Faculdade XPe

Capítulo 04

MODELAGEM DE SISTEMAS

Vinicius Fernandes de Almeida

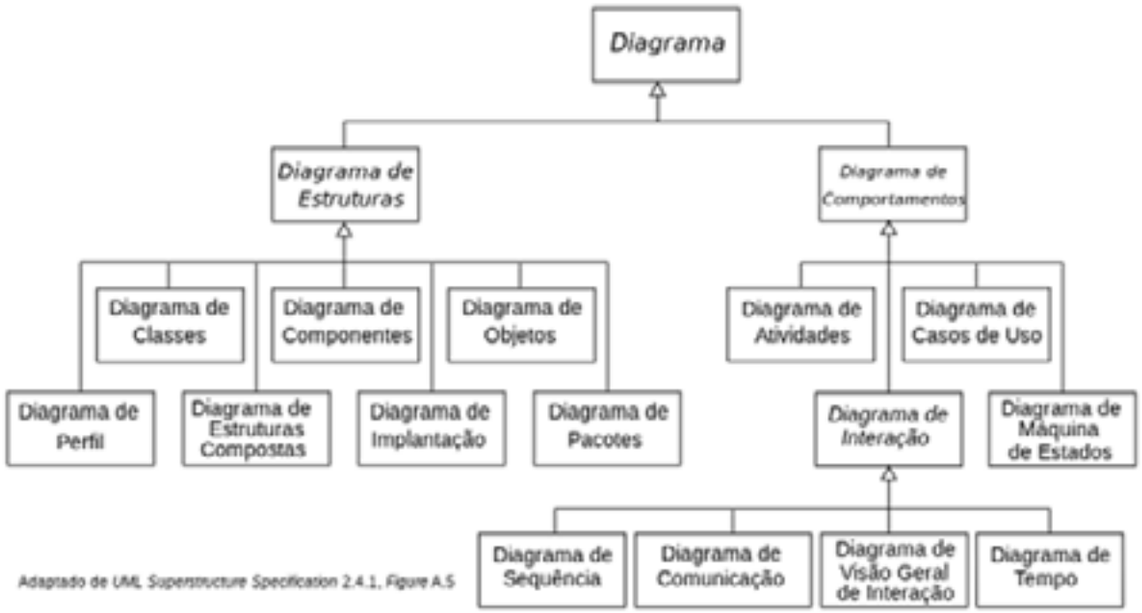
2024

Como vimos até o momento, o principal objetivo de um time, sendo ele desenvolvedores, lideres, cargos de coordenação e regra de negócios, antes mesmo de gerar modelos gráficos que retratem a realidade do que o sistema virar a ser, é a construção de modelos de entendimento de como o software deve ser. Para chegar no produto desejado com qualidade, no tempo e custos, temos alguns artifícios que nos auxiliam neste processo. Assim, a modelagem não é um produto final, mas sim o meio adequado para a construção de um sistema.

Unified Modeling Language (UML)

Unified Modeling Language ou apenas UML é uma linguagem que traz sentido de "permitir comunicação" para visualizar, especificar, construir, documentar, entender fluxos e sistemas. Ele surgiu na década de 1990 com a unificação de métodos no BOOCH (Object Oriented Software Engenearing ou OOSE, e Object Modeling Technique ou OMT) - tendo como mentores os renomados Grady Boosh, Ivar Jocobson e James Rumbaugh (France; Koryn, 2001). Sendo assim, tornou-se referência para modelagem de sistemas orientados a objetos e é muito utilizado em várias ferramentas.
Fato é que o UML apresenta vários tipos de diagramas para especificar aspectos dinâmicos que vão dizer como o sistema deve se comportar. A questão que o UML não oferece um "caminho das pedras", mas sim formas de se visualizar o que precisa. Podemos dividir em 02 (dois) grandes grupos os processos de modelagem, os modelos comportamentais que são aspectos dinâmicos e funcionais. Assim como modelagem de aspectos estruturais.

Figura 6 - Árvore de Diagramas



Fonte: google.com

Em sua concepção, um digrama é uma representação gráfica de informações, processos, sistemas e tudo o que pode ser colocado em forma visual. Normalmente utilizado para ilustrar ou exemplificar contextos, regras de negócios, fluxos de trabalho e afins. São muito utilizados em várias áreas como engenharia, tecnologia, negócios e outros. Como mostrado na ilustração acima, são divididos por forma de representação e também possuem uma estrutura hierárquica.
Acaba que muitos dos diagramas podem conter características um dos outros porque alguns possuem certo grau de complexidade menor e também por estarem dentro da mesma árvore hierárquica. Contudo, cada um tem sua função e finalidade.

Diagramas Comportamentais

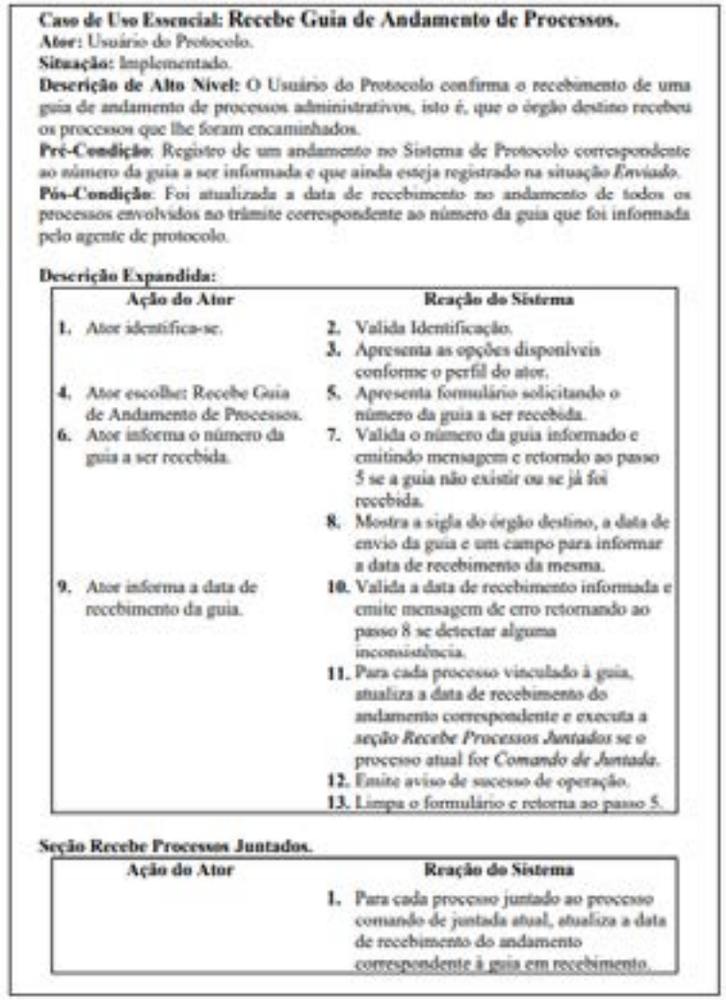
Na modelagem computacional nós temos um sistema que consiste na representação da dinâmica e das funcionalidades do sistema, onde damos o nome de modelagem comportamental. Os aspectos dinâmicos de um sistema têm por representação como ele irá responder com base em estímulos ou eventos vindos de forma externa a ele. Enquanto no de funcionalidade, está relacionado ao fluxo de atividades e/ou operações a serem tomadas como: decisão, iterações, desvios, recursividade, dentre outros. Em outras palavras, os aspectos dinâmicos representam “o que” o sistema deve fazer, em contrapartida, os funcionais descrevem “como” deve-se fazer.

As ferramentas de UML apresentam alguns modelos para ambos os casos. Para os aspectos dinâmicos, temos:

- Caso de Uso;
- Diagrama de Interação (Sequência e de Colaboração);
- Diagrama de Transição de Estados.
- Para os aspectos funcionas, destacamos:
- Discrição textual de Caso de Uso;
- Diagrama de Atividades.

Dentre as possibilidades que temos tanto para aspectos dinâmicos ou funcionais, o Caso de Uso é um diagrama ou descritivo contextual muito utilizado. Eles se mostram uma excelente ferramenta de comunicação entre os envolvidos. Os casos de uso podem ser especificações de outros casos de uso também, tendo relação direta e apontamento.

Figura 7 - Descrição Textual de Caso de Uso



Caso de Uso Essencial: Recebe Guia de Andamento de Processos.
Ator: Usuário do Protocolo.
Situação: Implementado.
Descrição de Alto Nível: O Usuário do Protocolo confirma o recebimento de uma guia de andamento de processos administrativos, isto é, que o órgão destino recebeu os processos que lhe foram encaminhados.
Pré-Condição: Registro de um andamento no Sistema de Protocolo correspondente ao número da guia a ser informada e que ainda esteja registrado na situação *Enviado*.
Pós-Condição: Foi atualizada a data de recebimento no andamento de todos os processos envolvidos no trâmite correspondente ao número da guia que foi informada pelo agente de protocolo.

Descrição Expandida:

Ação do Ator	Reação do Sistema
1. Ator identifica-se.	2. Valida Identificação. 3. Apresenta as opções disponíveis conforme o perfil do ator.
4. Ator escolhe: Recebe Guia de Andamento de Processos. 6. Ator informa o número da guia a ser recebida.	5. Apresenta formulário solicitando o número da guia a ser recebida. 7. Valida o número da guia informado e emitindo mensagem e retornado ao passo 5 se a guia não existir ou se já foi recebida. 8. Mostra a sigla do órgão destino, a data de envio da guia e um campo para informar a data de recebimento da mesma.
9. Ator informa a data de recebimento da guia.	10. Valida a data de recebimento informada e emite mensagem de erro retornando ao passo 8 se detectar alguma inconsistência. 11. Para cada processo vinculado à guia, atualiza a data de recebimento do andamento correspondente e executa a *seção Recebe Processos Juntados* se o processo atual for *Comando de Juntada*. 12. Emite aviso de sucesso de operação. 13. Limpa o formulário e retorna ao passo 5.

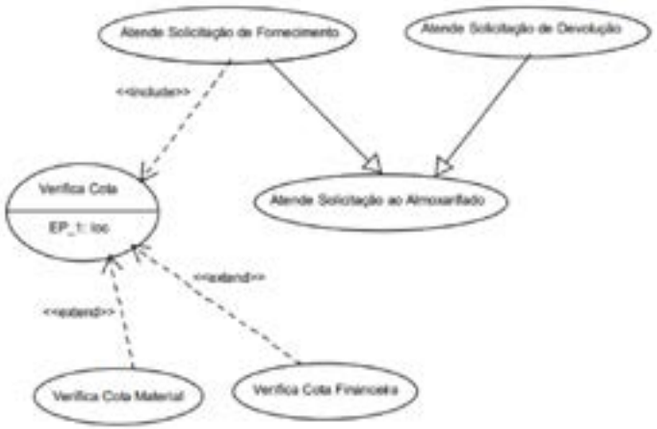
Seção Recebe Processos Juntados.

Ação do Ator	Reação do Sistema
	1. Para cada processo juntado ao processo comando de juntada atual, atualiza a data de recebimento do andamento correspondente à guia em recebimento.

Fonte: google.com

Quando formos especificar a dinâmica de caso de uso de um sistema, utilizaremos o diagrama de caso de uso com seus relacionamentos. Na ilustração a seguir temos note que temos direção de sentidos com <<include>> e outros com sentido de <<extend>>. No exemplo da inclusão um caso de uso compõe o outro, no caso de estender o caso de uso é continuação do fluxo.

Figura 8 - Relacionamento de Caso de Uso

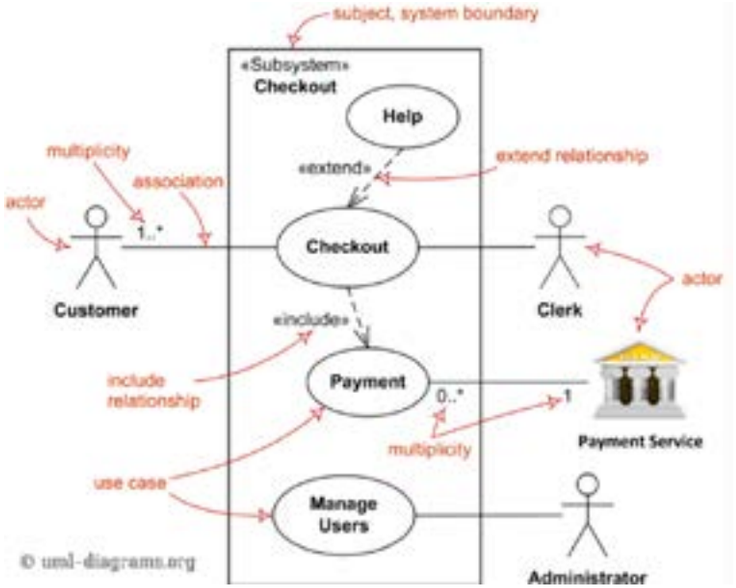


Fonte: google.com

No diagrama o “Atende Solicitação Almoxarifado” é a forma generalista dos casos de uso especificados “Solicitação de Devolução” e “Atende Solicitação de Fornecimento”. Por sua vez, o “Atende Solicitação de Fornecimento” está incluído no caso de uso “Verifica Cota”. Os casos “Verifica Cota Material” e “Verifica Cota Financeira” são estendidas de “Verifica Cota”.
Um ou mais pacotes podem estar na mesma representação. A ideia de realizar essa separação se dá para separar os sistemas em subsistemas e facilitar o entendimento. Temos algumas regras e recomendações para a construção de um diagrama de caso de uso (Boooch; Rumbaugh; Jacobson, 1999):

- Deve ter nome;
- Minimizar o cruzamento de linhas;
- Deve conter certo nível de abstração;
- Serem dispostos perto quando houver relação;
- Deve apresentar apenas o agente e seu caso para o entendimento;
- Não ter especificação excessiva e não deixar de especificar o importante;
- Deve estar focado em um aspecto do sistema, seu próprio escopo.

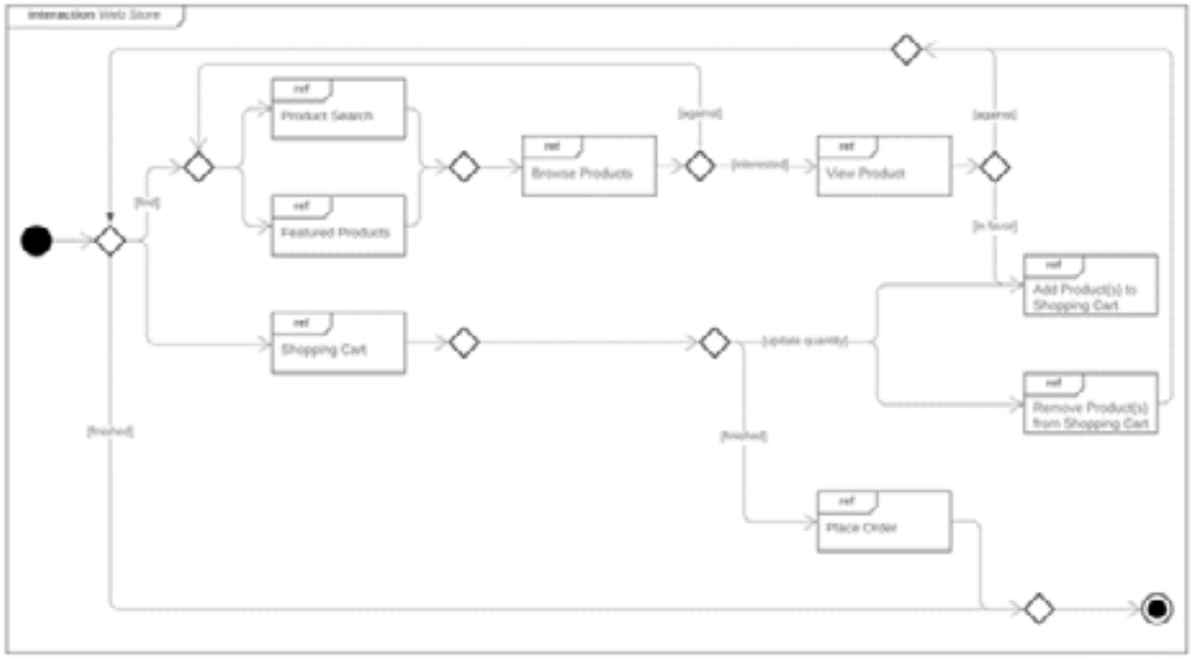
Figura 9 - Diagrama de Caso de Uso



Fonte: google.com

Nos diagramas de interações, eles especificam os relacionamentos entre os objetos com base na sua estrutura organizacional visando as mensagens emitidas e recebidas por eles. Sendo assim, temos uma ideia de ordenação temporal onde pode resultar em possíveis mudanças de estado ao longo do processo.

Figura 10 - Diagrama de Interação



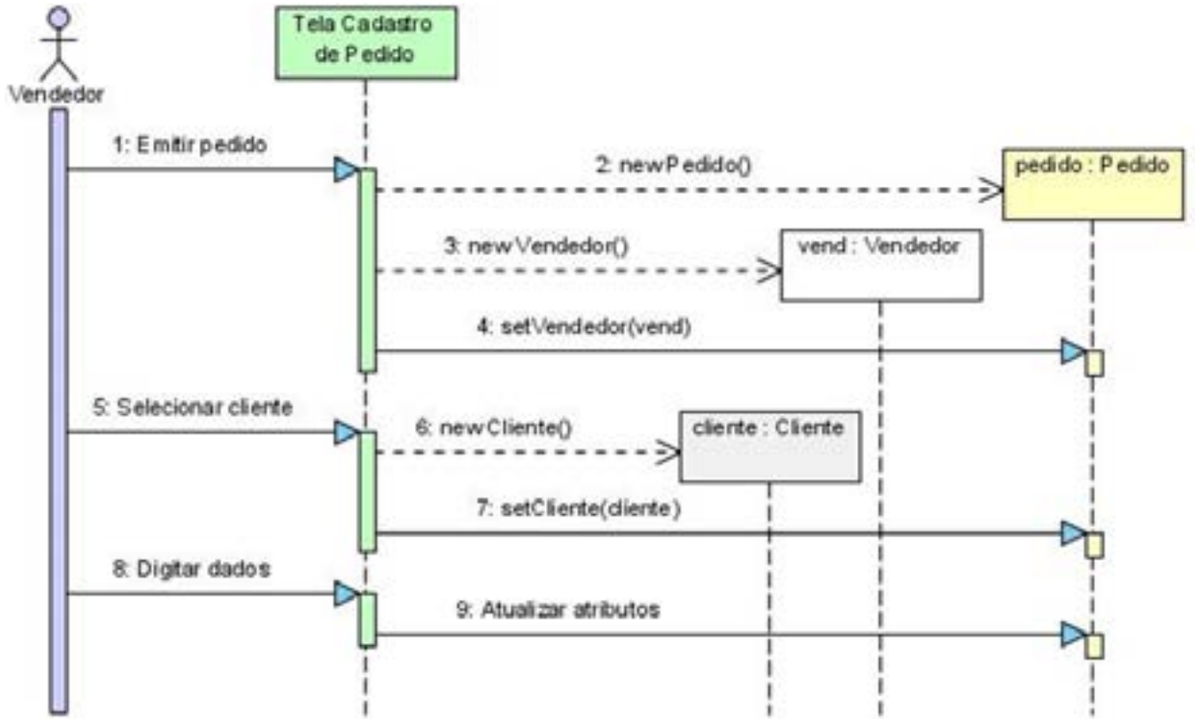
Fonte: google.com

No diagrama de sequência, além de ter o entendimento do que tem interações, também temos as especificações de sequência que elas ocorrem. É bastante difundida para apresentação de interações entre o usuário e o sistema. Para isso são construídos diagramas com atores.

Pontos importantes que devemos levar em consideração na hora de criação do diagrama de sequência:

- Definir contexto da interação;
- Informar os parâmetros das mensagens;
- Dividir o diagrama em complexabilidade;
- Apresentação é da esquerda para a direita;
- Informar pós e pré-condições das mensagens;
- Representar a ordenação temporal das mensagens;
- Para os objetos que são criados e destruídos, utilizar as notações <<create>> e <<destroy>>.

Figura 11 - Diagrama de Sequência.



Fonte: google.com

Diagrama de colaboração, também conhecido como diagrama de comunicação, é um tipo de diagrama de UML onde visa salientar a interação entre objetos de um sistema. Mostrando assim a colaboração e troca de informações entre eles. Também necessita de uma ordem cronológica para completude de sentido. Destacamos neste diagrama:

- Dar nome ao diagrama;
- Estabelecer contexto;
- Colocar as classes como vértices em um grafo;
- Desenhar as ligações por onde vão fluir;
- Colocar a mensagem que inicia a interação;
- Informar se a ligação é local ou global;
- Numerar a sequência das interações.

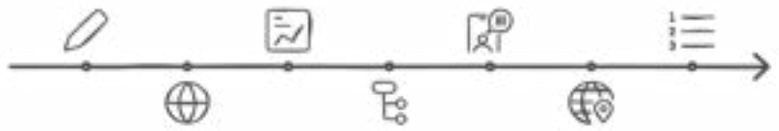
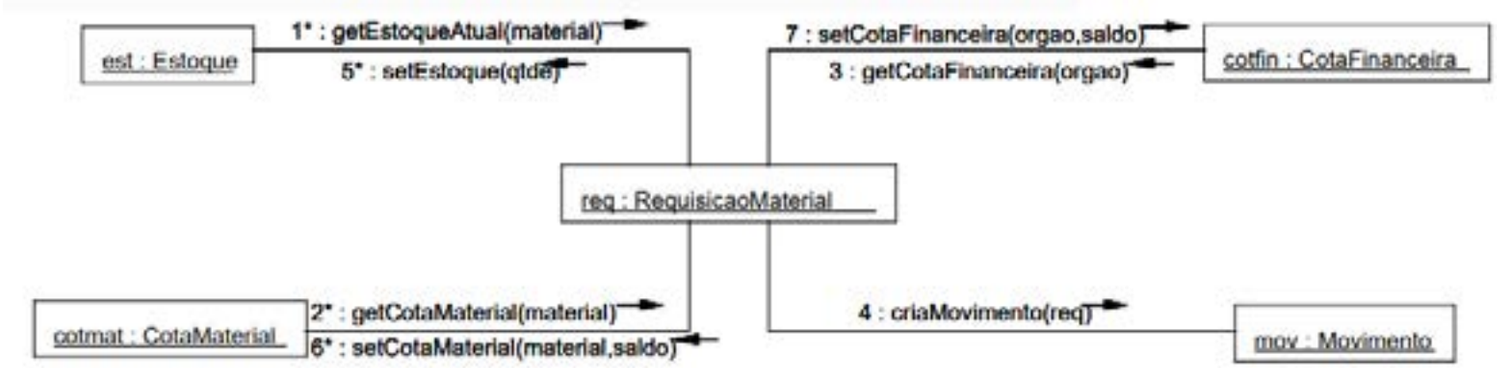


Figura 12 - Diagrama de Colaboração



Fonte: google.com

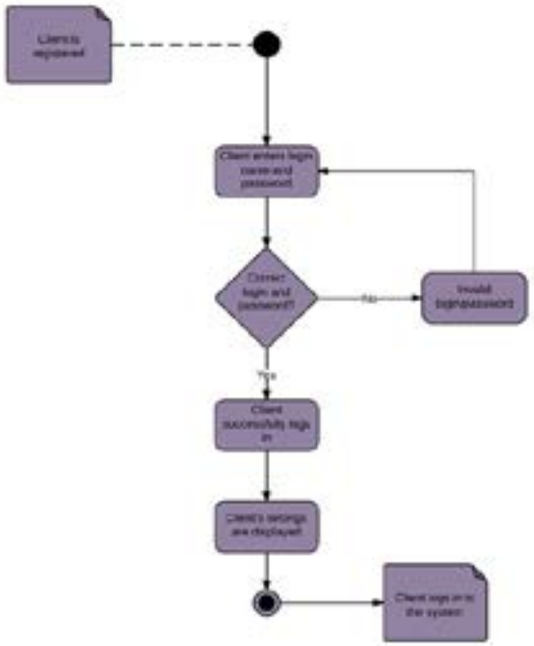
Dando continuidade, o diagrama de transição de estados lembra bastante a sequencial porque também tem uma ordem cronológica de acontecimentos, porém traz em sua peculiaridade, os estados compostos possíveis para cada situação. Cada estado tem uma condição e essa condição quando alterada resulta em outro estímulo que assim seguem em cadeia. Os estados são representados por retângulos e as transições por setas. Ele é utilizado mais especificamente para sistemas que tem esse tipo de comportamento com estados. Ajuda a identificar problemas com lógica e validar o comportamento do sistema.

Figura 13 - Diagrama de Transição de Estados

Fonte: google.com

Sobre o diagrama de atividades, ele pode ser também utilizado para processo de negócios assim como de sistemas. Representa as tapas ou por sua vez atividades necessárias para executar uma tarefa ou fluxo. Podemos incluir algumas notações no diagrama como pontos de tomada de decisão, fluxos condicionais, forks, joins, atividades paralelas ou sincronizadas. Ajuda a identificar pontos críticos em processos, bem como otimizar o desempenho e melhorar a eficácia do sistema.

Figura 14 - Diagrama de Atividades



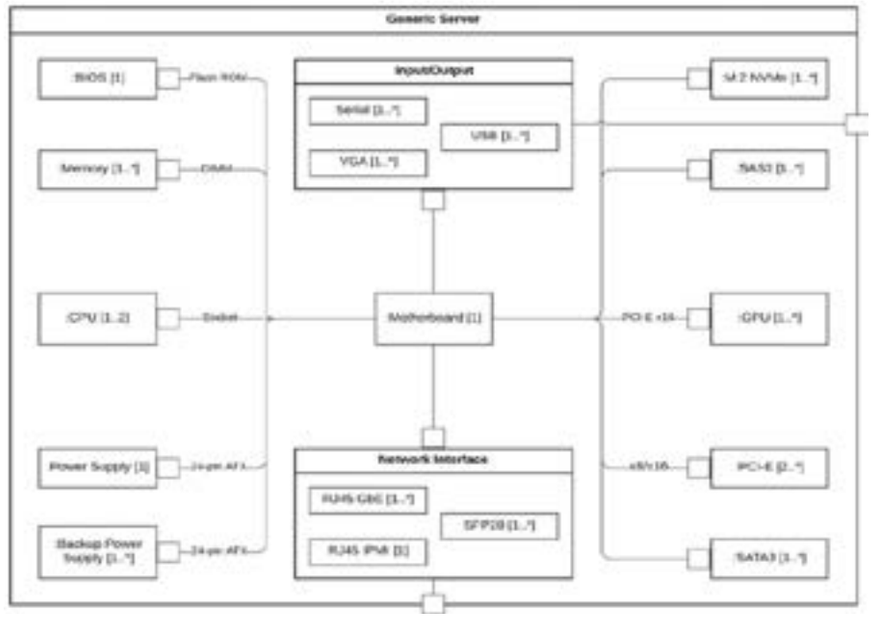
Fonte: google.com

Diagramas Estruturais

Diagramas de estrutura, como o próprio nome já diz, são modelagens de sistemas que descrevem estruturas de sistemas ou de parte do todo. Podem ser de parte física, equipamentos, componentes, como as que vimos até aqui no modelo lógico com as classes em orientação a objeto. Podemos dividir em diagramas de componentes que de fato é a parte física, pacotes que representam a estrutura do sistema, objetos que mostra a relação em um momento de execução, dentre outros.

Para o diagrama de componentes, podemos ir mais além do que somente representar partes físicas como representado na figura abaixo. Obviamente que parte da sua característica é esse tipo de representação. Contudo, podemos representar uma unidade funcional do sistema, uma interface como uma porta de comunicação entre componentes. Eleva o nível de representação de arquitetura. Como todo bom diagrama facilita a identificação de problemas e designer. Servindo também como documentação para manutenções futuras.

Figura 15 - Diagramas de Componente

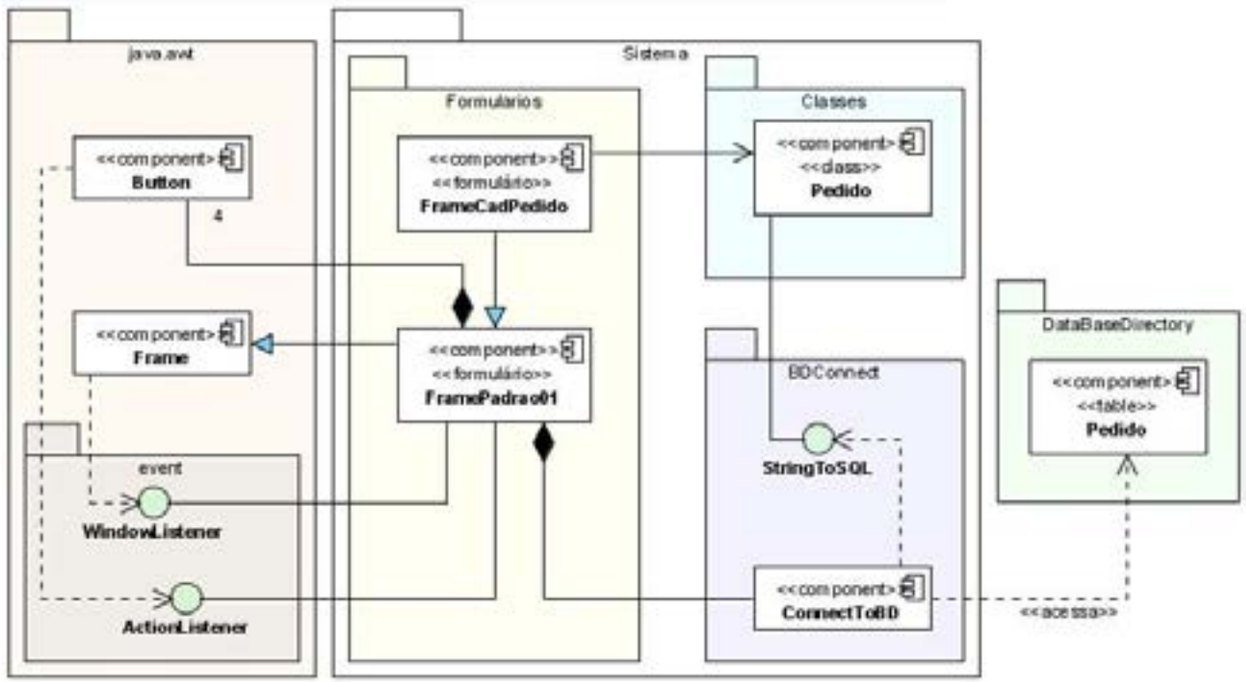


Fonte: google.com

Diferentemente do diagrama de estrutura, a estrutura composta visa criar uma modelagem de sistemas para representação interna de objetos, no caso quanto um objeto é composto por outro ou gera interação com outro. Vamos tomar de exemplo um sistema de controle de estoque. O objeto produto pode ser composta por outros objetos como valor, quantidade, entre outros. Podemos usar também notações para identificar cada parte como <<component>>, <<class>> e <<table>>.
Dentro dessa estrutura podemos vislumbrar:

- Atributos;
- Operações;
- Partes dos objetos;
- Conectores entre os pares.

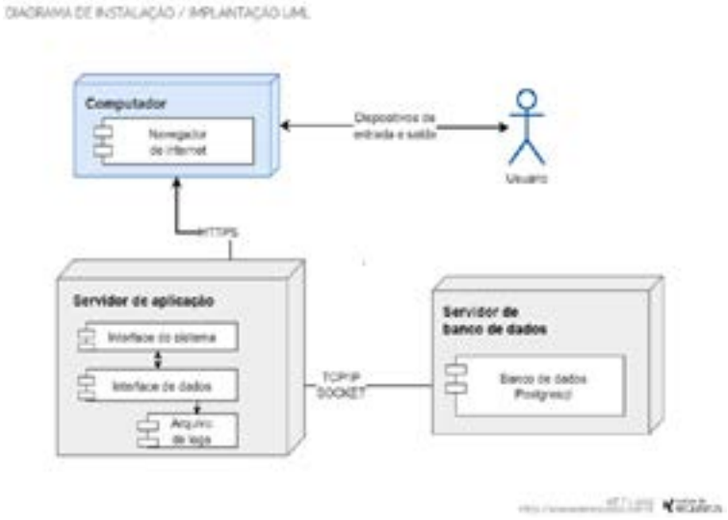
Figura 16 - Diagrama de Estrutura Composta



Fonte: google.com

Diagramas de implantação costumam apresentar componentes de software e hardware. Sempre tendo em mente o grau de granularidade que você presente apresentar, com base na relevância dentro do escopo. Contudo, visa mostrar o relacionamento entre partes, como exemplificado na figura abaixo.

Figura 17 - Diagrama de Implantação

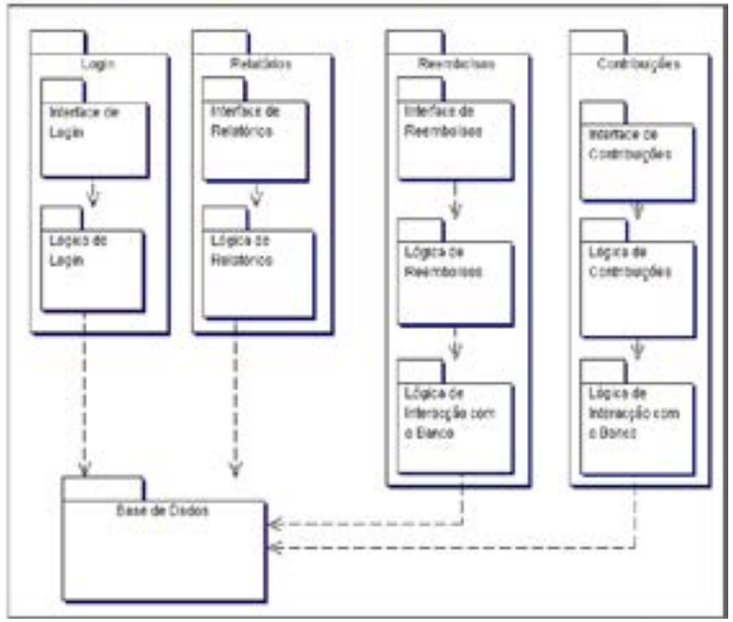


Fonte: google.com

Voltado para estruturar como o sistema está dividido em sua estrutura de pastas. Todo modelo de desenvolvimento de software segue padrões. Muitos destes padrões também obedecem relação hierarquia de pacotes para facilitar no entendimento e separar os domínios da aplicação.
Podemos ter no diagrama de pacotes:

- As dependências entre os pacotes;
- Os relacionamentos entre as classes;
- Classes, interfaces e enumerações em cada pacote.

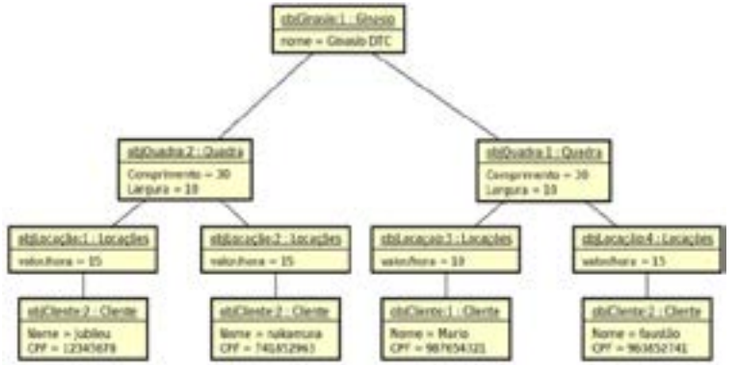
Figura 18 - Diagrama Pacotes



Fonte: google.com

Como era de se imaginar, o diagrama de objetos é muito difundido em orientação a objeto. Exemplifica o relacionamento entre os objetos. Neles encontramos nome da classe, atributos, métodos, tipo de relacionamento, tipo de classe e tudo mais que for relevante a um objeto. Em linhas gerais, ajuda a entender o comportamento dos objetos, especialmente aqueles com múltiplas instâncias e interconectadas.

Figura 19 - Diagrama de Objetos



Fonte: google.com

Modelo 4+1 de Visão Arquitetural

Um modelo arquitetural é uma visão abstrata da realidade. Fazemos modelos para capturar os elementos mais importantes a fim de termos uma boa discussão e tomarmos decisões para os projetos e produtos. Fazemos isso pois seria difícil discutir considerando TODOS os aspectos envolvidos.
Usar modelos nos ajuda a entender os problemas e estruturar as soluções, experimentar possíveis alternativas, simplificar questões complexas, identificar riscos e, com isso, evitar erros.
Apesar de ser importante trabalhar com modelos, precisamos organizá-los para podermos ter a visão que queremos do sistema, mas que não se tornem extremamente complexos a ponto de não ser possível usá-los.
Pensando nisso, foi proposto o Modelo 4+1 de visões arquiteturais:

Figura 20 - Modelo 4+1

Fonte: google.com

O Modelo 4+1 de Visão Arquitetural é uma abordagem utilizada para descrever a arquitetura de sistemas de software complexos. Proposto por Philippe Kruchten em 1995, esse modelo organiza a descrição da arquitetura em cinco diferentes visões, sendo quatro visões principais (lógica, de desenvolvimento, de processos, física) e uma visão adicional (cenários), que serve para validar as outras quatro.

As Cinco Visões do Modelo 4+1

1. Visão Lógica:

- **Descrição:** foca nos aspectos funcionais do sistema, ou seja, na organização dos componentes de software que suportam as funcionalidades principais. É destinada principalmente aos desenvolvedores, mostrando como o sistema é decomposto em módulos e como esses módulos interagem.
- **Aplicabilidade:** essencial para garantir que o sistema atenda aos requisitos funcionais especificados, permitindo uma visão clara da estrutura interna.

2. Visão de Desenvolvimento:

- **Descrição:** trata da organização estática do software em termos de módulos, camadas, pacotes e componentes de código. Enfatiza a estrutura do sistema a partir da perspectiva dos desenvolvedores, detalhando a organização dos arquivos de código-fonte, bibliotecas e outros artefatos.
- **Aplicabilidade:** facilita o gerenciamento da complexidade do desenvolvimento, auxiliando na alocação de tarefas entre as equipes de desenvolvimento e na manutenção do software.

3. Visão de Processos:

- **Descrição:** enfatiza os aspectos dinâmicos do sistema, como o comportamento em tempo de execução, concorrência, threads e processos. Mostra como os componentes do sistema interagem durante a execução, descrevendo os principais processos e a comunicação entre eles.
- **Aplicabilidade:** crucial para a análise de desempenho, escalabilidade e integridade do sistema, especialmente em sistemas distribuídos ou com alta demanda de processamento.

4. Visão Física:

- **Descrição:** trata da topologia física da implementação, incluindo servidores, dispositivos de rede, e a distribuição dos componentes de software no hardware. Representa a arquitetura em termos de nós físicos, como servidores e sua conexão.
- **Aplicabilidade:** importante para a infraestrutura de TI, garantindo que a arquitetura do sistema seja implementável no ambiente físico, levando em conta fatores como desempenho, disponibilidade e resiliência.

5. Visão de Cenários (ou Casos de Uso):

- **Descrição:** utiliza cenários ou casos de uso para ilustrar e validar a arquitetura, mostrando como as outras quatro visões se integram para atender aos requisitos funcionais e não funcionais do sistema. Esses cenários geralmente são representados por diagramas de sequência ou narrativas detalhadas.
- **Aplicabilidade:** serve como um meio de verificação e validação, garantindo que todas as visões da arquitetura se alinhem para suportar as funcionalidades do sistema de forma coesa e eficiente.

Aplicabilidade do Modelo 4+1

O Modelo 4+1 é amplamente utilizado em projetos de sistemas complexos onde é necessário comunicar de forma clara e detalhada a arquitetura para diferentes partes interessadas, como desenvolvedores, arquitetos de sistemas, engenheiros de rede e gerentes de projeto. A divisão em diferentes visões permite que cada grupo de interessados se concentre nos aspectos mais relevantes para o seu papel, ao mesmo tempo em que proporciona uma visão integrada do sistema como um todo.

Além disso, o modelo é útil para a documentação e manutenção da arquitetura do sistema ao longo do tempo, facilitando a compreensão e a modificação da arquitetura à medida que o sistema evolui.

Exemplo Real de Utilização

Vamos considerar uma empresa que desenvolve uma plataforma de e-commerce com alta demanda, acessada por milhares de usuários simultaneamente, com funcionalidades como busca de produtos, carrinho de compras, processamento de pagamentos e interface de administração.

1. **Visão Lógica:** a arquitetura lógica poderia ser dividida em módulos como Gerenciamento de Produtos, Gerenciamento de Usuários, Sistema de Pagamentos e Módulo de Busca. Cada módulo seria responsável por uma parte da funcionalidade do sistema e poderia ser implementado como um conjunto de serviços ou componentes.
2. **Visão de Desenvolvimento:** a visão de desenvolvimento detalharia como os módulos são organizados no código-fonte. Por exemplo, o Módulo de Busca poderia ser implementado em uma estrutura de pacotes contendo classes responsáveis por consultas, indexação e caching de produtos. Os artefatos seriam organizados de forma a facilitar a escalabilidade e manutenção.
3. **Visão de Processos:** esta visão mostraria como o sistema lida com pedidos simultâneos, como a busca é realizada, como os processos de pagamento são tratados de forma concorrente e segura, e como os dados dos usuários são gerenciados em tempo real.
4. **Visão Física:** representaria a distribuição dos componentes em servidores diferentes. Por exemplo, o Módulo de Busca poderia ser distribuído em servidores dedicados com alta capacidade de processamento e armazenamento, enquanto o Sistema de Pagamentos poderia estar em uma infraestrutura isolada para garantir segurança e conformidade com as normas financeiras.
5. **Visão de Cenários:** poderia descrever o processo completo de um usuário desde a busca por um produto até a finalização da compra. Isso validaria que todas as visões se integram de forma coesa para permitir que o usuário tenha uma experiência fluida e segura.

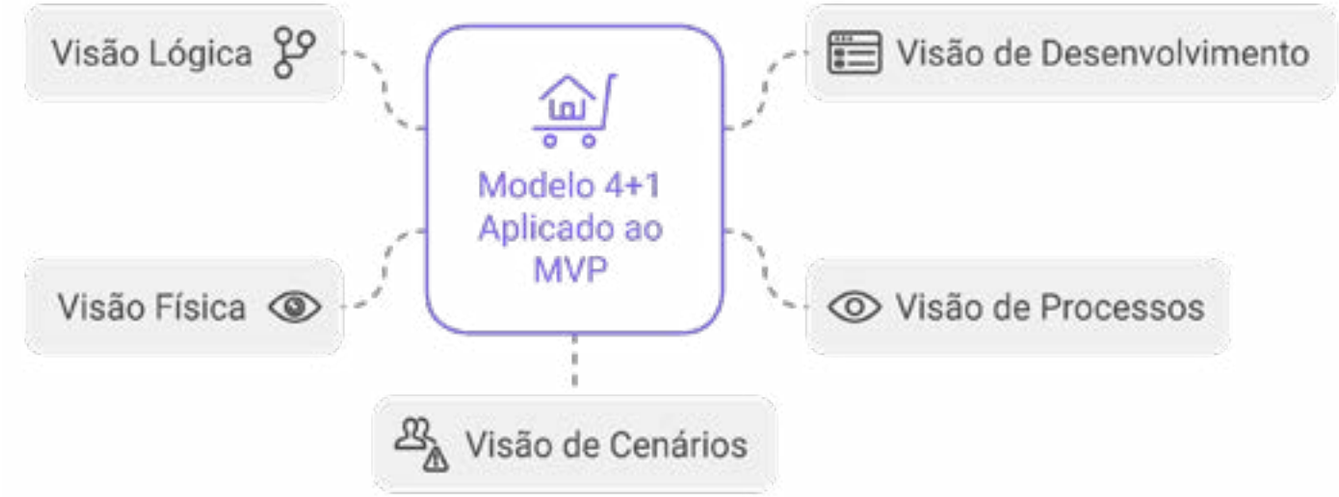
Ligando MVP e Modelo 4+1

MVP como o Foco Inicial

O MVP é a versão mais simples e funcional de um produto que pode ser lançada para começar a coletar feedback dos usuários. O objetivo é entregar rapidamente um produto que contenha apenas as funcionalidades essenciais.

Aplicando o Modelo 4+1 ao MVP

- **Visão Lógica:** mesmo que o MVP seja limitado em funcionalidades, ele precisa ter uma arquitetura lógica bem definida para garantir que as funcionalidades essenciais, como autenticação ou a principal função do sistema, sejam robustas e escaláveis. Essa visão garante que o MVP atenda aos requisitos funcionais iniciais de forma clara.
- **Visão de Desenvolvimento:** a arquitetura do MVP deve ser organizada de forma a permitir fácil expansão. Mesmo sendo um produto mínimo, a estrutura do código deve estar preparada para crescer conforme novas funcionalidades forem adicionadas.
- **Visão de Processos:** no caso de um MVP, essa visão se concentra em garantir que as funcionalidades básicas funcionem de maneira eficiente e com bom desempenho. Mesmo que o sistema seja simples, o comportamento em tempo de execução precisa ser analisado para evitar gargalos ou problemas de desempenho.
- **Visão Física:** para um MVP, a infraestrutura pode ser mínima, mas deve ser configurada de maneira que possa escalar à medida que o produto cresce. Essa visão garante que o sistema seja implementado em um ambiente que possa lidar com o tráfego inicial e ser facilmente ampliado.
- **Visão de Cenários:** os cenários para o MVP devem cobrir as operações mais críticas e frequentes que os usuários realizarão. Isso valida que a arquitetura mínima está correta e pronta para suportar as funções essenciais do sistema.



Referências

ABRAHANSSON, P.; SALO, O.; RONKAINEN, J.; WARSTA, J. Agile Software Development Methods. Review and Analysis. Espoo 2002, v. VTT Publicação 478, 2002.

AGILE ALIANCE WEB SITE. Agile Alliance [On-line]. Disponível em: <http://agilealliance.org>. Acessado em: abr. 2023.

AGILE MODELING WEB SITE. Agile Documentation [On-line]. Disponível em: <http://aligemodeling.com/essays/agileDocumentation.htm>. Acessado em: abr. 2023.

AMBLER, S. A., Test-Driven Development. Agile Data, 2003. Disponível em: <http://agiledata.org>. Acessado em: abr. 2023.

BECK, K. TDD by Example. 1. ed. Addison-Wesley Professional, 2002.

BOOCH, G., RUMBAUGH, J., JACOBSON, I., The Unified Modeling Language User Guide Reading. Addison Wesley, 1999.

CHRISTEL, M. G.; KANG, K. C. Issues in Requirements Elicitation. Software Engineering Institute, Technical Report CMU/SEI-92-TR-012 ESC-TR-92-012, 2012. Disponível em: <http://www.sei.cmu.edu/pub/documents/92.reports/pdf/tr12.92.pdf>. Acessado em: abr. 2023.

DAHLSTEDT, A., Requirements Engerneering - Chapter 11. Department od Computer Science, University of Skovde, 2003. Disponível em: <htt://www.ida.his.se/ida/kurser/informationsystems_engineering/kursmaterial/foreelasningar/Chapter11_2003.pdf>. Acessado em: abr. 2023.

EVANS, E., Domain Driven Design. Addison-Wesley, 2004.

FAULK, S. R.; Software Requirements: A tutorial. In: Thayer R. G.; Dorfman, M.; Software Requirement Engineering, 2nd ed. Wiley-IEEE Computer Society Press, 1997.

FRANCE, R., KOBRYN, C.; UML for Software Engineer. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 23. 2001.

HOOKS, I. Writing Good Requirements. In: INTERNATIONAL SYMPOSIUM OF THE INCOSE, 3., 1993. Disponível em: <http;//www.incose.org/rwg/writinggoodrqs_hooks.htm>. Acessado em: abr. 2023.

JEFFRIES, R. et al. TDD: The art of fearless programming, 2001.

SAWYER, P.; KOTONYA, G.; Chapter 2: Software Requirements. In: Guide to the Software Engineering Body of Knowledge, SWEBOK. A Project of the Software Engineering Coordinating, 2001. Disponível em: <htt://www.swebok.org/stoneman/trial_1_00.htm>. Acessado em: abr. 2023.

SCHARER, L., Pinpointing Requeriments. In: SYSTEM AND SOFTWARE REQUIREMENTS ENGINEERING. IEEE Computer Society Press, 1990.

SCHWABER, K.; SUTHERLAND, J., The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game. 2017.

SOMMERVILE, I.; SAWYER, P. Requirements Engineering: A good practice guide. 1. ed. New York: John Wiley & Sons, 1997.

VERHEYEN, G. Scrum: Framework, não metodologia. 2013.

WAZLAWICK, R. S. Engenharia de Software: Conceitos e Práticas. 2013.

WILDT, Daniel et al. eXtreme Programming: práticas para o dia a dia no desenvolvimento ágil de software. 2015.

YOUNG, R. R., Recommended Requirements Gathering Practices. CROSSTALK. In: The Jornal od Defense Software Engineering. Disponível em: <http://www.stsc.hill.af.mil/crosstalk/2002/04/young.html>. Acessado em: abr. 2023.


Faculdade
XPe
xpeducacao.com.br