



Compreender melhor o projeto e facilitar sua manipulação futura, reduzir custos de manutenção, auxiliar na identificação de problemas... Esses são apenas alguns argumentos para justificar as vantagens da modelagem antes da codificação

Por: Flávia Freire

Apesar de, nos dias de hoje, usarmos o computador, o palm ou o celular para fazermos anotações, quem é que nunca listou em um pedaço de papel os afazeres para poder se organizar sem se esquecer de nada? No caso dos programadores, é natural que, antes de sentar em frente ao computador, ele exponha suas ideias em forma de rabiscos, mesmo que de uma maneira que só ele possa entender. A fim de globalizar este ato, Grady Booch, Ivar Jacobson e James Rumbaugh uniram suas metodologias que, em 1990, eram as três mais populares em modelagem de orientação a objetos, com o apoio da empresa Rational Software Corporation, hoje adquirida pela IBM, para criar uma linguagem unificada rigorosa para a modelagem de software. Dos métodos Booch, OMT (Object Modeling Technique) e OOSE (Object-Oriented Software Engineering), respectivamente, surgiu em 1996 a primeira versão da UML (Unified Modeling Language). Logo, as grandes empresas começaram a contribuir para o projeto e, em 1997, a OMG (Object Management Group), ou Grupo de Gerenciamento de Objetos, adotou a UML como a linguagem padrão para modelagem de software. "É o resultado de mais de duas décadas de evolução da cultura de modelagem desse tipo de software. Serve para descrever um software sob diferentes pontos de vista, permitindo destacar detalhes ou gerar visões mais globais. Descreve o mesmo que o código, mas de uma forma que privilegia a compreensão, por permitir enxergar informações de uma forma mais clara que no código. É útil para a compreensão de um software e, principalmente, como instrumento de auxílio à construção de software bem estruturado", diz Ricardo Pereira e Silva, autor dos livros 'Como Modelar com UML 2' e 'UML 2 em Modelagem Orientada a Objetos'.

Ao contrário do que muitos pensam em um primeiro momento, a UML não é uma linguagem de programação, e sim uma linguagem visual utilizada para modelar sistemas computacionais por meio do paradigma de orientação a objetos. O objetivo principal da UML é auxiliar na definição de requisitos, estrutura lógica, dinâmica de processos, comportamento, entre outros, de um projeto, para só então ele sair dos diagramas direto para o ambiente de desenvolvimento. "Modelar um projeto significa descobrir os conceitos envolvidos - análise: o quê? -; e definir as classes de software que implementam os conceitos - projeto: como?. É fácil concluir que, sem a boa modelagem de um projeto, não há nenhuma garantia de que o software correspondente seja de qualidade, isto é, fiel aos conceitos",

explica Marcus Sampaio, doutor em informática pela Université de Montpellier II, da França. Marcus é também professor do Centro de Engenharia Elétrica e Informática, da Universidade Federal de Campina Grande, e considera a falta de preocupação de seus alunos com a modelagem um problema muito sério. "Uma explicação possível é: enquanto a codificação é tangível, o mesmo não acontece com o projeto. Nossa constatação, na UFCG, é que os alunos já vêm com uma enorme dificuldade de discernir sobre o intangível, ou sobre abstração, de um modo geral. É por isso que as disciplinas de projeto são colocadas mais para o final do curso, quando se espera que o aluno tenha amadurecido", diz.

No caso das empresas, Marcus ressalta a importância de ter desenvolvedores que saibam modelar e que modelem antes de programar. "É fundamental ter um desenvolvedor na empresa que tenha poder de abstração e que saiba conversar no nível do usuário. Programar, sem noção precisa de 'o quê?' e 'como?', é atividade de alto risco e custo". Ricardo diz que a codificação sem planejamento acarreta no risco de geração de um produto mal estruturado. "Consequências inerentes: dificuldade de compreensão e, como decorrência, de manutenção e baixo potencial de reusabilidade. Tudo isso têm impacto direto no custo do software ao longo de seu ciclo de vida, que se estende bem além do período de desenvolvimento. O problema é que quem trabalha dessa forma não se qualifica para o desenvolvimento de software de alta complexidade, correndo o risco de perder o controle do desenvolvimento quando diante de um desafio que ultrapasse algumas centenas de classes, por exemplo. Empresas que apenas possuam profissionais com essa característica correm sério risco de perda de controle do processo de desenvolvimento diante de desafios que envolvam alta complexidade". Para Carlos Majer, professor da Unicid (Universidade Cidade de São Paulo), o interesse em modelar projetor depende da cultura de negócios da empresa. "A empresa pressiona a equipe de desenvolvimento para apresentar um sistema o quanto antes. Esta pressão muitas vezes envolve os elementos tempo e financeiro, fazendo com que o desenvolvedor tenha que liberar versões precoces do sistema. Em outros casos, pode ocorrer do próprio desenvolvedor não ter sido devidamente exposto ao assunto, o que evidencia a falta de treinamento adequado, muitas vezes caro e inacessível", diz.



Grady Booch

Confira uma entrevista exclusiva com um dos autores da UML. Booch é reconhecido internacionalmente por seu trabalho em arquitetura e engenharia de software. Faz parte do Centro de Pesquisa Thomas J. Watson da IBM, como chefe cientista de engenharia de software e trabalhou na Rational Software Corporation desde que foi fundada, em 1981. É também autor de seis livros best-sellers.

TI: Como foi o trabalho de criação da UML, na Rational Software Corporation? Qual o propósito da UML?

Booch: Quando o Jim, o Ivar e eu decidimos criar a UML, nossa intenção era unificar nossos três métodos (de maneira que o U se tornasse UML), para criar uma linguagem de visualização, especificação, construção e documentação de artefatos de um sistema de software intensivo. Esta missão trouxe a UML de hoje. Devo lembrar que, naquele tempo em que três de nós juntamos forças, havia uma indústria vibrante por métodos, assim como o mundo do desenvolvimento estava começando a se desvencilhar das linguagens de algoritmos, como Fortran, C e Cobol, para orientação a objetos, como Smalltalk e C++. Haviam muitas grandes ideias circulando, e a Rational simplesmente tomou a decisão empresarial de simplificar o mercado – e fazê-lo crescer – juntando a melhor daquelas ideias e formando uma, de padrão aberto. Deste modo, a UML nasceu, primeiramente dentro da Rational, e então, muito rapidamente envolvendo virtualmente todos os principais atuantes do mercado de desenvolvimento de software, para lançar um verdadeiro padrão global.

TI: Muitos desenvolvedores web pulam o estágio de modelagem para ir direto para a fase de codificação. O que você poderia dizer sobre a importância de modelar um projeto antes de começar a codificá-lo?

Booch: Não são apenas desenvolvedores web que fazem isso. É um hábito comum. Pensar antes de fazer é sempre uma boa opção, e modelar antes de codificar é uma ótima coisa a fazer também. A questão é o quanto pensar e o quanto codificar. Se você perceber que a UML é uma linguagem para argumentar sobre o sistema, então ela é poderosa em termos de ajuda, considerando abstrações incisivas, tendo a separação apropriada das preocupações, tendo uma distribuição balanceada de responsabilidades. Modelar é particularmente importante quando consideramos preocupante as ações simultâneas, que não acontecem facilmente já que sua cabeça está focada no código. Algumas vezes, contudo, as pessoas tendem a sub-modelar, e este não é um bom caminho. Minha regra é: se eu posso expressar e entender um projeto olhando diretamente o código, tudo bem. Mas decisões arquiteturais raramente podem ser feitas, e por isso, elas são candidatas primordiais para uma modelagem mais formal.

TI: Existem 14 diagramas da UML. Quais deles são os mais utilizados?

Booch: Primeiramente, diagramas de classe, e então, dependendo do seu domínio, diagramas de casos de uso e diagramas de sequência ou de estado.

TI: Como os desenvolvedores podem escolher a ferramenta ou diagrama UML corretos para um projeto para que seja criado um modelo que descreva corretamente a ideia do projeto?

Booch: Escolher o modelo certo é uma questão de entendimento de qual visão você está tentando abstrair. O modelo 4+1 de Philip Kruchten é o meu framework arquitetural favorito, e se você começar com este, ele te leva para quais pontos de vista (quais modelos) você deveria considerar. Para escolher a ferramenta, bem, eu sou parcial ;-), mas minha sugestão é selecionar alguma coisa que justifique as suas razões, e não perca o caminho, para selecionar algo que se encaixe no seu trabalho.

TI: É fácil interpretar diagramas?

Booch: Para mim é como respirar. Eu posso olhar para um diagrama e rapidamente saber o que está acontecendo. Resumidamente, isto leva tempo e experiência, assim como com qualquer linguagem, para propriamente entendê-la intuitivamente.

TI: É possível modelar qualquer tipo de projeto com UML?

Booch: Estou escrevendo um handbook, no trabalho, sobre arquitetura de software (www.booch.com/architecture). Até agora não encontrei nenhuma espécie de sistema de software complexo que eu não pudesse modelar na UML.

TI: O que há de novo na UML 2? Você já está pensando nas próximas versões? O que você melhoraria em uma versão futura da UML?

Booch: A UML 2 trouxe uma variedade de elementos para a linguagem, particularmente em suporte de Model-Driven Development. Para o futuro? Bem, eu simplificaria o metamodelo e trabalharia para identificar os 20% da UML que se aplica aos 80% dos problemas de modelagem que as pessoas encontram em uso real.

Interpretando diagramas UML

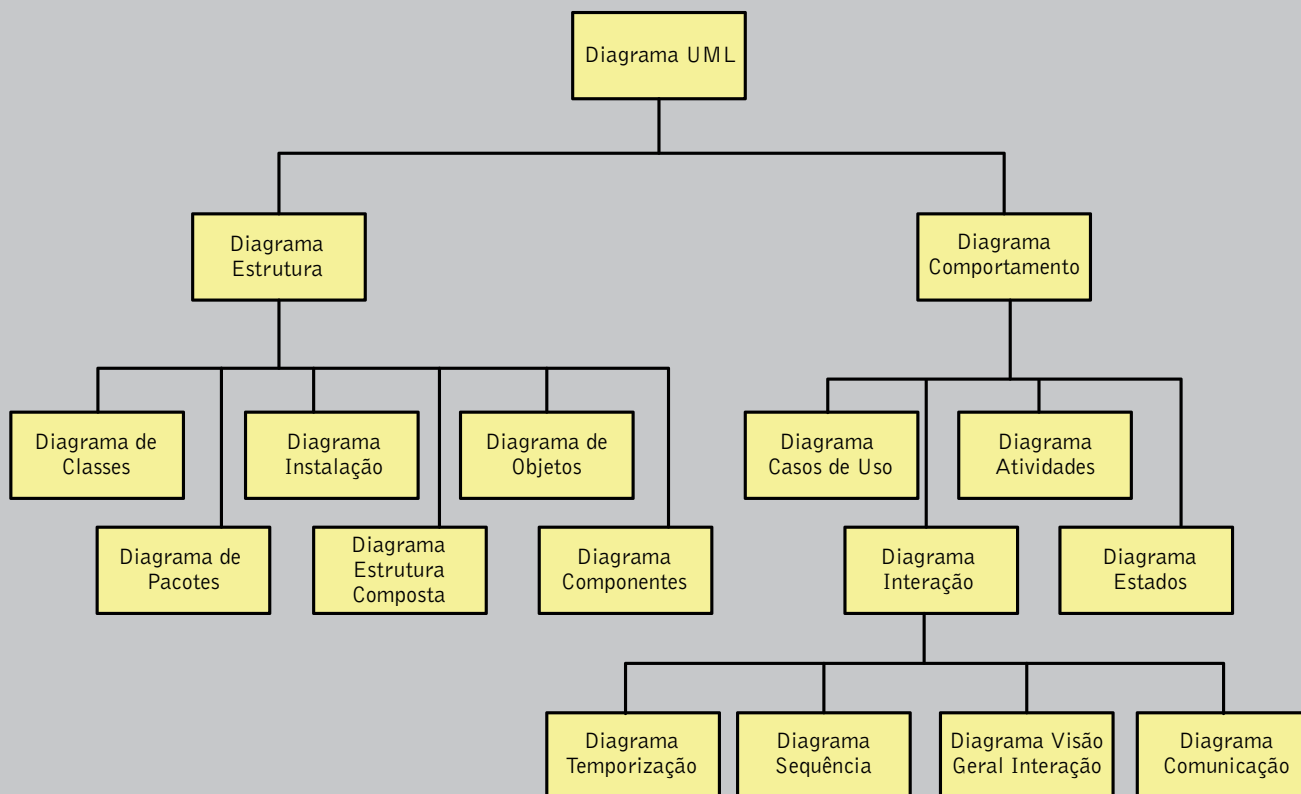
Primeiro, tem que saber modelar, para então conseguir interpretar. Como disse Grady Booch, é preciso uma certa experiência para olhar os diagramas e já saber como o projeto será desenvolvido. Uma das intenções dos criadores da linguagem era que estes diagramas pudessem ser 'passados de mão' sem que o autor tivesse que explicá-los ao desenvolvedor. "É o que se espera como rotina de trabalho em um ambiente de desenvolvimento. Isso, porém, não é obtido apenas com a linguagem UML. Assim como uma empresa precisa adotar regras para codificação ou testes, também deve estabelecer padrões de modelagem, para que a modelagem feita por um possa ser compreendida por todos. Outra questão a ser definida pelas empresas é o nível de detalhamento adotado nas especificações de projeto: modelagens mais detalhadas são mais independentes de outras fontes de informação e, conseqüentemente, permitem que o programador faça seu trabalho de forma mais independente", relata Ricardo.

Conheça os diagramas

Com a UML, os projetos são definidos através de diagramas. Com a última versão da linguagem, adotada pela

OMG (www.omg.com) em 2005, a UML passou a oferecer 13 tipos de diagramas que podem se complementar, cada um fornecendo uma visão do projeto a ser desenvolvido, e trouxe uma série de melhorias. "A UML 2 acrescenta quatro novos diagramas (de Pacotes, de Estrutura Composta, de Visão Geral de Interação e de Temporização), o que produz um acréscimo significativo de expressividade. Além disso, há melhoras significativas nos diagramas já existentes. Neste caso, cabe destacar a introdução da noção de agrupamento de mensagens no diagrama de Sequência, que permite a representação de laços e execuções condicionais, similares a comandos de linguagens procedurais. Outro destaque deve ser feito ao tratamento de software orientado a componentes, que melhorou bastante, dado que na primeira versão da linguagem o tratamento da abordagem era muito pobre sintática e semanticamente", explica o professor Ricardo. Para Marcus Sampaio, a UML 2.0 mantém os objetivos da UML 1.0, mas com novidades "A principal é a introdução de facilidades para a comunicação entre ferramentas (padrão XMI)", diz.

Com a versão 2.0 da UML, os diagramas foram divididos em Estruturais, Comportamentais e de Interação. Observe a figura abaixo que mostra todos os diagramas da UML, de forma hierárquica:



DIAGRAMAS ESTRUTURAIS

Diagramas de Classes, de Instalação, de Objetos, de Pacotes, de Estrutura Composta e de Componentes.

Diagrama de Classes: permite representar as classes do sistema, suas características (atributos) e ações (métodos). Você consegue expor as relações existentes entre as classes utilizando o conceito de orientação a objetos, o que é uma grande vantagem para os desenvolvedores que utilizam linguagens de programação orientadas a objetos, tão comum nos dias de hoje.

Diagrama de Instalação: descreve a configuração de elementos de suporte ao processamento, componentes de software, processos e objetos existentes nesses elementos. Ou seja, ilustra os componentes de hardware e software e sua interação com outros elementos de suporte ao processamento.

Diagrama de Objetos: é uma boa opção para explicar relacionamentos complexos entre classes. Mostra os objetos que foram instanciados das classes. Os nomes dos objetos são sublinhados e todas as instâncias são mostradas. Também é usado como parte dos diagramas de Colaboração, onde é mostrada a colaboração dinâmica entre os objetos do sistema. O Diagrama de Objetos fornece uma visão dos valores armazenados

pelos objetos de um diagrama de Classes em um determinado momento da execução de um processo. Os diagramas mostram apenas os métodos e propriedades que retornam objetos.

Diagrama de Pacotes: pode ser chamado também de Diagrama de Módulos. Pacote representa um grupo de classes, ou elementos, e é dependente de outros pacotes. O Diagrama de Pacotes mostra pacotes ou pedaços do sistema e relações entre pacotes. Ele é mais utilizado para ilustrar a arquitetura do projeto mostrando o agrupamento de suas classes. Não é preciso seguir uma hierarquia, ele pode ser utilizado em qualquer fase do processo de modelagem.

Diagrama de Estrutura Composta: mostra o relacionamento entre os elementos e especifica funcionalidades.

Diagrama de Componentes: descreve as dependências entre componentes de software, como de código fonte, código binário e executáveis. Destaca a função de cada módulo para facilitar a reutilização e auxilia no processo de engenharia reversa, por meio da organização dos módulos do sistema e seus relacionamentos. Ele mostra apenas o que é relevante em um subsistema de implementação. Este diagrama é representado na forma de tipos e não na forma de instâncias.

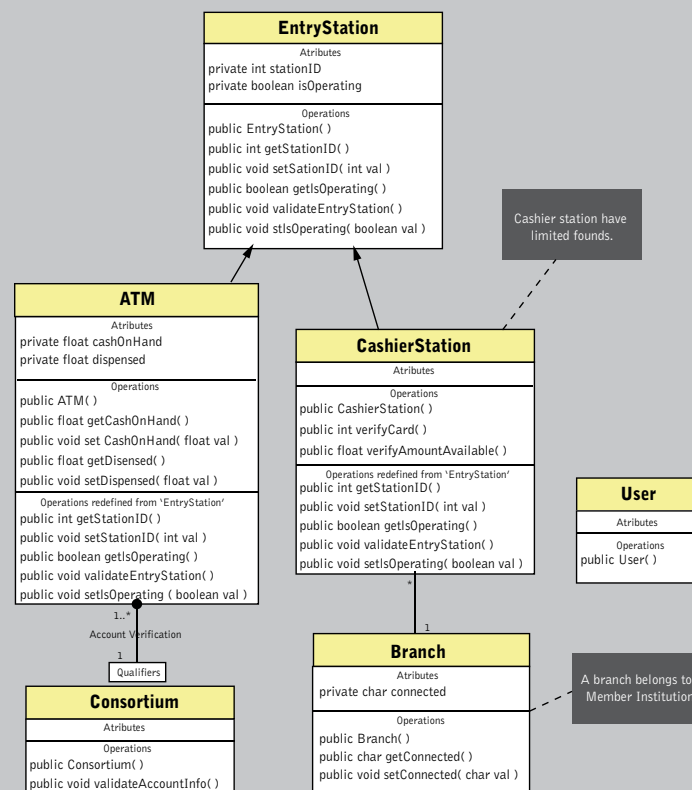


Diagrama de Classes

DIAGRAMAS COMPORTAMENTAIS

Diagramas de Casos de Uso, de Atividades e de Estados.

Diagrama de Casos de Uso: neste diagrama identificam-se os atores (participantes do caso de uso) e suas relações com os casos de uso (funcionalidades). É muito utilizado para a comunicação entre os colaboradores do sistema (desenvolvedores, analistas e usuários-chave), permitindo um nível de abstração macro que permite visualizar o que deve ser desenvolvido. Desta forma, o time sabe o que precisa ser feito. Apesar disto, este tipo de diagrama não mostra como deve ser feito.

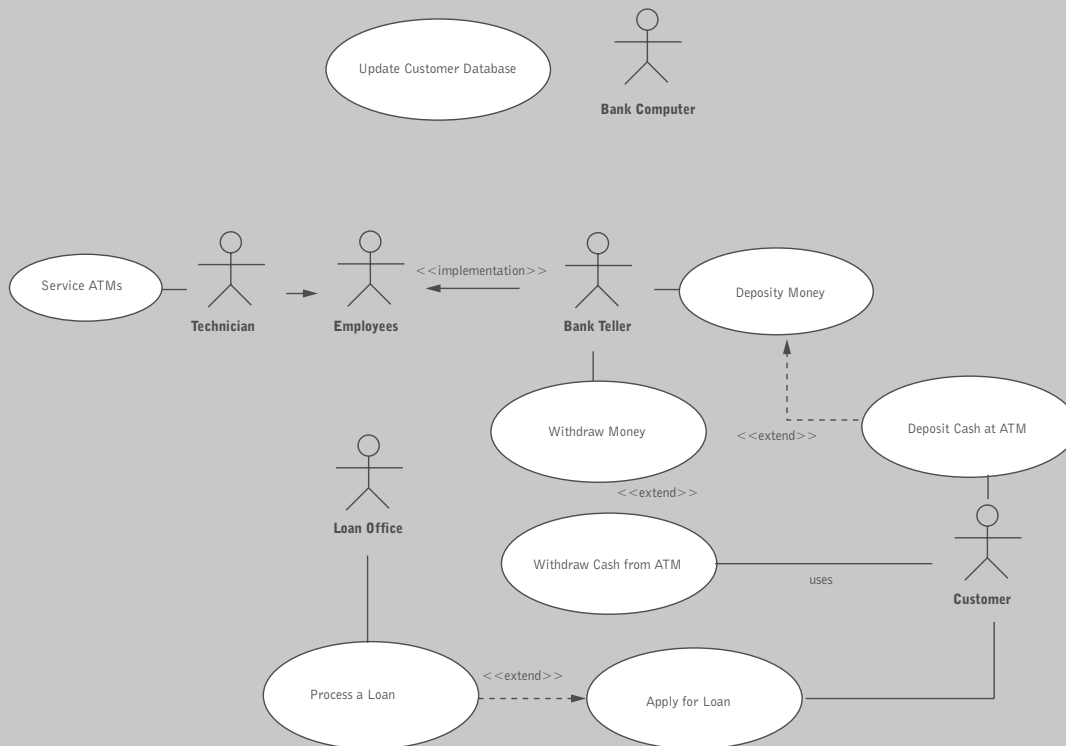


Diagrama de Casos de Uso

Diagrama de Atividades: adequado para representar o comportamento de algoritmos, processos de negócio e casos de utilização. Representa uma série de ações ou atividades, em que as transições são desencadeadas devido a simples conclusão dessas ações ou atividades.

Diagrama de Estados: descreve as sequências de estados que um objeto ou uma interação pode passar ao longo da sua existência em resposta a estímulos recebidos, em conjunto com as suas respostas e ações.

Fontes: Ricardo Pereira e Silva, Marcus Sampaio, Carlos Majer e Guia Prático UML 2 (<http://migre.me/2qtr>)

COMO CONSTRUIR OS DIAGRAMAS DA UML?

Existem diversos softwares para modelagem UML. Alguns destes editores até interpretam os diagramas e geram o código, mas sem garantias já que é preciso uma revisão manual. Conheça algumas ferramentas para edição de diagramas:

ArgoUML: software livre de modelagem UML que suporta todos os padrões de diagramas da UML 1.4. Roda em qualquer plataforma Java e está disponível em dez idiomas. O ArgoUML já ultrapassou 80 milhões de downloads e é usado por desenvolvedores do mundo inteiro. Site oficial: <http://argouml.tigris.org>

Jude: software para modelagem UML, gratuito e multiplataforma. Suporta UML, Diagrama de Entidade de Relacionamento, Flowchart, CRUD, Diagrama de Mapas e de Dados. Permite converter modelos. Site oficial: <http://jude.change-vision.com>

Microsoft Office Visio 2007: ajuda a visualizar, explorar e comunicar informações complexas, que vão de textos e tabelas a diagramas do Visio que comunicam informações. Em vez de imagens estáticas, pode-se criar diagramas do Visio conectados aos dados que exibem dados. Possui uma ampla variedade de diagramas para compreender, atuar a respeito e compartilhar informações sobre sistemas, recursos e processos organizacionais.

DIAGRAMAS DE INTERAÇÃO

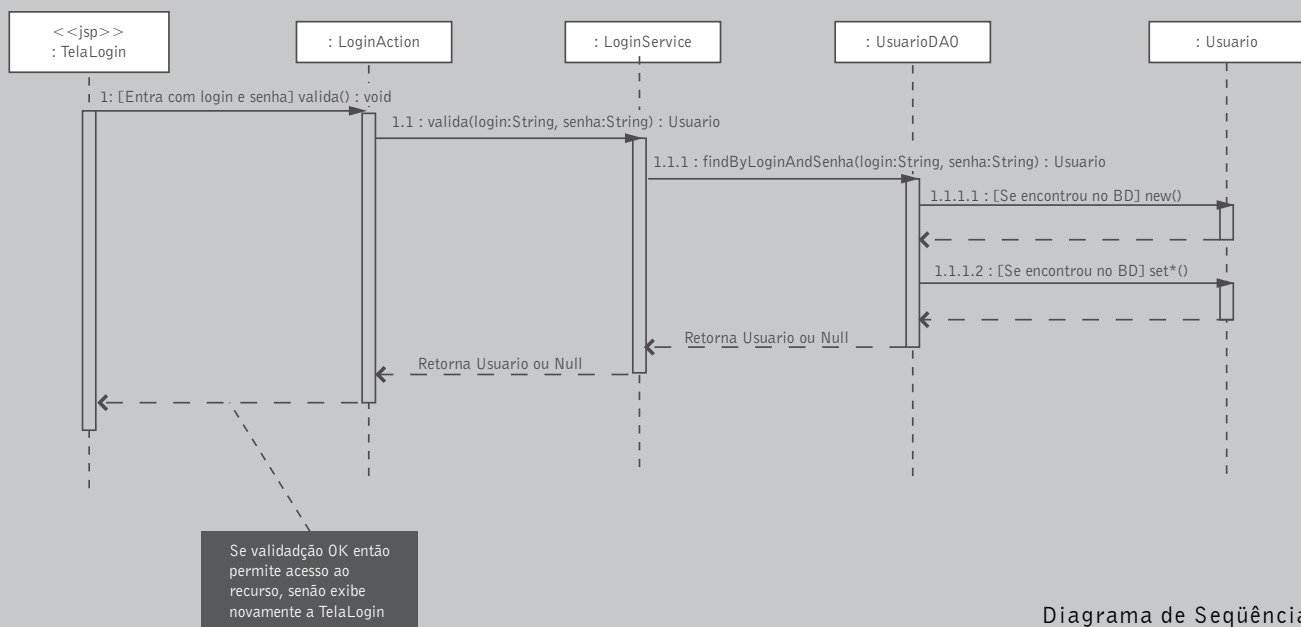
Diagramas de Temporização, de Sequência, de Visão Geral de Interação e de Comunicação.

Diagrama de Temporização: descreve a mudança no estado ou condição de uma instância de uma classe ou seu papel durante um tempo. Normalmente utilizado para demonstrar a mudança no estado de um objeto no tempo em resposta a eventos externos.

Diagrama de Sequência: ilustra interações entre objetos em um determinado período de tempo. Segundo Marcus Sampaio, pode ser visto como uma especialização de Diagrama de Casos de Uso, exprimindo uma ou diversas sequências de mensagens trocadas entre os usuários e o sistema.

Diagrama de Visão Geral de Interação: é uma variação do Diagrama de Atividades que fornece uma visão geral dentro de um sistema ou processo de negócio.

Diagrama de Comunicação: antes conhecido como Diagrama de Colaboração, complementa o Diagrama de Sequência. As informações mostradas no Diagrama de Comunicação são praticamente as mesmas apresentadas nos de Sequência, só que sem se preocupar com a temporalidade do processo, concentrando-se em como os objetos estão vinculados e quais mensagens trocam entre si durante o processo.



Fontes: Ricardo Pereira e Silva, Marcus Sampaio, Carlos Majer e Guia Prático UML 2 (<http://migre.me/2qtr>)

Site oficial: <http://office.microsoft.com/pt-br/visio>

Rational Rose: é uma ferramenta CASE que auxilia nos processos de construção de um software profissional. Foi criada pela Rational, posteriormente adquirida pela IBM e não é gratuita. Permite a modelagem com nove diagramas da UML (Casos de Uso, Classe, Componentes, Desenvolvimento, Objetos, Sequência Colaboração, Estados e Atividades). Permite também a construção de modelos de dados com possibilidade de exportação para construção da base de dados ou realização de engenharia reversa de uma base de dados existente. Dá suporte a Visual Studio. Site oficial: www.ibm.com/software/rational

VisualParadigm: ferramenta integrada para UML avançado que suporta o ciclo de desenvolvimento do software (análise, projeto, implementação, teste e depuração). Permite projetar todos os tipos de diagramas UML, reverter o mecanismo de código e gerar documentação. Possui exemplos e modelos passo-a-passo. Site oficial: <http://www.visual-paradigm.com>

yUML: ferramenta gratuita que permite criar diagramas UML de forma simples, podendo ser incluídos em uma página web. Há alguns diagramas prontos, que podem ser personalizados de acordo com suas necessidades. Site oficial: <http://www.yuml.me>