CAPTURANDO REQUISITOS DE PRODUTO DE SOFTWARE VIA UML

SÉRGIO CORDEIRO

Introdução

O levantamento dos requisitos dos produtos finais é uma das atividades mais importantes nas fases iniciais de um projeto. Essa atividade é, ainda, uma das mais difíceis de todas, principalmente porque apenas após o término do projeto pode-se medir com acuidade o acerto do levantamento. No caso de produtos de software, devido à complexidade dos mesmos, essa dificuldade é ainda maior. Por outro lado, o fato de tais produtos serem extremamente abertos, e portanto dependerem totalmente da especificação para que mesmo as características mais básicas sejam definidas, torna o levantamento crucial.
Requisitos errados são comuns, de acordo com as pesquisas realizadas. Pior, causam muitos danos, porque ocorrem muito cedo no ciclo de vida do projeto e, assim, exercem um efeito cascata sobre as atividades seguintes. Segundo pesquisa do Standish Group, realizada em 1994:

- 31% de todos os projetos foram cancelados
- 53% deles custaram mais de 80% a mais do que o orçado
- 13% foram realizados sem nenhuma participação dos usuários finais
- 12% tinham requisitos incompletos
- 12% apresentaram mudanças significativas nos requisitos ao longo do desenvolvimento

Já em 1994 apareceu uma norma, a IEEE 830, referente à especificação de requisitos para produtos de software. Essa norma, no entanto, padroniza apenas o conteúdo da especificação, não identificando métodos para descobrir quais devem ser os requisitos reais de um tal sistema [Leffingwell e Widrig 1999]. O PMBOK não recomenda nenhuma metodologia específica para a captura desses requisitos, mas são mencionadas ferramentas formais como o diagrama de contexto e o diagrama de fluxo de dados(DFD) [pmi 2015 1]. O primeiro tem como objetivo ilustrar o aspecto de mais alto nível de um sistema, que pode ser de software ou não: suas relações com entidades externas, ou seja, usuários e outros sistemas. O segundo ilustra os processos componentes do sistema e o fluxo de informações entre eles; tais fluxos são sempre entradas de um processo e saídas de outro. O diagrama de contexto é frequentemente chamado DFD de nível 0, o que deixa clara

a similaridade da abordagem; a principal diferença entre eles reside no fato de um retratar os aspectos externos e o outro, os aspectos internos do sistema.
Esses diagramas foram criados na década de 1980, como componentes de metodologias estruturadas idealizadas na época, como as de Constantine, GaneSarson e YourdonDeMarco [modernanalyst 2015 1, Yourdon 2015 3]. Outras ferramentas presentes em tais metodologias, no entanto, como o diagrama de acesso, por exemplo, não são citados pelo PMBOK; também não há menções a outras ferramentas de modelagem de software muito populares, como o diagrama de entidades e relacionamentos.
A ausência de ferramentas para tratamento de níveis mais detalhados pode ser considerada uma lacuna séria na metodologia. As metodologias estruturadas, mencionadas acima, executavam o detalhamento através da técnica de decomposição funcional, utilizando para isso o DFD. Essa abordagem se mostrou insuficiente com a evolução da disciplina de Engenharia de Software. Por exemplo, a baixa produtividade do analista, o tempo requerido para conclusão da etapa de análise do sistema, a dificuldade de se alterar alguns aspectos do sistema após implementado e a crescente importante de requisitos não-funcionais, como a segurança cibernética e o desempenho [Yourdon 2015 1]. Esses problemas suscitaram o surgimento de outros tipos de abordagem, como as metodologias de análise de dados e as de desenvolvimento rápido, bem como aprimoramentos no enfoque estruturado clássico. Entre as novas técnicas mais comuns, podem-se citar o emprego de ferramentas automatizadas, o uso de prototipação e o melhor casamento entre as etapas de levantamento de requisitos e implementação [Yourdon 2015 2].
Embora o diagrama de contexto e o diagrama de fluxo de dados sejam usados ainda hoje, foram desenvolvidas outras ferramentas gráficas complementares, algumas delas destinadas ao uso na captura de requisitos. O presente trabalho descreve algumas delas, justifica e ilustra seu uso. As ferramentas escolhidas fazem parte do conjunto de diagramas padronizados da UML (*Unified Modeling Language*). Nas seções seguintes são expostos maiores detalhes, sempre através de exemplos que ilustram os vários tipos de informação que se pode desejar expressar em um diagrama.

UML

A UML foi desenvolvida em meados da década de 1990, a partir da integração do trabalho dos teóricos Grady Booch, Ivar Jacobson and James Rumbaugh, sintetizado na obra "The Unified Modeling Language Reference Manual", de 1999. A versão 1.4.2 tornou-se um padrão ISO em 2005, sob o número 19501 [iso 2005 1].

Atualmente é gerenciada pelo OMG (Object Management Group) e se encontra na versão 2.5 [omg 2015 2]. Apesar de os autores preconizarem uma metodologia de desenvolvimento específica, chamada Rational Unified Process (RUP) [ibm 2015 1], as ferramentas da UML são empregadas em outros contextos diversos [omg 2015 1]. Como vantagens dos diagramas UML, em relação a ferramentas da geração anterior [1], podem-se citar:

(i) Obedecem a um paradigma de modelagem mais moderno, orientado a objetos.
(ii) Minimiza-se a redundância entre os diversos diagramas.
(iii) Utiliza-se um vocabulário e um conjunto de símbolos padronizado.
(iv) Maximiza-se o emprego de ferramentas eletrônicas [2] na elaboração da documentação.
(v) Permite-se o uso de geradores de código e outras ferramentas eletrônicas de desenvolvimento [2].
(vi) Abrangem outros aspectos do produto, além dos funcionais; por exemplo, a arquitetura geral do sistema pode ser visualizada ainda na fase de captura dos requisitos.
(vii) Adequam-se melhor a aplicações dirigidas por eventos (*event-driven*), como são a maior parte das interfaces com usuários e com outros sistemas atualmente.
(viii) Permitem estimar melhor a complexidade do produto final a partir dos primeiros levantamentos.
(ix) Permite a definição de entregas parciais já nas primeiras etapas da análise.

Uma característica peculiar da UML é que ela, apesar de originalmente genérica, pode ser expandida ou adaptada para domínios conceituais específicos através de extensões como perfis (*profiles*), padrões (*patterns*), gramáticas (*grammars*), tipos de dados (*data types*) e condições (*constraints*). Esse uso sofisticado da linguagem não será abordado neste trabalho.

Abordagem sugerida

O ideal perseguido pelas técnicas de gerenciamento é que o gerente seja capaz de executar seu trabalho sem conhecer detalhes técnicas do produto resultante. O levantamento dos requisitos é realizado por um profissional especializado, muitas

[1] Para uma comparação entre as ferramentas UML e as da análise estruturada clássica, ver a página 6 de [Podeswa 2005].

[2] Muitas ferramentas CASE para trabalho com UML são gratuitas, e algumas são de código aberto. O número de ferramentas de desenvolvimento ainda não é tão grande. Para uma lista, consultar [Wikipedia 2015 1].

vezes chamado de analista de negócios (*business analyst*, ou, ainda mais especificamente, *IT business analyst*), junto aos usuários. Esse analista criará um modelo de software, representado pelos diversos diagramas da UML, e ferramentas automatizadas auxiliarão o gerente a verificar coisas como, por exemplo, a aderência do produto aos requisitos. A finalidade do modelo é facilitar a comunicação entre os diversos *stakeholders* e entre estes e os desenvolvedores.
Empregar uma abordagem orientada a objetos desde as primeiras etapas do projeto oferece diversas vantagens. Objetos, que são entidades que abrangem dados e operações, são mais concretos que funções, principalmente para o público leigo. Pensar em termos de objetos é mais natural do que pensar em termos de funções. Além disso, o uso de objetos permite ocultar detalhes conforme a conveniência. A teoria dos objetos também oferece conceitos complementares e poderosos, como classes, generalização, herança, associação, estereótipos e polimorfismo, que possuem grande poder representativo de casos da vida real.
Vale lembrar que, para muitos produtos, a complexidade não reside nas operações feitas, e sim na base de dados, nas interfaces com outros sistemas ou com os usuários, no tratamento de eventos ou em requisitos puramente tecnológicos como plataforma utilizada, desempenho exigido e segurança da informação. Esses aspectos podem ser melhor capturados através de uma abordagem diferente da decomposição funcional. Nos casos em que a complexidade realmente residir na funcionalidade, entretanto, uma metodologia estruturada pode ser a mais indicada. A UML pode trabalhar com este tipo de metodologia também.
Um modelo UML representa aspectos estáticos e dinâmicos de um sistema. Na captura dos requisitos, podem-se usar os diagramas de casos de uso e os diagramas de atividades, quando na fase de Iniciação do projeto, e os diagramas de classe, as máquinas de estado e as especificações de testes na fase seguinte. O diagrama de pacotes pode também ser utilizado com proveito, inclusive auxiliando a geração da Estrutura Analítica do Projeto (EAP) [3] [4] [Podeswa 2005].
.

Diagramas UML

Diagrama de Casos de Uso. Um diagrama de casos de uso descreve uma interação simples no sistema. Essa interação pode ou não apresentar alternativas possíveis. O diagrama basicamente consiste de **atores**, que são as entidades externas ao sistema, **casos de uso**, que são as ações que o sistema permite que sejam executadas, e de eventuais **relacionamentos** entre eles. A informação pode ser

[3] Para explicações detalhadas, exemplos e gabaritos de documentos, ver os apêndices B, C, D e E de [Podeswa 2005].

[4] A versão mais atualizada da UML pode ser obtida em [omg 2015 2].

complementada por meio de artefatos comuns da UML, como títulos, notas, atributos, estereótipos, condições e símbolos especiais. Na figura 1, temos o exemplo de um ator cujo título é "Customer"interagindo com o sistema por meio de dois casos de uso, chamados "Login"e "Register with Book Shop"; esses casos são relacionados de uma forma estereotipada, chamada "extend". Esse estereótipo é pré-definido pela linguagem, e indica que a segunda atividade é um caso especial da primeira; o analista pode, se quiser, criar os estereótipos que achar úteis para descrever o problema.

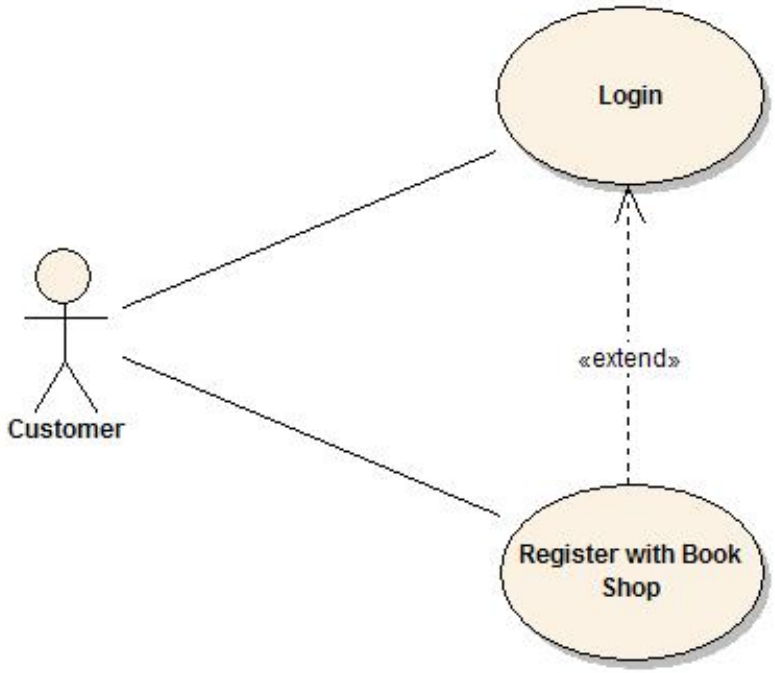

Figura 1 - Diagrama que ilustra a atividade de *login* em um sistema [sparx 2015 1].

A figura 2 mostra o uso de uma nota ("only available ..."), de outro estereótipo pré-definido ("include"), de um símbolo pré-definido (seta, para indicação de que o relacionamento é do tipo "generalização"), títulos para os relacionamentos (por exemplo, "(update)"), uma condição ("transferType = ...") e ainda mostrando a divisão em pacotes ("User Management").

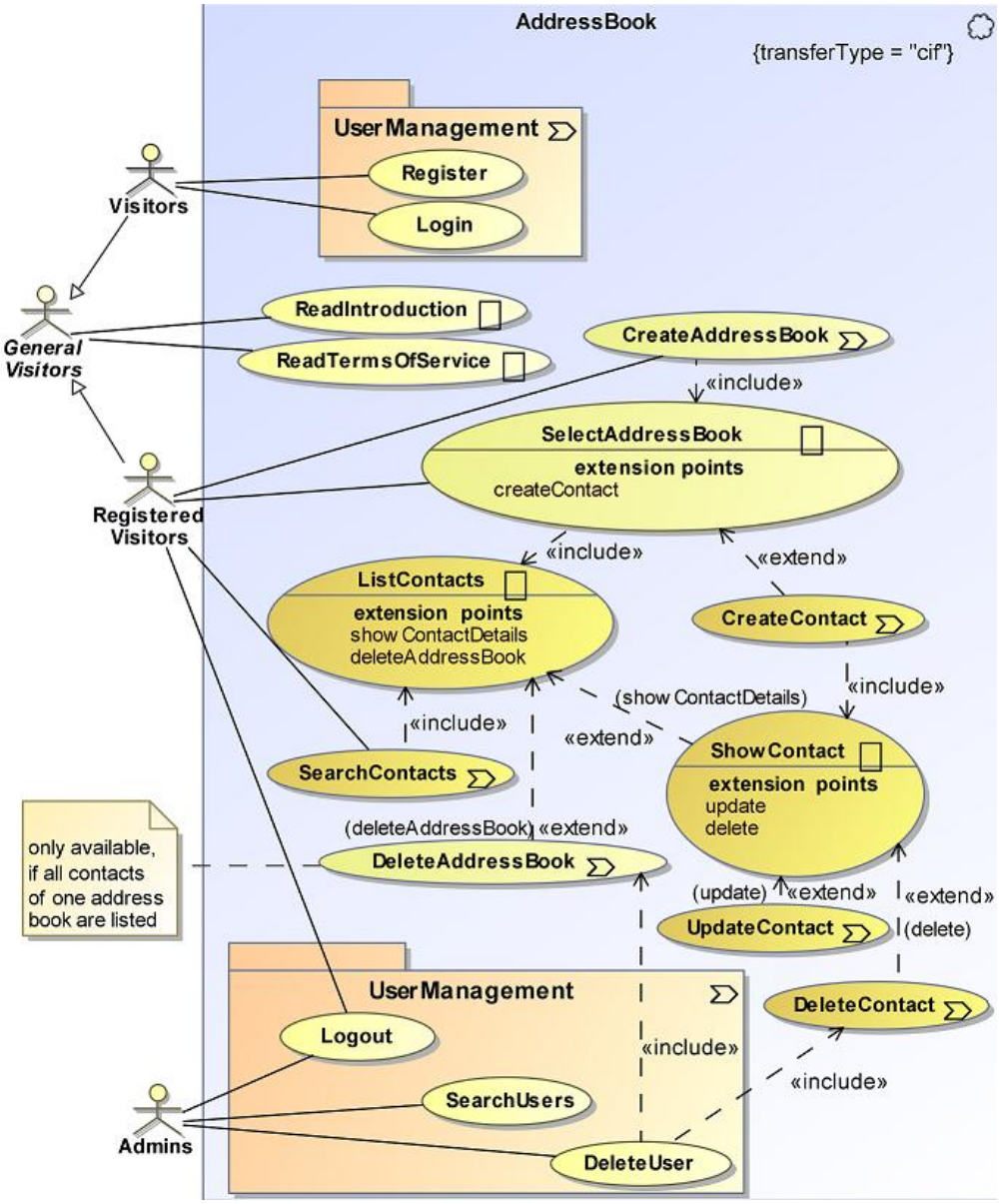

Figura 2 - Diagrama que ilustra a funcionalidade de uma aplicação de gerenciamento de contatos [lmu 2015 1].

A figura 3 mostra o uso do diagrama para indicar quais funcionalidades estarão presentes em diversas versões do sistema, uma informação particularmente importante quando se trabalha com metodologias de desenvolvimento ágil.

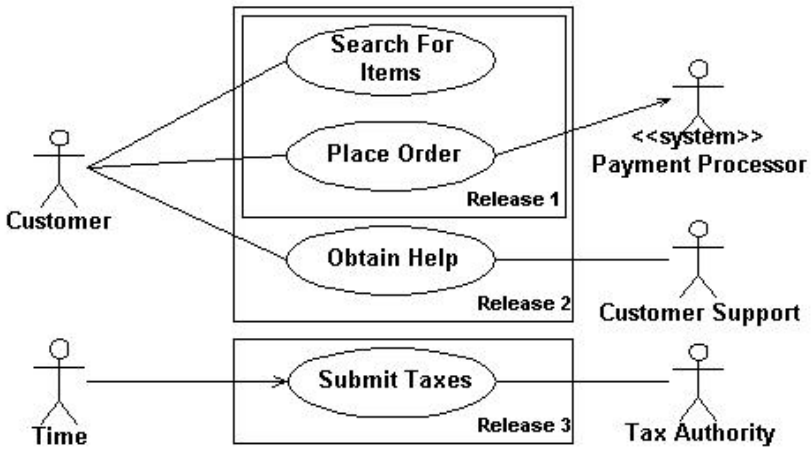

Figura 3 - Diagrama que ilustra funcionalidades de versões diversas de um sistema [agile 2015 1].

Os diagramas de casos de uso são diagramas simples, mas bastante úteis no levantamento dos requisitos, pois permitem dividir o sistema em subsistemas e definir casos de testes já num momento em que poucos detalhes estão disponíveis ao analista. Existem livros inteiramente dedicados ao uso desse tipo de diagrama. A informação mais valiosa, entretanto, que se pode obter por meio da diagramação dos casos de uso, é uma estimativa precoce do esforço necessário para o desenvolvimento. As estatísticas apontam que um sistema de software é descrito por algo entre 20 e 100 casos de uso; o patamar superior corresponde a um esforço de cerca de 10 homens-ano, recomendando-se que projetos maiores do que isso sejam divididos em subprojetos para que um bom gerenciamento seja possível [Podeswa 2005]. Um conjunto de diagramas descreve os diversos **cenários** presentes na aplicação. Um cenário é cada um dos caminhos alternativos possíveis em um diagrama de casos de uso. Com frequência, sistemas similares apresentam cenários similares, e os diagramas criados para modelar um podem ser aproveitados para o outro.
As figuras seguintes apresentam o diagrama de contexto e o diagrama de casos de uso para um mesmo sistema, de forma a facilitar a comparação entre as diferentes ferramentas. Apesar de à primeira vista serem muito similares em sua abordagem, um exame mais atento revela que os diversos cenários possíveis são muito melhor ilustrados no diagrama de casos de uso, apesar de este exemplo específico não lançar mão de todos os recursos disponíveis, como estereótipos e condições [Burge 2015 1].

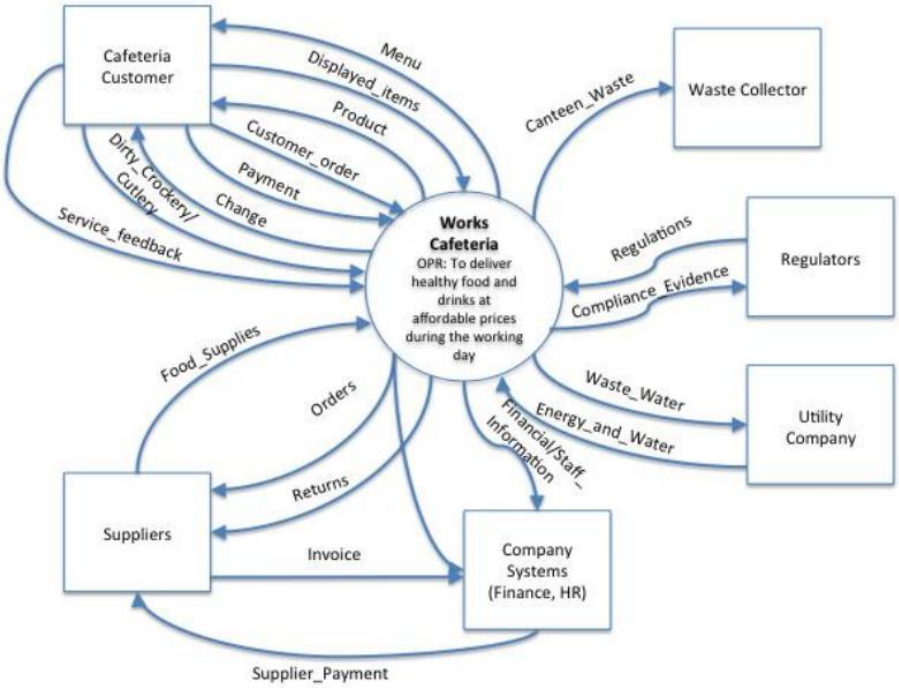

Figura 4 - Diagrama de contexto para sistema de cafeteria [Burge 2015 1].

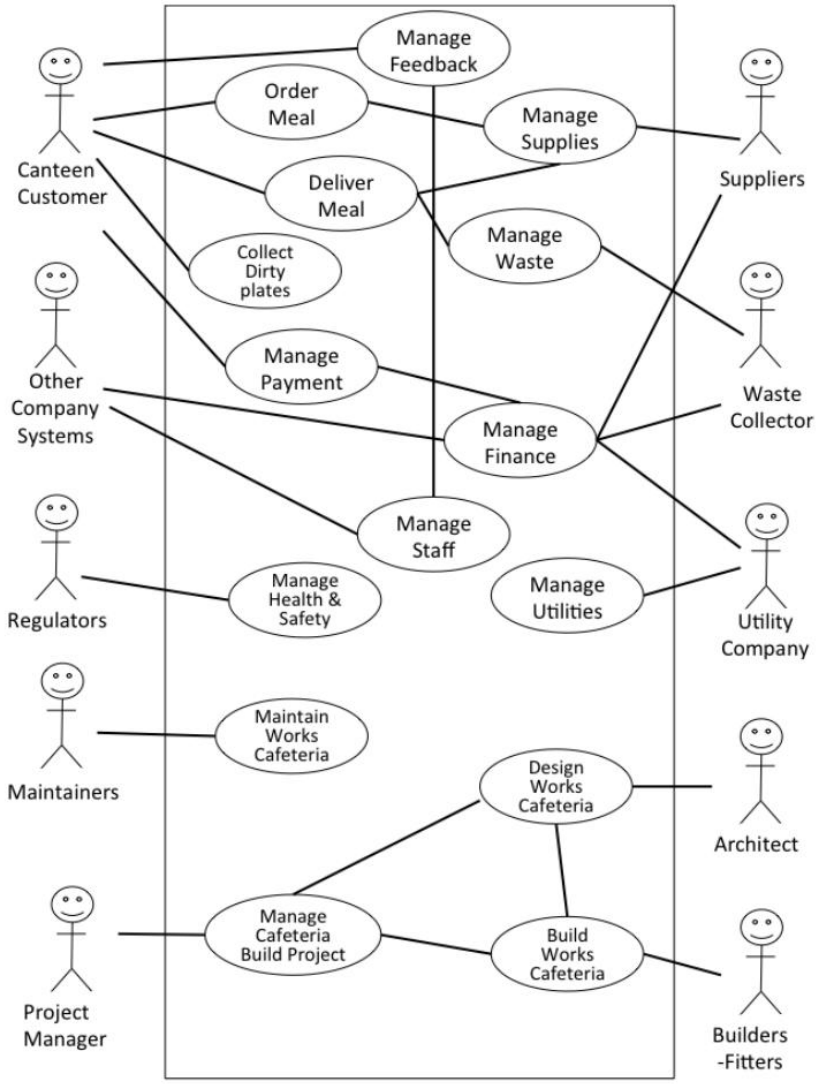

Figura 5 - Diagrama de casos de uso para sistema de cafeteria [Burge 2015 1].

Diagrama de Atividades. Um diagrama de atividades descreve a lógica associada a um caso de uso. Ele usa diversos tipos de artefatos diferentes, como (figura 6) **eventos ocorridos** ("Schedule Printed"), **eventos temporais** ("April 1st"), junção lógica (a barra que recebe eventos como entrada), **atividades** ("Determine Mailing List"), que podem ser detalhados posteriormente em outro diagrama, o símbolo de precedência temporal (a seta), passagem de dados entre atividades ("Mailing List"), passagem de objetos entre atividades ("Labeled Schedules") e o resultado final ("Ready for Mail Pickup") além dos já vistos anteriormente, como notas, estereótipos e condições [agile 2015 2].

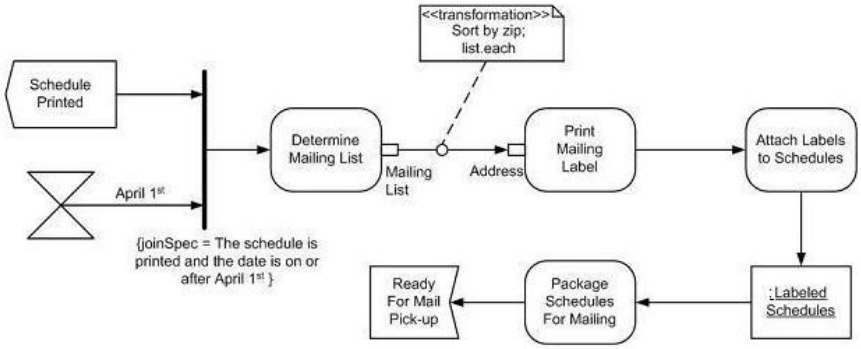

Figura 6 - Diagrama de atividades referente a um sistema de mala direta [agile 2015 2].

Outros símbolos muito comuns nesse tipo de diagrama são: ponto inicial, ponto final e decisão, como aparecem na figura 7. Quando uma atividade é um caso de uso completo, o símbolo é substituído por uma elipse, como no diagrama de casos de uso.

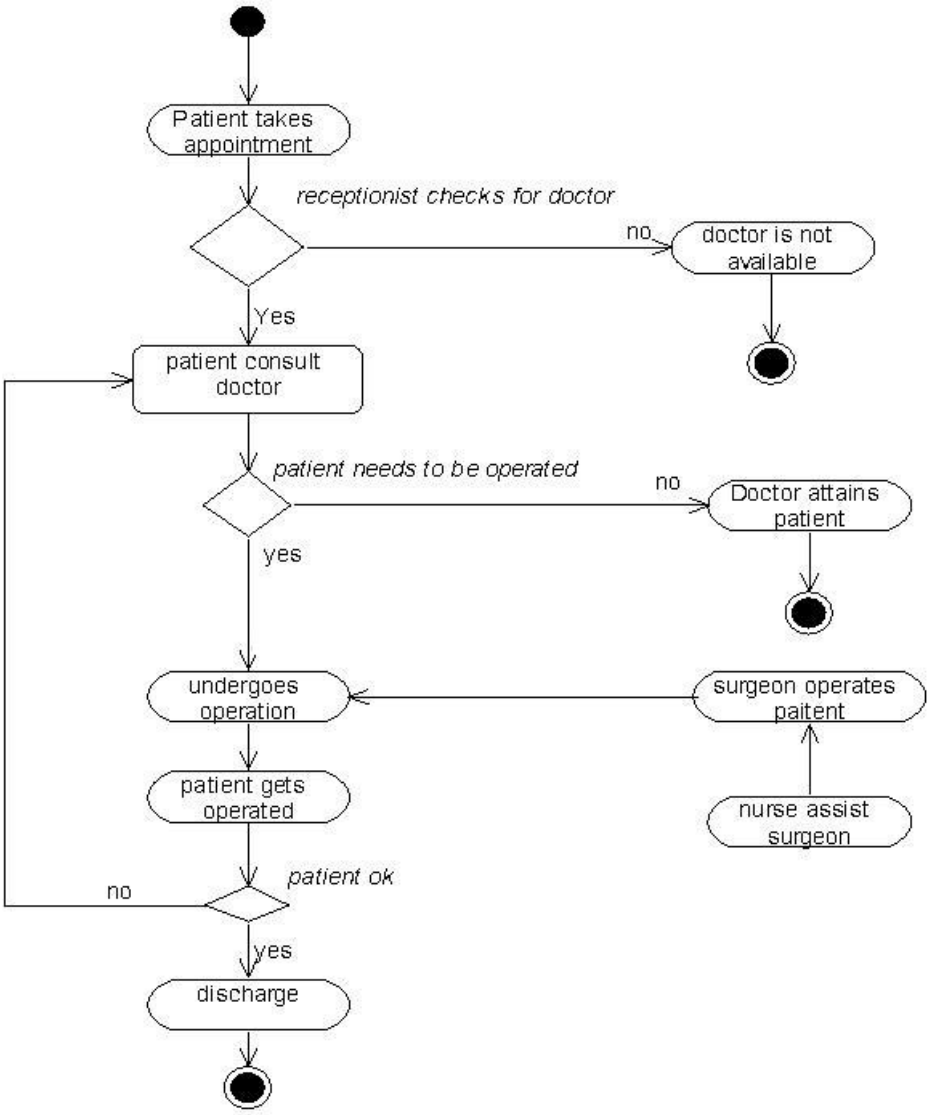

Figura 7 - Diagrama de atividades referente a um sistema de gerenciamento de hospital [agile 2015 2].

O diagrama de atividades pode indicar também, por meio de **raias**, quem é responsável por um processamento (figura 8).

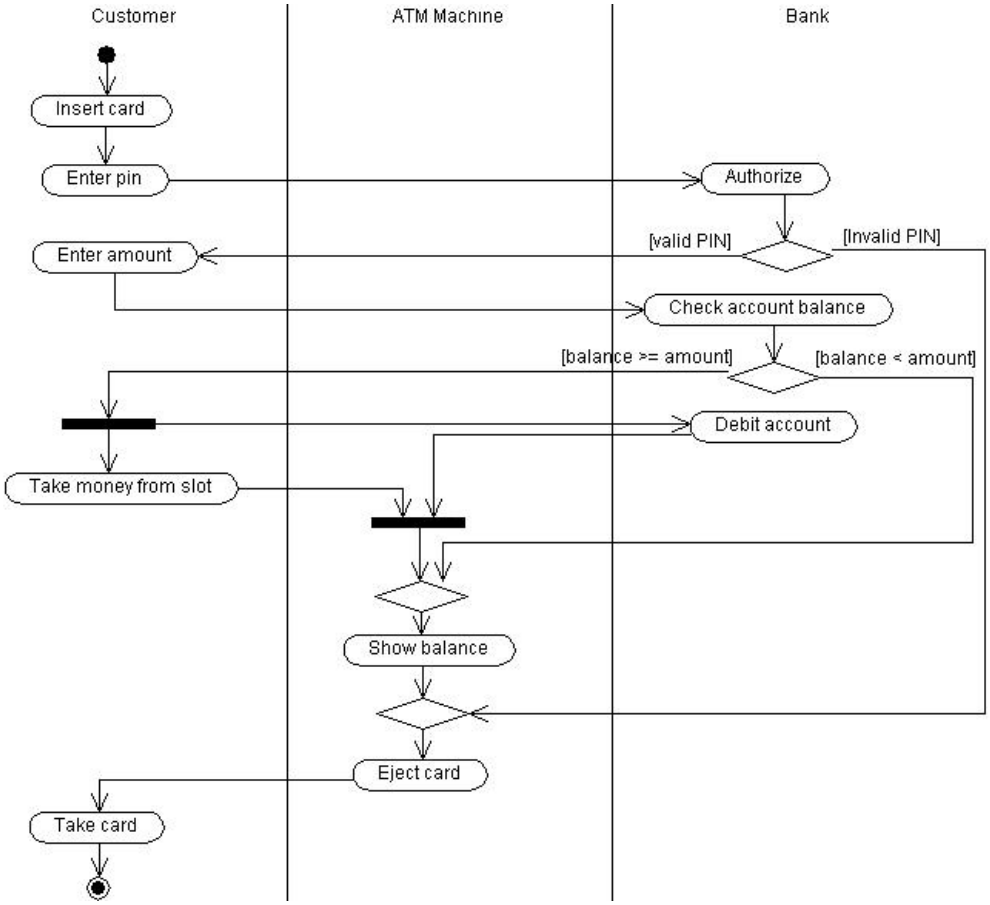

Figura 8 - Diagrama de atividades referente a um caixa eletrônico [embarcadero 2015 1].

Quando comparado a outras ferramentas, como fluxogramas e DFDs, o diagrama de atividades apresenta as vantagens de ser padronizado, o que facilita o uso de ferramentas CASE, ser bastante expressivo, devido ao grande número de artefatos possíveis, e integrar-se às demais ferramentas UML. Esta última característica permite que ele complemente e/ou seja complementado por outros diagramas, como tabelas de decisão, árvores de decisão, diagrama de atores, etc. Apesar do grande avanço que o diagrama de atividades representa, a complexidade e variedade das operações possíveis em um sistema de software ainda não permitiu que ferramentas eletrônicas gerem o código da aplicação a partir dele [Podeswa 2005].

Outros diagramas. Além dos diagramas explorados, a UML oferece outros que, apesar de concebidos para auxiliar as fases posteriores de um projeto, podem ser usados ainda na fase de levantamento para prover alguma informação específica.

Por exemplo, os **diagramas de classes** são muito úteis para modelar os aspectos estáticos de um sistema, e se mostram muito úteis quando a complexidade reside mais no volume e variedade dos dados tratados do que nas operações requeridos, condição em que se encontram muitos sistemas de TI. Nesses casos, é muito comum a situação em que, aparentemente, os requisitos estão claros, mas há uma profunda discordância entre os *stakeholders* e desenvolvedores com relação a certos conceitos. A análise estática é a indicada para resolver esse problema.
Os diagramas de classes podem ser complementados por **diagramas de estrutura** e por **diagramas de objetos** [Podeswa 2005].

Em quê ainda não se avançou

Com relação ao importantíssimo conjunto de atividades referentes aos testes de um sistema, a metodologia disponível atualmente ainda é basicamente a definida em 1976 por Glenford Myers, em sua obre "The Art of Software Testing". Trata-se de uma abordagem estruturada, que define grupos de testes unitários, *black-box*e sistêmicos a serem aplicados, partindo do particular para o geral. Não existem diagramas UML devotados a esse tipo de atividade. No entanto, é muito imporatente que os testes sejam definidos, pelo menos em linhas gerais, nas fases iniciais do desenvolvimento. Neste caso, o conselho disponível dos teóricos é que os testes unitários sejam elaborados a partir dos diagramas de atividades e os testes *black-box*, a partir dos diagramas de casos de uso. Os testes sistêmicos, por sua vez, são genéricos o suficiente para não diferirem muito de um sistema para outro, por isso o gerente do projeto pode lançar mão facilmente da experiência passada, sua ou da equipe, para definí-los com precisão adequada durante a etapa de captura de requisitos [Podeswa 2005].
Uma outra deficiência da UML é que não existem ferramentas para documentar a origem dos requisitos captados. O rastreamento das origens dos requisitos é requerido pela IEEE 830, por exemplo, e é uma atividade de alto custo e complexidade. O gerente deverá, para essa atividade, valer-se de anotações cuidadosas ou, se possível, de um sistema de informação específico para registro da genealogia dos requisitos [Leffingwell e Widrig 1999].

Referências

[agile 2015 1] AGILE MODELING, **UML 2 Use Case Diagramming Guidelines**. Disponível em `http://www.agilemodeling.com/style/useCaseDiagram.htm`, acesso em 08/08/2015.

[agile 2015 2] AGILE MODELING, **UML 2 Activity Diagrams: An Agile Introduction**. Disponível em `http://www.agilemodeling.com/artifacts/activityDiagram.htm`, acesso em 08/08/2015.

[Burge 2015 1] STUART BURGE, **The Systems Engineering Tool Box**. Disponível em http://www.burgehugheswalsh.co.uk/Uploaded/1/Documents/CD-Tool-Box-V1.0.pdf, acesso em 08/08/2015.

[embarcadero 2015 1] EMBARCADERO, **Practical UML: A Hands-On Introduction for Developers**. Disponível em http://edn.embarcadero.com/print/31863, acesso em 08/08/2015.

[ibm 2015 1] IBM, **IBM Rational Software**. Disponível em http://www.ibm.com/software/rational, acesso em 08/08/2015.

[iso 2005 1] INTERNATIONAL STANDARDS ORGANIZATION, **ISO/IEC 19501:2005 - Information technology - Open Distributed Processing - Unified Modeling Language (UML) Version 1.4.2**. Disponível em http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=32620, acesso em 08/08/2015.

[Leffingwell e Widrig 1999] DEAN LEFFINGWELL e DON WIDRIG, **Managing Software Requirements**, Addison Wesley, 1999, , ISBN 0-201-61593-2.

[lmu 2015 1] LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN, **UWE – UML-based Web Engineering - Example: Secure Address Book**. Disponível em http://uwe.pst.ifi.lmu.de/examples/SecureAddressBook/diagrams, acesso em 08/08/2015.

[modernanalyst 2015 1] MODERNANALYST.COM, **What is a Context Diagram and what are the benefits of creating one?**. Disponível em http://www.modernanalyst.com/Careers/InterviewQuestions/tabid/128/ID/1433/What-is-a-Context-Diagram-and-what-are-the-benefits-of-creating-one.aspx, acesso em 08/08/2015.

[omg 2015 1] OBJECT MANAGEMENT GROUP, **Unified Modeling Language (UML) Resource Page**. Disponível em http://www.uml.org/, acesso em 08/08/2015.

[omg 2015 2] OBJECT MANAGEMENT GROUP, **(Versão corrente da UML)**. Disponível em http://www.omg.org/spec/UML/Current, acesso em 08/08/2015.

[pmi 2015 1] PROJECT |MANAGEMENT INSTITUTE, **Develop Project Charter Data Flow Diagram**. Disponível em http://marketplace.pmi.org/Pages/ProductDetail.aspx?GMProduct=00101142201, acesso em 08/08/2015.

[Podeswa 2005] Howard Podeswa, **PUML for the IT Business Analyst**, Thomson Course Technology, 2005, Boston, ISBN 1-59200-912-3.

[scol 2015 1] SOURCE CODE SOLUTIONS BLOG, **UML Online Hospital Management System**. Disponível em http://sourcecodesonline.blogspot.com.br/2011/04/uml-online-hospital-management-system.html, acesso em 08/08/2015.

[sparx 2015 1] SPARX SYSTEMS, **The Use Case Model**. Disponível em http://www.sparxsystems.com/resources/tutorial/use_case_model.html, acesso em 08/08/2015.

[Wikipedia 2015 1] WIKIMEDIA FOUNDATION, **List of Unified Modeling Language Tools**. Disponível em http://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools, acesso em 08/08/2015.

[Yourdon 2015 1] EDWARD YOURDON, **Major Issues in Systems Development**. Disponível em http://www.yourdon.com/strucanalysis/wiki/index.php/Chapter_6, acesso em 08/08/2015.

[Yourdon 2015 2] EDWARD YOURDON, **Changes in Systems Analysis**. Disponível em http://www.yourdon.com/strucanalysis/wiki/index.php/Chapter_7, acesso em 08/08/2015.

[Yourdon 2015 3] EDWARD YOURDON, **Data Flow Diagrams**. Disponível em http://www.yourdon.com/strucanalysis/wiki/index.php/Chapter_9, acesso em 08/08/2015.