrama de classes deve ser removida.
4. O nome dos métodos deve ser respeitado entre os diagramas de classe e sequência; caso contrário uma iteração entre objetos representada no diagrama de sequência pode fazer referência a um método não existente. Caso isso ocorra, é complexo entender se o método deveria existir ou não, se foi um erro apenas de nome ou se foi erro de representação no diagrama de classes.
5. Os diagramas de classe e sequência devem ser sincronizados. As mudanças no diagrama de classe impactam o diagrama de sequência e, se não forem aplicadas corretamente e de modo síncrono, a implementação do sistema pode

apresentar erros. Portanto, sempre que uma mudança no diagrama de classes ocorrer deve-se atualizar imediatamente o diagrama de sequência.

6. Os objetos representados no diagrama de comunicação e de sequência devem apresentar um padrão de nomenclatura; isso evita que um objeto da mesma classe tenha nomes diferentes nos dois diagramas. Por exemplo: um objeto chamado de “D” pode ser uma instância da classe X no diagrama de comunicação e da classe Y no diagrama de sequência. Isso cria inconsistência. Adotando-se uma regra de nomenclatura de objetos, esse problema pode ser facilmente evitado.

7. Cada uma das situações representadas no diagrama de casos de uso deve ter uma operação correspondente no diagrama de classes. Isso garante que os casos de uso serão tratados de forma correta na implementação em sua totalidade. Além disso, evita que haja erros de interpretação do tipo “esta operação representa quais casos de uso? ”.

8. Cada caso de uso deve ter um substantivo e um verbo associados. O substantivo do caso de uso corresponde ao nome da classe que o representa no diagrama de classes. O verbo deverá ser relacionado a uma operação representada no diagrama de classes, garantindo inclusive que os diagramas respeitem a regra anterior.

9. Cada caso de uso deve ocorrer pelo menos uma vez no diagrama de sequência, caso contrário há uma inconsistência entre os dois diagramas.

10. Para cada caso de uso representado no diagrama de sequência, todos os atores associados ao caso de uso devem ser representados como participantes. Caso isso não ocorra, o ator do diagrama de casos de uso pode estar representado de forma errada ou o diagrama de sequência pode ter sido construído de forma errada. Nos dois casos há inconsistência.

11. Para cada caso de uso deve existir ao menos um diagrama de sequência.

12. Para cada diagrama de sequência deve existir um diagrama de tempo.

13. Os participantes associados a um determinado diagrama de tempo devem estar representados no diagrama de sequência correspondente.

É importante entender que essas regras, ao serem seguidas, diminuirão o número de inconsistências nos diagramas UML. Porém, não há garantia que os diagramas gerados serão totalmente consistentes. Além disso, as ferramentas de geração de

diagramas UML consegue manter a consistência de diagramas em um mesmo projeto até um certo nível. Assim, em projetos maiores recomenda-se a utilização de softwares de desenvolvimento de diagramas. Existe também uma versão Ágil do UP, que pode ser vista em Ambysoft (2020).

EXEMPLIFICANDO

Apesar de você ainda não conhecer a fundo os diagramas existentes na linguagem UML, é possível entender o quanto uma inconsistência nos diagramas pode prejudicar o desenvolvimento de software.

Imagine que ao desenvolver o diagrama de casos de uso de um software, o cliente tenha sido enfático com relação a um determinado caso e a equipe tenha negligenciado o ponto 7, sobre cada caso de uso ter uma operação correspondente no diagrama de classes, por achar que esse caso já estava sendo implementado em outra operação.

Ao final da implementação, o caso de uso em questão muito provavelmente não terá sido tratado da forma correta, e o cliente, ao testar o software, perceberá gerando uma necessidade de correção no produto final. O tempo gasto com essa correção vai alocar recursos de desenvolvimento da empresa que poderiam estar em outros projetos gerando prejuízo financeiros para a empresa, além de deixar o cliente muito insatisfeito com a situação.

Esse caso poderia ter sido evitado caso a inconsistência apontada pelo ponto 7 tivesse sido levado em consideração, poupando tempo de correção do software, horas de programadores e deixando o cliente satisfeito.

UTILIZAÇÃO DE DIAGRAMAS UML

Uma questão acerca da utilização de diagramas UML para o UP deve ser tratada de forma mais específica. Tanto o UP quanto a UML são independentes de objetivo, ou seja, foram desenvolvidos para o desenvolvimento de qualquer tipo de software. Porém, em alguns casos é preciso saber como utilizar essas ferramentas de forma a obter bons resultados. Um dos tipos de software que demandam uma utilização consciente dos diagramas são os sistemas de informação.

Segundo Laudon e Laudon (2014), um sistema de informação é um conjunto relacionado de componentes de software que apresentam as funções de processar, recuperar, armazenar e distribuir informações em uma organização.

Essas informações serão utilizadas para controle, coordenação e tomada de decisões. Esse tipo de software é intimamente relacionado com as questões empresariais, o que deve ser levado em conta em seu desenvolvimento.

Como o foco está nos sistemas de software, é natural que a UML não enfatize os aspectos de um sistema de informações que visam ao valor e suporte que ele pode fornecer aos negócios, como estratégia (por exemplo, cadeias de valor e objetivos estratégicos) e organização (por exemplo, organogramas e processos de negócios). Esses problemas são tratados na "modelagem de negócios" (também chamada de modelagem corporativa). Mesmo assim, a UML dispõe de ferramentas que podem auxiliar nesses casos. Segundo Wazlawick (2014, p. 7), os diagramas que podem auxiliar no processo são os diagramas de casos de uso de negócio e os de atividade de negócio. Ocasionalmente, os diagramas de máquinas de estado também podem ser utilizados, porém existem ferramentas mais apropriadas para esses casos, como o BPMN (BARBARÁ; VALLE, 2009).

Portanto, é importante saber que no caso de sistemas de informação, a utilização da UML deve ser feita de forma diferenciada em relação ao desenvolvimento de um software com outras finalidades. Além disso, a questão mostra a versatilidade da linguagem UML, que mesmo não tendo sido desenvolvida para aplicações específicas é capaz de apresentar soluções para os problemas enfrentados.

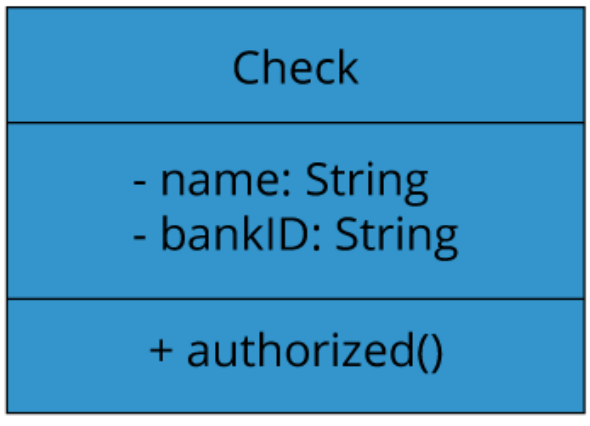
MECANISMOS COMUNS DA UML

O último tópico a ser abordado nesta seção é a questão dos mecanismos gerais da linguagem UML. Existem, portanto, 14 diagramas disponíveis para serem utilizados na linguagem UML e todos eles apresentam particularidades. Entretanto, há alguns mecanismos que são comuns a todos os diagramas e podem ser utilizados para melhorar a legibilidade, bem como incluir mais informações em determinado diagrama sobre o modelo.

Os mecanismos comuns são quatro: **especificações**, **adornos**, **divisões comuns** e **mecanismos de extensão**. Como sabemos, a UML é uma linguagem gráfica, porém associada a cada elemento dessa linguagem gráfica existe uma **especificação** que descreve exatamente aquele elemento. Por exemplo: relacionada a uma classe existe toda a especificação, que descreve todos os atributos, operações e comportamentos que a classe incorpora. Visualmente, o elemento da classe mostrará apenas parte dessa informação. Basicamente, a parte visual permite um entendimento rápido e global de cada parte do sistema, enquanto a especificação permite especificar os detalhes.

Cada diagrama UML começa com um símbolo e depois os adornos são adicionados ao diagrama para compor os elementos da modelagem. Um **adorno**, portanto, é um elemento do diagrama UML que tem uma arte gráfica única e direta, tornando-se uma representação visual objetiva daquele elemento. Veja o exemplo da classe na Figura 1.6. Neste adorno estão os elementos principais da classe, ou seja, seu nome, seus atributos e métodos. Nada mais é necessário para o entendimento global do diagrama de classes, porém essa representação é um adorno da UML.

Figura 1.6 | Adorno classe, em que o nome da classe é *Check*, os atributos são *name* e *bankID* e o método é *authorized()*.



Fonte: elaborada pelo autor.

Divisões comuns: os blocos de construção dos diagramas UML apresentam, em quase todos os casos, uma dicotomia entre sua interface e implementação (UML-DIAGRAMS, 2016). A interface é como um “contrato”, e a implementação é uma das possíveis realizações deste contrato, responsável por complementar a semântica da interface. Essa estrutura é a base do polimorfismo em linguagens orientadas a objetos, quando se define uma interface com vários métodos que serão implementados apenas nas subclasses.

Por fim, os **mecanismos de extensão** foram criados na UML para permitir que sejam feitas modificações na linguagem sem a necessidade de mudar toda a linguagem. Os três mecanismos de extensão da UML são (UML-DIAGRAMS, 2016):

- **Estereótipos:** é possível, na UML, utilizar o “desenho” de um determinado bloco e modificá-lo para um propósito específico, criando um novo objeto.
- **Restrições:** é possível, também na UML, alterar as restrições na construção de um diagrama. Em UML, as restrições são representadas pelas *strings* que acompanham as ligações entre elementos.
- **Valores predefinidos:** é possível predefinir valores específicos em um diagrama, para guiar a implementação do sistema ou gerenciamento de configurações do sistema.

> **ASSIMILE**
>
> Considerando o que foi colocado na questão dos sistemas de informação, é importante reforçar que cada desenvolvimento de software apresentará suas particularidades, e a linguagem UML está preparada para lidar com isso. Seus diagramas são versáteis a ponto de permitir sua utilização na modelagem de diversos tipos de sistemas. Além disso, a existência dos mecanismos de extensão da linguagem permite que ela seja modificada de acordo com as necessidades da modelagem de um software específico. Essas características tornam a UML uma linguagem extremamente útil e poderosa no desenvolvimento de sistemas.

Chegamos ao final da seção, e agora você já é capaz de relacionar a utilização dos diagramas UML com o método UP de desenvolvimento. Além disso, compreende o que é necessário para manter a consistência dos diagramas desenvolvidos e o motivo dessa consistência ser tão importante durante o desenvolvimento. O caso do desenvolvimento de sistemas de informação também foi apresentado para demonstrar o que ocorre em sistemas que apresentam particularidades. Por fim,

você conheceu os mecanismos comuns da linguagem UML, sendo capaz de aprender a fundo como construir seus diagramas e como modelar softwares utilizando a linguagem. Bons estudos!

REFERÊNCIAS

ALSAMMAK, I. L. H.; SAHIB, H. M. A.; ITWEE, W. H. A method of ensuring consistency between UML Diagrams. **Journal University of Kerbala**, v. 16, n. 1, 2018.

AMBYSOFT INC. **The Agile Unified Process**. Ambysoft, 2020. Disponível em: https://bit.ly/2P8yikw. Acesso em: 17 jun. 2020.

BARBARÁ, S.; VALLE, R. **Análise e Modelagem de Processos de Negócio**. São Paulo: GEN Atlas, 2009. 232 p.

HARZA, R.; DEY, S. Consistency between Use Case, Sequence and Timing Diagram for Real Time Software Systems. **International Journal of Computer applications**, v. 85, n. 16, 2014.

JACOBSON, I.; BOOCH, G.; RUMBAUGH, J. **The Unified Software Development Process**. Massachusetts, EUA: Addison-Wesley, 1999.

LAUDON, K.; LAUDON, J. **Sistemas de Informação Gerenciais**. 11. ed. Campinas: Pearson Universidades, 2014. 504 p.

PRESSMAN, R. S.; MAXIM, B. R. **Engenharia de Software**. 8. ed. Porto Alegre: AMGH, 2016.

UML – DIAGRAMS. **The Unified Modeling Language**. 2016. Disponível em: https://bit.ly/3f7qM41. Acesso em: 28 maio 2020.

WAZLAWICK, R. S. **Análise e Design Orientados a Objetos para Sistemas de Informação**. São Paulo: GEN-LTC, 2014. 488 p.