UNIDADE 1 – CONTEXTO DA ARQUITETURA DE SOFTWARE
MÓDULO 1 – O QUE É ARQUITETURA DE SOFTWARE?

01

1 - ANTES DE INICIAR

Assim como na disciplina de Análise e Projeto de Sistema 2, para realizarmos as atividades desta disciplina, iremos precisar de um software de modelagem. Há várias opções gratuitas, como o software livre **Visual Paradi**gm. Outro software similar é o **Violet UML Editor**, também gratuito.

Observações:

1: caso você já tenha feito a disciplina de Análise e Projeto de Sistema 2, provavelmente você já utilizou um software para modelagem UML. Nesse caso, não será necessário instalar um novo software. Basta você utilizar o mesmo que já vinha utilizando.

2: não é escopo desta disciplina realizar um treinamento de Visual Paradigm, Violet UML, Eclipse ou qualquer outra ferramenta de UML. Para tanto, caso você tenha dificuldade no uso dessas ferramentas, pesquise no Youtube ou Google vídeos ou manuais de como utilizá-las.

Para que você possa aproveitar ao máximo essa disciplina, é necessário o conhecimento prévio dos seguintes assuntos:

- **Orientação a objeto**: conceitos como classe, método, evento, objeto, chamada, herança e outros são fundamentais para a diagramação UML.
- **UML**: Conhecer os diagramas existentes, sobretudo o **diagrama de classe** e **de componentes** será importante ao longo de nossa disciplina.

Esses conteúdos fazem parte do seu curso, então aproveite para compreendê-los bem!

02

2 - OS ENTENDIMENTOS FUNDAMENTAIS SOBRE A ARQUITETURA

Nos últimos anos estamos presenciando o crescimento de uma disciplina imprescindível para a engenharia de software conhecida como Arquitetura de Software. Arquiteto de Software e Líder de Arquitetura são papeis que estão cada vez mais presentes na indústria de software. Existe inclusive uma **associação internacional** sobre arquitetura de software. Tudo bem, mas o que é arquitetura de software?

É justamente a arquitetura de software que vamos explorar nesta disciplina.

Em particular, é sobre as principais questões no que se refere ao design e a tecnologia a serem consideradas na construção de sistemas que processam múltiplas solicitações simultâneas de usuários e outros sistemas de software. O objetivo é descrever os elementos essenciais de conhecimentos e habilidades fundamentais que são necessários para ser um arquiteto de software na indústria de tecnologia da informação (TI).

Ao falarmos de arquitetura, a primeira analogia que vem na mente é a **construção de edifícios**. Os livros de engenharia de *software* normalmente usam esta analogia para motivar as fases do ciclo de vida do *software* tradicional.

Em uma concepção simplificada da arquitetura, os requisitos para um edifício são coletados, um projeto é criado para atender a estes requisitos, o desenho é refinado para produzir modelos elaborados. A construção baseia-se nas plantas, e a estrutura resultante é então ocupada e usada.

Associação internacional
Você pode conhecer a associação internacional sobre arquitetura de software acessando o site http://www.iasahome.org/web/home/home).

03

Teoricamente, quando falamos de software, alguns passos podem ser previstos:

- os requisitos são especificados,
- um projeto de alto nível é criado,
- algoritmos detalhados são desenvolvidos com base no design,
- o código é escrito para implementar os algoritmos
- finalmente, o sistema está implantado e usável.

Para conceituarmos a arquitetura de software vamos recorrer ao IEEE. O motivo desta escolha é o propósito da criação do padrão **ISO/IEEE 1471-2000**. Este padrão tem justamente o objetivo de ajudar no consenso entre autores, estudantes e profissionais sobre o que é e para que serve arquitetura de software.

A definição de arquitetura de software do IEEE, a qual adotaremos ao longo de nossa disciplina, é a seguinte:

Arquitetura de Software é a organização fundamental de um sistema, representada por seus componentes, seus relacionamentos com o ambiente, e pelos princípios que conduzem seu design e evolução.

IEEE-1471
O IEEE Computer Society elaborou o padrão IEEE-Std-1471-2000, que é um conjunto de práticas recomendadas para descrever arquiteturas de sistemas de informação.

04

Arquitetura de *software* envolve a integração de metodologias e modelos de desenvolvimento de *software*, mas não podemos confundi-la com metodologias para a análise e projeto de sistema. A arquitetura de *software* é um corpo de métodos e técnicas que nos ajuda a gerenciar as complexidades do desenvolvimento de *software*.

Evolução de *software* é o fenômeno de mudança que ocorre no *software* ao longo dos anos e das múltiplas versões, desde seu início até o completo abandono do sistema.

Essa mudança não está só relacionada com a adição e remoção de funcionalidades, mas também está relacionada com a manutenção do código ao longo do ciclo de vida do *software*. Essa manutenção pode melhorar ou deteriorar tanto atributos externos de qualidade do *software*, os quais são percebidos pelos usuários (desempenho, tolerância a falhas, disponibilidade), quanto os atributos internos de qualidade do *software*, os quais são percebidos pelos envolvidos no desenvolvimento (testabilidade, legibilidade, reusabilidade).

Uma vez que um dos principais objetivos de se projetar uma arquitetura é o de **atingir a qualidade desejada pelos interessados no sistema**, se torna claro o papel da arquitetura em conduzir a evolução do *software*, uma vez que ela conterá decisões que contribuirão para a preservação da qualidade do sistema durante seu ciclo de vida.

Existem três entendimentos fundamentais sobre a arquitetura que ajudam a contextualizá-la em relação à engenharia de *software*:

1 – Toda a aplicação tem uma arquitetura;

2 - Toda aplicação tem pelo menos uma arquitetura;

3 - Arquitetura não é uma fase do desenvolvimento.

1 – Toda a aplicação tem uma arquitetura;

Voltando a nossa analogia de um prédio, é evidente que todo edifício tem uma arquitetura. Não significa dizer que esta arquitetura é funcional e que deixa os construtores orgulhosos, mas mesmo assim, todo edifícil, seja ele feio ou bonito tem uma arquitetura.

O mesmo pode ser observado com os *software*s. Através desta observação alguns questionamentos são levantados: De onde vem a arquitetura desta aplicação? Quais as propriedades desta arquitetura? Esta arquitetura é boa ou é ruim?

Mesmo com estes questionamentos é inegável que um *software* possui uma arquitetura.

2 - Toda aplicação tem pelo menos uma arquitetura;

O segundo conceito flui a partir do primeiro, em virtude de decisões realizadas ao longo do projeto ou de necessidades específica é possível que uma aplicação não tenha uma única arquitetura. Isso quer dizer que é possível que uma aplicação tenha uma mistura de diferentes arquiteturas.

3 - Arquitetura não é uma fase do desenvolvimento.

A Arquitetura de *software* refere-se à essência de uma aplicação, as principais decisões de design, às principais abstrações que caracterizam uma aplicação.

Em uma compreensão simplista, tradicional, e imprecisa, a arquitetura é um produto específico de uma determinada fase do processo de desenvolvimento. Mas para que esta premissa seja atendida seria necessário conhecer todos os requisitos do projeto. Assim, se pensarmos na arquitetura como

um produto de uma fase limitando o seu momento de atuação, pricipalmente no início de projeto, estamos restringindo a arquitetura a apenas algumas decisões de design. Em muitos casos, estas decisões necessitam ser violadas por necessidades posteriores identificadas ao longo do projeto. Assim, temos que encarar a arquitetura de *software* como uma atividade que permeia todo o processo de desenvolvimento.

05

3 - ARQUITETURA E SEU PAPEL NO DESENVOLVIMENTO DE SOFTWARE

Após entendermos o conceito da arquitetura de *software* e de aprofundarmos nosso entendimento nos conceitos fundamentais, vamos entender qual o papel da arquitetura de *software* em um contexto mais amplo do desenvolvimento de sistemas e como, em uma abordagem mais moderna, as demais disciplinas podem se beneficiar da arquitetura.

3.1 Requisitos

A arquitetura deve começar no início de qualquer atividade de desenvolvimento de *software*. Noções de estrutura, design e solução são bastante apropriadas durante a atividade de análise de requisitos.

Entretanto, a visão acadêmica tradicional de análise de requisitos e especificação é de que a atividade, e o documento de requisitos resultante, deve permanecer imaculada por qualquer consideração de um projeto que pode ser usado para satisfazer os requisitos identificados.

De fato, alguns pesquisadores na comunidade requisitos são bastante radicais sobre este ponto.

Agora, imagine se os invetores das máquinas de lavar roupa levassem isso ao pé da letra. Se simplesmente automatizasse a solução manual dos séculos passados, teríamos máquinas que batiam roupas nas pedras na beira do rio. Assim, é fácil perceber que ter noções de estrutura e design é um ponto crítico durante a fase de análise de requisitos.

06

Considere mais uma vez a analogia com os edifícios. Quando decidimos que precisamos de uma nova casa ou apartamento, ou uma reforma em nossa moradia atual, iniciamos um processo de raciocínio sobre as nossas necessidades, independentemente de como eles podem ser satisfeitos. Nós pensamos na quantidade de quartos, no estilo de janelas, como será a iluminação e assim por diante. Temos experiência com habitação, e esta experiência nos permite raciocinar rapidamente e articuladamente sobre nossos desejos.

Entretanto, sem o conhecimento necessário, é possível fazermos análises de custo e cronograma para atender às necessidades? A resposta é não.

Assim é com o *software*. Especificação de requisitos de forma independente de qualquer preocupação com o modo como esses requisitos podem ser atendidos leva à dificuldade na avaliação da praticidade, prazo para o desenvolvimento ou custo do projeto.

Vendo o que pode ser feito em outros sistemas, por outro lado, que tipo de interfaces com o usuário está disponível, o que o novo hardware é capaz de fazer, que tipo de serviços podem ser prestados, pode facilitar a implementação e pode servir para nortear a análise e requisitos.

07

3.2 Design

Design é, por definição, a atividade que cria a arquitetura do *software*.

Esta arquitetura é o resultado de um conjunto de decisões de design. Estas decisões abrangem questões que surgem ao longo do processo de desenvolvimento.

Desta forma, são 3 as características que temos que levar em consideração:

1. Se considerarmos a fase de projeto tradicional de Design e Arquitetura não devemos encará-la como um momento exclusivo, ou seja, o "lugar" ou o "tempo" quando a arquitetura do sistema é desenvolvida.
2. Como as principais decisões de arquitetura são tomadas ao longo do processo de desenvolvimento, o design deve ser encarado como um aspecto também de outras atividades do desenvolvimento.
3. Um rico repertório de técnicas de design é necessário para auxiliar o arquiteto a tomar as decisões de design que devem compor a arquitetura. Riscos para o *deploy*, segurança, escolhas de componentes *open-souce* ou proprietário são aspectos que também devem ser considerados para a definição da arquitetura.

08

3.3 Implementação

A tarefa de implementação é criar código-fonte executável que seja fiel à arquitetura e que desenvolva plenamente todos os detalhes da aplicação.

No entanto, temos que considerar alguns aspectos adicionais ao longo da implementação.

Primeiro, a atividade de implementação **pode** modificar a arquitetura. Se as principais decisões de *design* são feitas ao trabalhar com o código fonte, eles são parte da arquitetura tanto quanto qualquer decisão principal feita muito mais cedo no processo.

Em segundo lugar, não podemos presumir que a arquitetura é concluída antes do início da implementação. Na verdade, pode haver revisão da arquitetura enquanto o código é desenvolvido. O que devemos considerar é manter todas as decisões registradas. Ou seja, a atividade de implementação não pode virar as costas para as decisões anteriores. As modificações realizadas durante a fase de implementação devem ser documentadas como parte da arquitetura do aplicativo.

Não estamos falando, entretanto, que implementação não deve respeitar as decisões arquiteturais anteriores. A palavra de ordem para a criação da aplicação é que **ela deve ser fiel à arquitetura**.

Assim, para melhor entendermos, o que devemos levar em consideração é que a implementação deve seguir os elementos estruturais encontradas na arquitectura e que todo o código de fonte corresponde a várias partes da arquitetura.

Desta forma, o código fonte **não deve**:

- utilizar novos elementos que não tenham um correspondente na arquitetura definida.
- conter novas conexões entre elementos da arquitetura que não são encontradas na arquitetura.

09

3.4 Análise e Teste

Análise e os testes são atividades realizadas para avaliar a qualidade de um artefato.

Mas você pode se perguntar: então estas atividades devem ser realizadas apenas ao final do projeto?

O que deve ser levado em consideração é que uma das virtudes de realizar a análise antes da existência código é a economia gerada:

Quanto mais cedo for detectado um erro, menor o custo agregado para a correção.

Vamos avaliar inicialmente os benefícios da análise. A arquitetura de um aplicativo pode ser examinada considernado:

- sua consistência,
- exatidão,
- e sua capacidade de atendimento aos requisitos não funcionais.

Se a arquitetura sobre a qual a implementação é baseada é de alta qualidade, a probabilidade de a implementação ser de alta qualidade é significativa.

Como um artefato formal, o modelo de arquitetura (documento que descreve a arquitetura do *software*) pode ser examinado em relação a sua **coerência interna** e **correção**: a verificação sintática do modelo pode identificar, por exemplo, componentes incompatíveis, especificação incompleta das propriedades, e padrões de comunicação indesejáveis. De forma mais significativa, a análise de fluxo pode ser usada para detectar falhas de segurança.

10

Em segundo lugar, o modelo de arquitetura pode ser examinado por **coerência com os requisitos**. Ou seja, os requisitos e a arquitetura devem ser consistentes entre si.

Em terceiro lugar, o modelo de arquitetura pode ser utilizado na determinação de estratégia de teste a ser aplicada ao código-fonte. Como a **arquitetura fornece o projeto para o código-fonte**, a coerência entre elas é essencial. Desta forma, a arquitetura serve como fonte de informações para a definição de testes baseados nas especificações.

Na mesma linha, a arquitetura pode proporcionar um **meio de antecipar os testes** gerando redução de custos e contribuindo para diminuir o cronograma do projeto. Por exemplo, se um componente específico é reutilizado em uma nova arquitetura, os testes unitários podem ser reduzidos se a análise da arquitetura confirma que o contexto e as condições de utilização deste componente são as mesmas (ou mais limitado) do que o uso anterior.

Como observado, um **desenvolvimento focado na arquitetura** fornece uma variedade de novas e significativas oportunidades para avaliar e, consequentemente, possibilitar a melhoria da qualidade dos sistemas.

11

4 - O ARQUITETO DE *SOFTWARE*

4.1- Quem é o arquiteto de *Software*

Vimos o que é a arquitetura de *software*, mas quem é responsável por definir esta arquitetura?

A resposta para esta pergunta é simples: O **arquiteto de *software***. Mas o perfil deste profissional, bem como sua responsabilidade não é tão simples assim.

Um arquiteto de *software* é alguém capaz de unir **método**, **intuição** e **reutilização** a fim de definir a arquitetura viável para um determinado projeto de *software*.

Método refere-se a um conjunto de técnicas que podem ser executadas várias vezes para resolver um problema particular, ou categoria de problema.

A **intuição** é a capacidade de conceber, entender e aplicar ideias sem necessariamente ser capaz de se comunicar ou explicar toda a lógica por trás delas.

Reutilização refere-se à reutilização de soluções que foram mostrados bem sucedida no passado.

12

A essência do trabalho de um arquiteto de *software* é ser capaz de aplicar e encontrar o equilíbrio certo entre estes três "fontes" da arquitetura.

Assim podemos afirmar que o papel de um arquiteto de *software* é de grande responsabilidade em um projeto de *software*. Até mesmo o sucesso do projeto vai ser influenciado diretamente pela solução arquitetônica empregada.

Em virtude da importância e responsabilidade exercida por este profissional é natural persarmos que esse papel deve sempre ser exercido por alguém que tenha **autoridade** e **liderança**. Entretanto, em muitas organizações de desenvolvimento de *software* a responsabilidade não é acompanhada com autoridade específica. Desta forma, a fim de fazer o seu trabalho de forma eficaz, o arquiteto deve, então, encontrar uma outra maneira de exercer influência e liderança.

Ele deve possuir uma série de **habilidades**, tais como:

- **Designer de *software*;**
- **Especialista de domínio;**
- **Técnico;**
- **Especialista em padrões;**
- **Economista.**

Designer de *software*

Um arquiteto de *software* deve ser um excelente designer de sistemas de *software*. Ele deve ser capaz de reconhecer, reutilizar, ou criar soluções de design eficazes e aplicá-las de forma adequada. Ele deve estar familiarizado com os estilos arquitetônicos fundamentais e padrões que estão na base da disciplina de arquitetura de *software*. Um arquiteto pode até ter alguns padrões que fazem parte de seu arsenal privado de ferramentas de design, acumulado ao longo do tempo. Por outro lado, um arquiteto também deve ser capaz de reconhecer projetos ineficazes, estilos impróprios e padrões, e decisões de design sem a flexibilidade adequada.

Especialista de domínio

Muitas organizações de desenvolvimento de *software* produzir vários sistemas dentro do mesmo domínio de aplicação. Por exemplo, a Microsoft trabalha principalmente dentro do domínio de aplicativos para desktop. O domínio principal do Google é a disseminação de informação através da Internet; já a Boeing e tem um extenso foco em sistemas embarcados; NASA sistemas espaciais.

Assim, quanto mais especialista em um domínio, melhor será o arquiteto.

Técnico

Um arquiteto de *software* deve saber que suas soluções realmente vai funcionar quando for desenvolvida. Assim, um arquiteto de *software* precisa garantir que a tecnologia de *software* existente é capaz de suportar a implementação da arquitetura definida. Um design bonito não é inútil se não pode ser implementado.

Especialista em padrões

As exigências para a contratação de uma empresa de desenvolvimento estão cada vez mais rigorosas. Por este motivo, é crítico que um arquiteto tenha domínio dos padrões de mercado. Por exemplo, para poder ser contratada pelo governo americano uma empresa de desenvolvimento deve ser certificada CMMI 5. Um arquiteto deve ter conhecimento deste padrão.

Economista

Um grande designer de *software* não é necessariamente um grande arquiteto de *software*. Um bom arquiteto não pode se ater APENAS às melhores técnicas, os seus desenhos devem ser sobretudo realistas, economicamente viável, tecnologicamente implementável, e com o menor risco para o projeto. Em outras palavras, um arquiteto deve ser mais do que apenas uma (grande) designer. Um arquiteto deve elaborar uma arquitetura que efetivamente resolve o problema em questão, respeitando simultaneamente as restrições impostas ao projeto.

4.2- O que um arquiteto de *software* faz?

O trabalho de um arquiteto de *software* exige uma série de habilidades. Assim como um designer de *software* exige um talento específico, um arquiteto vai gastar muito de seu tempo fazendo outras coisas.

Há quatro **tarefas** que um arquiteto necessita executar independe do tipo de projeto, do seu nível de experiência, o tipo de organização, do domínio do aplicativo, ou do segmento de indústria em que trabalha. São elas:

- **Desenvolver estratégia do projeto**

Um arquiteto talentoso será capaz de produzir excelentes soluções técnicas para os problemas que lhe forem apresentado. Mas, apesar de crítico para o trabalho de um arquiteto, isso não é o bastante. Um arquiteto deve ajudar a desenvolver uma estratégia viável para o projeto. Uma estratégia que, se implementada corretamente, resultará em um produto tecnicamente bom, mas também que agregue valor para a organização.

- **Design de sistemas**

Uma parte central da definição de uma estratégia para o projeto é definir a arquitetora para o projeto. De posse das principais restrições impostas pelo projeto, o arquiteto deve definir como os desafios serão superados através de uma arquitetura robusta e flexível.

- **Comunicação com os interessados**

Durante o projeto de definir a estratégia para o projeto e fazer o design da arquitetura, o arquiteto deverá constantemente com os diferentes interessados pelo projeto. Entre os interessados incluem-se os desenvolvedores, testadores, gerentes de diferentes níveis, clientes, líderes técnicos e os usuários.

- **Liderar**

Um arquiteto deve tomar várias decisões que são fundamentais para o sucesso do projeto. Algumas destas decisões podem ser impopulares, mas ele deve garantir que os principais interessados compraram esta decisão. Para que isso seja possível é imprescindível que o arquiteto seja um líder. Durante todo o projeto, o arquiteto deve colocar os interessem do projeto sobre os seus próprios interesses e liderar através de exemplo.

No cumprimento destas tarefas, o arquiteto terá que aplicar, combinar e até aprimorar o conjunto de habilidades descritas na seção anterior.

14

5 - CONCEITOS BÁSICOS

Quando tratamos da arquitetura, uma série de termos e conceitos é fundamental para o completo entendimento desta importante disciplina da engenharia de *software*. Desta forma, vamos explorar estes conceitos para aprimorar o nosso entendimento.

- **Arquitetura**

Técnico

Um arquiteto de *software* deve saber que suas soluções realmente vai funcionar quando for desenvolvida. Assim, um arquiteto de *software* precisa garantir que a tecnologia de *software* existente é capaz de suportar a implementação da arquitetura definida. Um design bonito não é inútil se não pode ser implementado.

- **Arquitetura de Referência**

A arquitetura de referência é um conjunto de princípios de *design* aplicáveis simultaneamente em diferentes sistemas relacionados tipicamente, que compartilham o mesmo domínio de aplicação.

- **Arquitetura Prescritiva**

A arquitetura prescritiva é a arquitetura como concebida para o sistema, ou seja, a arquitetura idealizada pelos arquitetos que deve direcionar o desenvolvimento do *software*.

A arquitetura prescritiva não precisa necessariamente existir de uma forma tangível, podendo inclusive estar apenas nas mentes dos arquitetos. Entretanto é sempre recomendávelque ela seja capturada em uma notação ou outra forma de documentação. **Saiba+**

Saiba+

É importante notar que documentar a arquitetura prescritiva não é suficiente. O leitor deve lembrar que a arquitetura não é apenas uma fase no processo de desenvolvimento de um sistema de *software*, mas sim constitui a base fundamental para o sistema. Assim, a qualquer momento durante o processo de desenvolvimento, as decisões de projeto de arquitetura, que são parte da arquitetura prescritiva, serão refinadas e documentadas.

15

- **Arquitetura Descritiva**

A arquitetura descritiva é referida como tal quando descreve a forma como o sistema foi implementado. A arquitetura descritiva é, portanto, a arquitetura conforme realizada no sistema.

Durante o tempo de vida de um sistema de *software* poderão existir diferentes arquiteturas prescritivas e descritivas. Cada par correspondente de tais arquiteturas representa a arquitetura do *software* em um dado momento.

- **Degradação Arquitetônica**

É a diferença entre as arquiteturas descritiva e prescritiva.

Em um cenário ideal, as duas arquiteturas, prescritiva e descritiva, sempre serão iguais. Entretanto, na maioria das vezes isso não é verdade. A diferença entre elas pode ocorrer por diversas razões:

- Desenvolvedor que não se preocupa em documentar o que está fazendo;
- Percepção de prazos curtos que impedem pensar na atualização da documentação;
- Falta de uma arquitetura prescritiva documentada;
- Necessidade ou desejo de otimizar o sistema "que só pode ser feito no código"; e
- Técnicas inadequadas que não se deseja documentar.

16

- **Perspectivas arquitetônicas**

Podemos definir as pespectivas como um conjunto de decisões de projeto de arquitetura.

A noção de uma perspectiva arquitetônica é destacar alguns aspectos de uma arquitetura enquanto outros aspectos são identificados.

As arquiteturas de *software* abrangem as decisões tomadas por uma grande variedade de interessados em diferentes níveis de detalhe e abstração. O objetivo é focar a atenção, por exemplo, para fins de análise, a um subconjunto das decisões.

Outra perspectiva é a **implementação**. Um sistema de *software* não pode cumprir seu propósito até que seja implantada, ou seja, até que seus módulos executáveis sejam fisicamente colocados sobre os dispositivos de *hardware* em que se destinam a ser executados.

A perspectiva de implantação de uma arquitetura pode ser fundamental para avaliar se o sistema será capaz de satisfazer as suas necessidades. Por exemplo, colocar muitos componentes grandes em um

pequeno dispositivo com memória e potência da CPU limitadas, ou a transferência de grandes volumes de dados através de uma conexão de rede com baixa largura de banda vai impactar negativamente o sistema.

17

- **Componentes**

Componentes de *software* são elementos que encapsulam o processamento e os dados na arquitetura de um sistema.

As decisões que compõem a arquitetura de um sistema de *software* abrangem uma interação rica e a composição de muitos elementos diferentes. Estes elementos abrangem as preocupações principais do sistema, incluindo:

- Processamento;
- Estado;
- Interação.

Um componente de *software* é uma entidade arquitetônica que:

- Incorpora um subconjunto da funcionalidade e de dados do sistema;
- Restringe o acesso a esse subconjunto através de uma interface explicitamente definida;
- Tem dependências explicitamente definidas no seu contexto de execução.

Processamento

Que também pode ser referido como a funcionalidade ou o comportamento.

Estado

Que também pode ser referido como a informação ou dados.

Interação

Que também pode ser referida como a interligação, a comunicação, coordenação, ou mediação.

18

- **Conectores**

Um conector é um elemento arquitetônico encarregado de efetuar e regular as interações entre os componentes de *software*.

Outro aspecto fundamental de sistemas de *software* é a interação entre os blocos de construção do sistema. Muitos sistemas modernos são construídos a partir de um grande número de componentes complexos, distribuídos em vários hosts, que são atualizados dinamicamente durante um determinado período de tempo. Em tais sistemas, garantir interações adequadas entre os componentes pode tornar-se ainda mais importante e desafiador para os desenvolvedores do que a funcionalidade dos componentes em si; em outras palavras, **as interações do sistema se tornam a principal preocupação arquitetônica**.

Conectores de *software* são a abstração arquitetônica encarregada de gerenciar as interações de componentes.

- **Configuração**

Uma configuração arquitetônica é um conjunto de associações específicas entre os componentes e conectores de arquitetura de um sistema de *software*.

19

- **Estilo Arquitetural**

Um estilo arquitetônico é uma coleção nomeada de decisões de projeto de arquitetura que:

- São aplicáveis em um determinado contexto de desenvolvimento,
- Limitam as decisões de projeto de arquitetura que são específicos para um determinado sistema dentro desse contexto,
- Provocam qualidades benéficas em cada sistema resultante.

Com o desenvolvimento de muitos sistemas para uma enorme gama de domínios de aplicação, o que se tem observado é que, sob determinadas circunstâncias, certas escolhas de *design* regularmente resultam em soluções de sucesso. Em comparação a alternativas possíveis, essa solução é mais elegante, eficaz, eficiente, confiável e escalável.

Esta lista de decisões de arquitetura de design, portanto, não é específica para um determinado sistema (ou classe de sistemas). Pelo contrário, estas decisões de arquitetura são aplicáveis a qualquer sistema que compartilha o contexto de prestação de serviços distribuídos.

- **Padrão Arquitetural**

Estilos arquitetônicos fornecem decisões gerais de *design* que podem restringir ou podem necessitar de refinamentos para que possa ser aplicados a um sistema.

Já um padrão de arquitetura fornece um conjunto de decisões de design específico que foram identificadas como eficazes para a organização de certas classes de sistemas de *software*.

Assim, um padrão de arquitetura é uma coleção de decisões de projeto de arquitetura que são aplicáveis a um problema recorrente de *design*, parametrizados para explicar diferentes contextos de desenvolvimento de *software* em que o problema aparece.

20

- **Modelos**

Um modelo de arquitetura é um artefato que capta algumas ou todas as decisões de *design* que compõem a arquitetura de um sistema. Modelo arquitetônico é a documentação dessas decisões de design.

Um modelo é um resultado da atividade de modelagem, o que constitui uma parte significativa das responsabilidades de um arquiteto *software*. Um sistema pode ter diversos modelos distintos associados. Os modelos podem variar na quantidade de detalhes que eles capturar, na perspectiva específica arquitetônica que eles capturam, no tipo de notação que eles usam, e assim por diante.

O modelo arquitetural é usado como a base para a maioria das outras atividades no processo de desenvolvimento de *software* baseado em arquitetura, como a análise e a implementação do sistema.

21

RESUMO

Neste módulo vimos que a Arquitetura de *Software* é a organização fundamental de um sistema, representada por seus componentes, seus relacionamentos com o ambiente, e pelos princípios que conduzem seu design e evolução.

Também vimos como uma visão adequada da arquitetura de *software* afeta todos os aspectos das atividades de engenharia de *software*. Esta influência se dá da seguinte forma:

- A atividade requisitos é vista como uma parceira da atividade de arquitetura, em que produtos e projetos anteriores fornecem o vocabulário para articular novas exigências, e em que novas idéias de design fornecer a inspiração para estratégias e exigências para o detalhamento dos requisitos do produto.
- A atividade de projeto é enriquecida por técnicas que exploram os conhecimentos adquiridos no desenvolvimento de produtos anteriores.
- A atividade de implementação é centrado na criação de uma implementação fiel da arquitetura e utiliza uma variedade de técnicas para alcançar isso de uma forma eficaz em termos de custos,
- As atividades de análise e de teste pode ser focada e orientada pela arquitetura, que oferece a perspectiva de detecção mais precoce de erros e exames mais eficientes e eficazes do produto. Qualidade superior a um preço mais baixo deve ser o resultado.

Foram apresentadas as principais responsabilidade de um arquiteto de *software* e o perfil deste importante profissional.

Foram explorados ainda alguns conceitos básicos da arquitetura de *software* como:

Arquitetura de Referência, Arquitetura Prescritiva, Arquitetura Descritiva, Degradação Arquitetônica, Perspectivas arquitetônicos, Componentes, Conectores, Configuração, Estilo Arquitetural, Padrão Arquitetural e Modelos.

UNIDADE 1 – CONTEXTO DA ARQUITETURA DE SOFTWARE
MÓDULO 2 – O QUE É ARQUITETURA DE SOFTWARE?
01

1 - CARACTERÍSTICAS DA ARQUITETURA DE SOFTWARE

Em nosso módulo anterior definimos a arquitetura de *software* como a organização fundamental de um sistema, representada por seus componentes, seus relacionamentos com o ambiente, e pelos princípios que conduzem seu *design* e evolução. Analisando esta definição com um pouco mais de atenção, podemos extrair algumas das características fundamentais da arquitetura de *software*. Estas características que abordaremos a seguir.

1.1- Arquitetura define a estrutura

Muito do tempo de um arquiteto está ocupado em estruturar de forma sensata uma aplicação através de um conjunto de componentes inter-relacionados, módulos, objetos ou qualquer unidade de particionamento *software*.

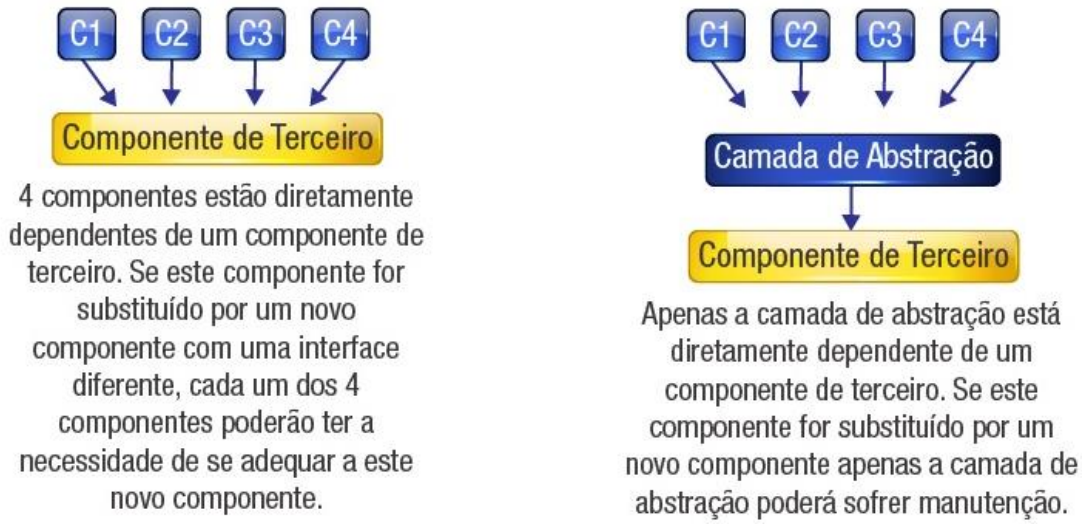
Uma arquitetura deve ser projetada para atender às necessidades específicas e às restrições da aplicação a que se destina.

Por exemplo, um requisito para um sistema de gerenciamento de informações: pode ser que este aplicativo seja distribuído em diferentes localizações e que uma determinada funcionalidade, bem como seus dados, deve estar concentrada em um local único. Outro requisito é que as funcionalidades do aplicativo devem estar acessíveis a partir de um navegador web. Todas estas necessidades podem impor algum tipo de restrição estrutural para a aplicação e, simultaneamente, possiblitar uma arquitetura onde existe uma coleção de componentes relacionados.

Ao particionar um aplicativo, o arquiteto atribui responsabilidades para cada componente. Estas responsabilidades definem as tarefas que cada componente tem dentro do aplicativo. Deste modo, cada componente desempenha um papel específico na aplicação, e o conjunto global do componente que compreende a arquitetura para fornecer as funcionalidades requeridas para o sistema.

02

A questão estrutural fundamental para quase todas as aplicações é minimizar as dependências entre os componentes, criando uma arquitetura de baixo acoplamento. Existe uma dependência entre os componentes quando uma mudança em um potencialmente força uma alteração nos outros. Ao eliminar dependências desnecessárias, alterações são localizadas e não se propagam ao longo de uma arquitetura. Veja a figura a seguir.



Dois exemplos de dependências de componentes

Para simplificar, as dependências excessivas são simplesmente uma coisa ruim. Dentre as **características negativas de uma dependência excessiva** encontram-se:

- **Mais difícil fazer mudanças nos sistemas;**
- **Mais caro os testes das alterações;**
- **Aumentam o tempo de construção; e**
- **Tornam o desenvolvimento em equipe concorrente mais difícil.**

03

1.2 - Comunicação Específica entre os Componentes da Arquitetura

Quando um aplicativo é dividido em um conjunto de componentes, torna-se necessário pensar sobre como esses componentes irão se comunicar. Os componentes em uma aplicação podem existir no mesmo endereço, e se comunicar através de chamadas de método simples. Eles podem ser executados em diferentes segmentos ou processos, e se comunicar através de mecanismos de sincronização. Ou múltiplos componentes podem precisar ser simultaneamente informados quando ocorre uma ação no ambiente da aplicação. Existem diferentes possibilidades.

Uma série de estruturas utilizadas com sucesso na comunicação de certos tipos de comunicação entre componente foi catalogada através de padrões. Esses padrões são essencialmente plantas de arquitetura reutilizáveis que descrevem a estrutura e interação entre coleções de componentes participantes.

Cada padrão tem características que o tornam adequado para usar em tipos particulares satisfazendo-se certos requisitos de arquitetura.

Por exemplo, o padrão conhecido como cliente-servidor tem várias características úteis, tais como servidores que suportam um ou mais clientes através de uma interface publicada e uma comunicação síncrona de solicitação e resposta entre os clientes e o servidor. Opcionalmente, os clientes podem estabelecer sessões com servidores, o que pode manter o estado sobre os seus clientes conectados. Arquiteturas cliente-servidor também devem fornecer um mecanismo para que os clientes localizem servidores, tratem erros e, opcionalmente, oferecer segurança no acesso ao servidor.

04

O poder dos padrões de arquitetura decorre de sua utilidade e capacidade de transmitir informações para o projeto. Se usados adequadamente em uma arquitetura, você alavancar o conhecimento de *design* existente usando padrões.

Grandes sistemas tendem a usar vários padrões combinados de maneira que satisfaçam os requisitos de arquitetura.

Quando uma arquitetura é **baseada em padrões**, também se torna mais fácil para os membros da equipe entender o *design*, como o padrão infere estrutura de componentes, comunicações e mecanismos abstratos que devem ser fornecidos.

Quando um arquiteto diz que seu sistema é **baseado em uma arquitetura cliente-servidor de três camadas**, imediatamente uma quantidade considerável de informações sobre o *design* deste sistema pode ser identificada. De fato, este é um poderoso mecanismo de comunicação.

Vamos explorar os principais padrões de arquitetura em um módulo posterior de nosso curso.

05

1.3- Arquitetura de requisitos não funcionais

Os requisitos não funcionais são aqueles que não aparecem nos casos de uso. Ao invés de definir o que o aplicativo faz, eles estão preocupados com a forma como a aplicação fornece a funcionalidade necessária.

Há três áreas distintas de requisitos não funcionais:

- **Restrições técnicas**

Estas serão familiares a todos. As restrições técnicas limitam as opções de projeto, especificando certas tecnologias que o aplicativo deve usar. "O projeto só dispõe de desenvolvedores Java, por isso devemos desenvolver em Java". "O cliente só tem suporte para o Banco de Dados DB2".

- **Restrições de negócios**

Estas restrições estão ligadas mais s empresa como um todo e não a razões técnicas. Por exemplo, "A fim de aumentar a nossa base de clientes em potencial, temos de interagir com aplicativos disponíveis em IOS". Outro exemplo é "O fornecedor de nosso middleware elevou os preços a nívies proibitivos, por isso estamos migrando para uma versão open source".

- **Atributos de qualidade**

Definem requisitos de um aplicativo em termos de escalabilidade, disponibilidade, facilidade de mudança, portabilidade, usabilidade, desempenho e assim por diante. Atributos de qualidade abordar questões de interesse para usuários do aplicativo, bem como outras partes interessadas, como a própria equipe do projeto ou o patrocinador do projeto.

Uma arquitetura de aplicação deve, portanto, abordar explicitamente estes aspectos do *design*. Arquitetos precisam entender os requisitos funcionais, e criar uma plataforma que os contemple e cumpra simultaneamente os requisitos não-funcionais.

06

1.4- Arquitetura é uma abstração

Uma das descrições mais úteis, mas muitas vezes inexistentes, a partir de uma perspectiva arquitetônica é algo que é coloquialmente conhecido como ***marketecture***.

A *marketecture* é uma documentação, geralmente de uma página, que representa de forma informal a estrutura e as interações do sistema. Ela mostra os principais componentes e suas relações e tem algumas caixas de texto que retratam as filosofias de *design* incorporadas na arquitetura.

Trata-se de um excelente veículo para facilitar a discussão pelas partes interessadas durante a concepção, construção, revisão, e, até mesmo, no processo de vendas. É fácil de entender e explicar e serve como um ponto de partida para uma análise mais profunda.

A *marketecture* cuidadosamente trabalhada é particularmente útil porque é uma descrição abstrata do sistema. Na realidade, qualquer descrição arquitetônica deve empregar abstração, a fim de ser compreensível pelos membros da equipe e partes interessadas no projeto. Isto significa que detalhes desnecessários são reprimidos ou ignorados, a fim de concentrar a atenção e análise sobre as questões arquitetônicas marcantes. Isso geralmente é feito através da descrição dos componentes na arquitetura como caixas pretas, especificando apenas as suas propriedades externamente visíveis.

07

Um dos mecanismos mais poderosos para descrever uma arquitetura é a **decomposição hierárquica**.

Os componentes que aparecem em um nível de descrição são decompostos em mais detalhe na documentação que acompanha o *design*.

Como um exemplo, a imagem abaixo descreve uma hierarquia de dois níveis muito simples usando uma notação informal, com dois dos componentes no diagrama de nível superior decomposta.

Ilustração: refazer a imagem, mantendo as informações e aumentando a fonte para facilitar a leitura.

Descrevendo uma arquitetura hierárquica

Diferentes níveis de descrição na hierarquia tendem a ser de interesse para diferentes desenvolvedores em um projeto. Na imagem, é provável que os três componentes na descrição de nível superior sejam projetados e construídos por diferentes equipes de trabalho. A arquitetura exibe as responsabilidades de cada equipe, definindo as dependências entre eles.

08

Neste exemplo hipotético, o arquiteto refinou a concepção de dois dos componentes, presumivelmente porque alguns requisitos não funcionais necessitam de definições adicionais. Uma hipótese é que um serviço de segurança existente deve ser utilizado, ou o broker deve fornecer uma função de roteamento de mensagens específicas exigindo um serviço de diretório que tem um nível conhecido de pedido de transferência. Independentemente disso, este aperfeiçoamento cria uma estrutura que define e limita a concepção mais detalhada desses componentes.

Observe que na arquitetura simples da figura anterior não houve a decomposição do componente *cliente*. Isto se deve ao fato de que o comportamento do cliente não é significativo em relação aos requisitos não-funcionais gerais do aplicativo. A forma como o cliente obtém a informação que é enviada para o broker não é uma questão que diz respeito ao arquiteto, e, consequentemente, o projeto detalhado é deixado em aberto para a equipe de desenvolvimento do componente.

O componente cliente poderia ser o mais complexo na aplicação. Ele pode ter uma arquitetura interna definida por sua equipe de projeto, que atende às metas de qualidade específicas. No entanto, estas são preocupações localizadas em que o arquiteto pode deixar para a equipe de *design* do cliente resolver.

Este é um exemplo de como suprimir detalhes desnecessários na arquitetura.

figura

Produção: inserir como link a imagem “Descrevendo uma arquitetura hierárquica” da tela anterior.

09

2 - Visões de Arquitetura de Software

A arquitetura de *software* é algo complexo de se representar e, consequentemente, de se documentar. Por este motivo, é possível identificarmos diferentes maneiras de olhar e compreender uma arquitetura. Voltando a nossa analogia com a construção civil, as visões da arquitetura de *software* seriam as diferentes plantas existentes para a construção de uma casa ou um prédio.

Em 1995 Philippe Krutchen em seu artigo *4 + 1 Model View* traz à luz uma maneira de descrever e compreender uma arquitetura através de quatro pontos de vista:

- **Visão lógica**
- **Visão de processo**
- **Visão física**
- **Visão de desenvolvimento**

Estes pontos de vista são amarrados pelos casos de uso arquiteturalmente significativos (muitas vezes chamado de cenários). Estes basicamente capturam os requisitos para a arquitetura e, portanto, estão relacionadas a mais de um ponto de vista em particular.

Ao trabalhar com as etapas em um caso de uso, a arquitetura pode ser "testada", explicando como os elementos de *design* na arquitetura respondem ao comportamento exigido no caso de uso.

Visão lógica

Esta visão descreve os elementos significativos da arquitetura e as relações entre eles. A visão lógica essencialmente captura a estrutura do aplicativo usando diagramas de classe ou equivalentes.

Visão de processo

Esta visão se concentra em descrever os elementos de concorrência e de comunicação de uma arquitetura. Em aplicações de TI, as principais preocupações estão descrevendo componentes de segmentos multithreaded e os mecanismos de comunicação síncrona ou assíncrona usado.

Visão física

Esta visão descreve como os principais processos e componentes são mapeados para o hardware da aplicação. Ele pode mostrar, por exemplo, como os servidores de banco de dados e web são distribuídos através de um número de servidores.

Visão de desenvolvimento

Esta visão capta a organização interna dos componentes de *software*, tipicamente como eles são mantidos em um ambiente de desenvolvimento ou ferramenta de gerenciamento de configuração. Por exemplo, a descrição de uma hierarquia de pacote e classe de uma aplicação Java representa o ponto de vista do desenvolvimento de uma arquitetura.

10

Desde a descrição realizada por Krutchen, tem havido muita reflexão, experiência e desenvolvimento na área de visões de arquitetura. Um trabalho que merece destaque é o realizado pelo **SEI**.

Conhecido como abordagem "**Views and Beyond**", este trabalho recomenda a captura de um modelo de arquitetura usando três diferentes pontos de vista:

- **Módulo**

Esta é uma visão estrutural da arquitetura, que compreende os módulos de código como classes, pacotes e subsistemas no *design*. Ele também capta módulo de decomposição, herança, associações e agregações.

- **Componente e conector**

Esta visão descreve os aspectos comportamentais da arquitetura. Componentes são tipicamente objetos, linhas, ou processos, e os conectores descrever como os componentes interagem. Conectores comuns são tomadas, middleware como CORBA ou memória compartilhada.

- **Alocação**

Esta visão mostra como os processos na arquitetura são mapeados para hardware, e como eles se comunicam através de redes e bases de dados. Ela também captura uma vista para o código-fonte dos sistemas de gerenciamento de configuração, e que no grupo de desenvolvimento tem a responsabilidade de cada um dos módulos.

SEI

Software Engineering Institute (SEI) é um centro de pesquisa e desenvolvimento patrocinado pelo Departamento de Defesa dos Estados Unidos da América que provê uma prática avançada de engenharia de software qualificando graus de qualidade de software.

11

3 - Arquitetura e Tecnologia

Os arquitetos devem tomar decisões de *design* no início de um ciclo de vida do projeto. Muitos destas decisões são difíceis de serem validadas e testadas até que o sistema, ou pelo menos parte do sistema, seja efetivamente construída. Realizar a prototipação criteriosa de componentes chaves para a arquitetura pode ajudar a aumentar a confiança em uma determinada abordagem de *design*, mas às vezes ainda é difícil ter certeza do sucesso de uma escolha de *design* especial em um determinado contexto de aplicação.

Devido à dificuldade de validar decisões iniciais do projeto, os arquitetos podem optar, de forma sensata, por abordagens experimentadas e testadas para solucionar certas classes de problemas. Este é um dos grandes valores de padrões de arquitetura. Eles permitem aos arquitetos reduzir riscos, aproveitando projetos bem sucedidos com atributos de engenharia conhecidos.

Os padrões são uma representação abstrada de uma arquitetura, uma vez que eles podem ser atingidos de várias formas durante a implementação. Por exemplo, a arquitetura padrão de publicação-assinatura (*publish—subscribe*) descreve um mecanismo abstrato de baixo acoplamento, comunicações muitos-para-muitos (*many-to-many*) entre editores de mensagens e assinantes que desejam receber mensagens. No entanto, não especificam como publicações e assinaturas são gerenciadas, que protocolos de comunicação são utilizados, quais os tipos de mensagens podem ser enviados, e assim por diante. Estes são todos considerados detalhes de implementação.

Em virtude deste nível abstrato, no meio acadêmico, há uma série de discuções a respeito de modelos arquiteturais. Alguns destes modelos nem mesmo são capazes de serem implementados de forma concreta pelos engenheiros de *software*.

Felizmente, a indústria de *software* teve um direcionamento mais prático. Padrões de arquitetura amplamente utilizados são suportados em uma variedade de estruturas pré-construídas e disponíveis, seja através de tecnologia comercial, seja por meio de tecnologia *open source*. **Saiba+**

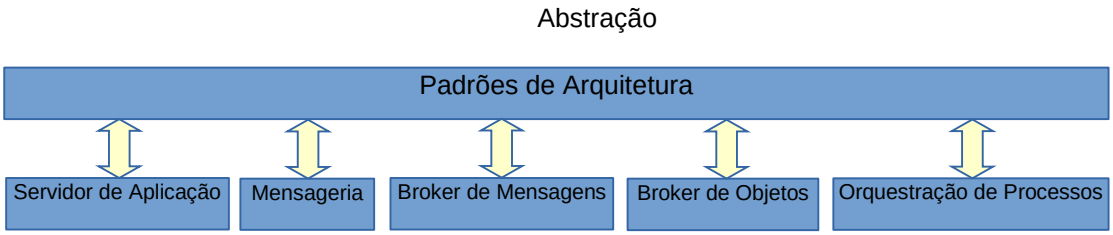
Saiba+

De qualquer forma, se um projeto necessita de uma estrutura de publicação-assinatura (*publish—subscribe*) de mensagens, broker de mensagem ou de uma arquitetura de três camadas, as opções de tecnologia disponíveis são muitas e variadas. Este é um exemplo de tecnologias que fornecem infraestruturas de *software* reutilizáveis e comprovadas tecnologicamente.

12

Como retrata a imagem abaixo, várias classes de tecnologias são utilizadas na prática para fornecer implementações de padrões de arquitetura para uso em sistemas de TI. Dentro de cada classe, produtos comerciais ou *open source* podem ser utilizados. Embora estes produtos sejam similares, eles possuem diferentes conjuntos de recursos e, por este motivo, são utilizados de forma diferente e têm diferentes restrições em relação a sua utilização.

Ilustração: refazer a imagem, mantendo as informações e aumentando a fonte para facilitar a leitura.



Mapeamento entre os padrões de arquitetura e tecnologias concretas

A diversidade de tecnologia disponível no mercado é muito grande. Esta diversidade traz vantagens e desvantagens. A concorrência entre fornecedores de produtos apresenta constantes inovações, melhores conjuntos de recursos e implementações, e preços mais competitivos, mas também dificulta a seleção de um produto que possui atributos de qualidade que satisfazem os requisitos da **aplicação**.

O desafio do arquiteto está em compreender os benefícios e os pontos fracos no início do ciclo de desenvolvimento de um projeto e escolher a mais adequada para o seu projeto. Esta não é uma tarefa fácil e os riscos e custos associados à seleção de uma tecnologia inadequada são elevados. A história da indústria de *software* está cheio de escolha mal feita e, consequentemente, de projetos fracassados.

Aplicação

Todas as aplicações são diferentes em alguns aspectos, e raramente existe um produto que atenda por completo todas as necessidades do projeto. Diferentes implementações de tecnologia têm diferentes conjuntos de pontos fortes e fracos e custos e, consequentemente, serão mais adequados para alguns tipos de aplicações do que outros.

13

4 - Arquitetura de *Software* em Projetos Ágeis

Atualmente, muito tem se falado de **métodos ágeis de desenvolvimento de *software***. O mercado identificou estas metodologias como a solução para entregar projetos de *software* que efetivamente entregam valor para o usuário final de maneira contínua e de forma mais rápido. Métodos ágeis de desenvolvimento de *software* prometem apoiar feedback contínuo e absorver mudanças nos requisitos em todo o ciclo de vida de desenvolvimento de *software*, através de uma estreita colaboração entre clientes e desenvolvedores e permite entregas frequentes de recursos de *software* necessários para um sistema.

Os métodos ágeis de desenvolvimento de *software* são baseados no *Agile Manifesto*, que foi publicado por um grupo de desenvolvedores e consultores de *software* em 2001. Este manifesto descreve os valores fundamentais que sustentam pontos de vista da comunidade ágil sobre diferentes aspectos dos processos de desenvolvimento de *software*, pessoas, práticas e artefatos.

Alguns dos métodos ágeis de desenvolvimento de software mais conhecidos são:

- Extreme Programming (XP) e
- Scrum.

Por este motivo, vamos limitar nossa discussão a respeito dos princípios e práticas de arquitetura para estas duas metodologias. Com isso, esperamos fornecer uma discussão mais específica, coerente e profunda de como fazer métodos ágeis e práticas de arquitetura trabalhar em harmonia para alavancar as vantagens de ambas as disciplinas para o desenvolvimento de alta qualidade e *software* de baixo custo forma iterativa e incremental, sem atrasos desnecessários e riscos do projeto.

14

4.1 Scrum

Scrum tem emergido como um dos principais, se não a principal metodologia ágil de desenvolvimento de sistema.

Scrum é um termo usado no jogo do rugby que significa "fazer uma bola fora do jogo voltar para o jogo" através de esforços da equipe. No desenvolvimento de software, Scrum é uma abordagem de gerenciamento de projeto iterativo e incremental que fornece uma simples inspeção e adaptação ao invés de técnicas específicas.

A formação scrum no rugby é utilizada para reinício do jogo em certos casos.

15

Projetos baseados em Scrum entregam *software* em incrementos chamados ***sprints*** (geralmente 3-4 iterações semana, ou até duas semanas em alguns casos). Cada *sprint* começa com o planejamento, durante o qual as histórias de usuário são tomadas a partir *backlogs* com base nas prioridades, e termina com uma revisão *sprint*. É esperado que a atividade de planejamento dure por algumas horas (por exemplo, 4 horas). A reunião de revisão do *sprint* também pode durar cerca de 4 horas. Todos os principais interessados são esperados para participar do planejamento do *sprint* e das reuniões de revisão do *sprint*, realizadas no início e no fim de cada *sprint*, respectivamente.

Uma equipe scrum realiza uma breve reunião (por exemplo, no máximo 15 minutos) no início de cada dia. Esta reunião é chamada de "reunião diária do scrum", e é destinada a permitir que cada membro da equipe responda a apenas três perguntas:

- O que eu fiz ontem?
- O que farei hoje?
- Quais são impedimentos para o meu trabalho?

Cada projeto scrum deverá ter, pelo menos, três artefatos:

- ***backlog* de produtos**,

- ***backlog* de sprint**,
- e o **gráfico de *burn-down***, que tem como objetivo fornecer um relatório da situação em termos de trabalho acumulado ainda a ser feito.

A comunidade de arquitetura de *software* tem emprestado o termo ***backlog*** e propôs que o processo de arquitetura também deve manter um *backlog* para projetar e avaliar a arquitetura do *software* de forma iterativa.

Os *backlog*s do Scrum consistem em requisitos que precisam ser implementados durante os ciclos de sprint atuais ou futuras. Assim, definir um *backlog*s de arquitetura traz uma abordagem iterativa e incremental para a arquitetura.

Sprint

Um sprint é a unidade básica de desenvolvimento em Scrum. Sprints tendem a durar entre uma semana e um mês, e são um esforço dentro de uma faixa de tempo (ou seja, restrito a uma duração específica) de comprimento constante. Cada sprint é precedido por uma reunião de planejamento (*Sprint Planning*), onde as tarefas para o sprint são identificadas e um compromisso estimado para o objetivo do sprint é definido e seguido por uma reunião de revisão ou de retrospectiva, onde o progresso é revisto e lições para os próximos sprints são identificadas.

16

4.2- Programação Extrema (XP – eXtream Programming)

Programação Extrema ou simplesmente **XP** é outra abordagem ágil muito popular que foi desenvolvida com base em princípios e práticas adotadas em níveis extremos de senso comum. Como outros métodos ágeis, o XP também defende iterações curtas e lançamentos frequentes de código com o objetivo de aumentar a produtividade e acomodar as mudanças de requisitos.

XP foi projetado para equipes de oito a dez desenvolvedores que trabalham com a linguagem de programação orientada a objeto. A abordagem rapidamente se tornou popular, entre os desenvolvedores de *software* que não estavam satisfeitos com as abordagens de desenvolvimento de *software* tradicionais.

Entre as **principais práticas XP** encontram-se:

- **Jogo de planejamento;**
- **Pequenas entregas;**

- **Projeto simples;**
- **Testes;**
- **Refatoração;**
- **Programação em pares;**
- **Integração contínua;**
- **Propriedade coletiva;**
- **Cliente no local;**
- **Quarenta horas por semana;**
- **Apenas governa.**

XP

Os cinco valores fundamentais da metodologia **XP** são: comunicação, simplicidade, *feedback*, coragem e respeito. A partir desses valores, possui como princípios básicos: *feedback* rápido, presumir simplicidade, mudanças incrementais, abraçar mudanças e trabalho de qualidade.

Jogo de planejamento

Uma estreita interação entre clientes e desenvolvedores é incentivada para estimar e priorizar requisitos para o próximo lançamento. Os requisitos são capturados como histórias dos usuários sobre cartões de história. Os programadores são esperados para planejar e entregar apenas as histórias do usuário acordados com os clientes.

Pequenas entregas

Uma versão inicial de um sistema é liberada para operação após algumas iterações. Novos recursos são entregues em versões subsequentes em uma base diária ou semanal.

Projeto simples

XP incentiva os desenvolvedores a manter o projeto de um sistema tão simples quanto possível.

Testes

O primeiro teste significa que os desenvolvedores escrevem testes de aceitação para o seu código antes de escrever o código em si. Os clientes escrevem testes funcionais para cada iteração, e no fim de cada iteração, é esperado que todos os testes sejam executados com êxito.

Refatoração

O projeto de um sistema evoluiu, transformando o *design* do sistema de uma forma que todos os casos de teste sejam executados com êxito.

Programação em pares

O código de produção é escrito por dois desenvolvedores.

Integração contínua

Tudo novo código é integrado ao sistema de forma constante. Todos os testes funcionais ainda devem ocorrer após a integração ou o novo código é descartado.

Propriedade coletiva

Todos os desenvolvedores trabalhando em conjunto. Isso significa que qualquer desenvolvedor pode fazer alterações em qualquer parte do código a qualquer momento tal for considerado necessário.

Cliente no local

Um cliente senta-se com a equipe de desenvolvimento o tempo todo. O cliente responde a perguntas no local, realiza testes de aceitação, e assegura o progresso no desenvolvimento.

Quarenta horas por semana

Se alguém da equipe de desenvolvimento tem que trabalhar horas extras em duas semanas consecutivas, é um sinal de um grande problema. Os requisitos devem ser selecionados para cada iteração de uma forma que os desenvolvedores não necessitem trabalhar horas extras.

Apenas governa

Os membros de uma equipe de subscrever um conjunto de regras. As regras podem ser alteradas a qualquer momento, desde que haja um consenso sobre a forma de avaliar os efeitos da mudança.

4.3- Fazendo a arquitetura e as abordagens agéis funcionarem

Com a popularização das metodologias ágeis, está ocorrendo um reconhecimento crescente da importância de prestar mais atenção aos **aspectos arquitetônicos**. Assim, está aumentando o número de esforços destinados a identificar os desafios técnicos e organizacionais envolvidos na integração de abordagens ágeis de desenvolvimento de *software* em métodos tradicionais.

Esses esforços resultaram em várias propostas para combinar os pontos fortes dos elementos centrais de cada abordagem de desenvolvimento de *software*.

Mesmo com estes esforços, ainda existem lacunas a serem preenchidas quando falamos em integrar processos ágeis e processo mais centrados em arquitetura.

Pode-se argumentar que um dos pré-requisitos importantes para eliminar a lacuna existente entre as abordagens ágil e arquitetônica é construir e divulgar uma base de conhecimento sobre os pontos de conflito e de conciliação entre os princípios. Tal base de conhecimento deve incluir também os desafios, os problemas e riscos que as equipes de desenvolvimento de *software* enfrentam quando tentam seguir os princípios de projetos ágeis que não incorporam as práticas de uma abordagem centrada em arquitetura.

18

Várias observações foram feitas sobre a combinação de abordagens ágeis e arquitetônicas, alguns desses resultados foram compilados através de práticas ágeis bem conhecidas, juntamente com práticas arquitetônicas para mostrar que muitas das práticas ágeis têm princípios ou práticas equivalentes, e estas podem ser facilmente adaptadas e aplicadas em ambientes ágeis.

Observe a tabela a seguir.

Práticas Ágeis	Abordagem arquitetural
Sprint	A natureza iterativa do *design* de uma arquitetura de *software* pode ser tratada através de um *backlog* das preocupações arquiteturais serem tratadas.
Sprint planning	Priorização de requisitos significantes arquiteturalmente para cada iteração.
Sprint review	Revisão arquitetural

Daily meetings	Compartilhar a lógica e o conhecimento da arquitetura através de reunião de grupo arquitetural
Onsite customer	Envolver os principais interessados durante a definição da arquitetura o máximo possível
Continuous integration	*Architecture-level integration and interoperability—quality attribute approaches* Integração em nível de arquitetura e interoperabilidade com uma abordagem de atributos de qualidade.
Refactoring	Refatoração em nível de arquitetura usando padrões e estilos de arquitetura.
Simple *design*	*Design* baseado em padrões para garantir sua simplicidade e que seja bem conhecido
Collective code ownership	Garantir que os principais interessados comprem a ideia das principais decisões arquiteturais
Coding standards	Utilização de templates e padrões arquiteturais para facilitar a apoio e padronização
Test-driven development	Testes baseados na arquitetura

19

Para garantir uma **integração entre metodologias ágeis e uma abordagem centrada em arquitetura**, tem se observado uma ênfase crescente sobre o papel vital dos arquitetos de *software*.

Os arquitetos devem agir como facilitadores dos projetos de desenvolvimento de *software* e como os representantes de atributos de qualidade global de um sistema.

Para agir como facilitador, o arquiteto deve ter as seguintes **características**:

- **Ter uma boa compreensão das abordagens ágeis.**
- **Saber como convencer o proprietário do produto (*product owner*) a respeito de uma decisão de *design* em situações conflitantes.**
- **Conhecer a arquitetura global, recursos necessários e o estado de implementação.**
- **Documentar e comunicar a arquitetura a todas as partes interessadas.**
- **Deve estar disposto a exercer múltiplos papeis como: arquiteto de solução, arquiteto de *software*, desenvolvedor etc.**

- **Liderar um esforço para institucionalizar o papel de arquitetos como facilitadores e prestadores de serviços para os projetos.**

20

É importante ter em mente que os arquitetos de *software* geralmente projetam a arquitetura, mas são os desenvolvedores que materializam a arquitetura projetada. Os desenvolvedores de *software* devem ser igualmente responsáveis pelo tratamento da arquitetura como uma entidade de primeira classe que fornece o projeto de todo o sistema. Assim, o papel dos desenvolvedores de *software* é igualmente importante no sucesso da combinação das abordagens ágeis e arquitetural.

A integração das metodologias ágeis e da arquitetura de *software* não é algo dominado pelo mercado onde já existe um conjunto de práticas bem definidas e consolidadas. Entretanto, histórias de sucesso têm crescido e à medida que o número de projetos bem sucedidos aumenta, as melhores práticas também aumentam.

Nesa disciplina tivemos apenas uma pequena introdução sobre o assunto. Desta forma, é importante que vocês pesquisem e se aprofundem neste assunto que, sem dúvida, vai se tornar cada vez mais relevante na área de tecnologia, sobretudo em relação aos processos de desenvolvimento de *software*.

21

RESUMO

Neste módulo podemos explorar com um pouco mais de profundidade o conceito de arquitetura de *software*. Onde podemos observar que durante a definição da estrutura do sistema busca-se minimizar as dependências entre os componentes, criando uma arquitetura de baixo acoplamento.

Vimos também que, se usados adequadamente em uma arquitetura, a utilização de padrões alavanca o conhecimento de *design* e torna mais fácil entender o *design* do sitema.

Foi abordado também que a arquitetura de *software* deve abordar explicitamente os aspectos não funcionais de um ssitema.

Vimos que um dos mecanismos mais poderosos para descrever uma arquitetura é a decomposição hierárquica, onde inicialmente temos uma documentação informal com a estrutura e as interações do sistema de forma mais ampla e estes componentes são decompostos em mais detalhe na documentação que acompanha o *design*.

As diferentes formas de visualizarmos a arquitetura são denominadas visões, as quais estão divididas em: Visão lógica, Visão de processo, Visão física e Visão de desenvolvimento.

Vimos também que a indústria de *software* teve um direcionamento prático onde uma infinidade de estruturas pré-construídas estão disponíveis seja através de tecnologias comerciais ou através de tecnologia open source.

Foi abordado também que, à medida que as metodologias ágeis tornam-se mais populares, é importante que a arquitetura e estas metodologias possam se unir para trazer os benefícios de ambas abordagem para os projetos de desenvolvimento de *software*.

UNIDADE 1 – CONTEXTO DA ARQUITETURA DE SOFTWARE
MÓDULO 3 – ARQUITETURA CORPORATIVA X ARQUITETURA DE SOFTWARE
01

1- O QUE É ENTERPRISE ARCHITECTURE?

Nossa disciplina diz respeito à arquitetura de *software*. Entretanto, na área de tecnologia da informação temos outro tipo de arquitetura, a **arquitetura corporativa**. Nesta etapa do nosso estudo vamos explorar a arquitetura corporativa, os principais frameworks e como a arquitetura de *software* está relacionada à arquitetura corporativa.

A arquitetura corporativa pode ser vista como um modelo para o posicionamento ideal de recursos no ambiente de TI para o apoio fundamental da função de negócios.

O objetivo da arquitetura corporativa é criar um ambiente de TI unificado (sistemas de *hadware* e *software* padronizados) em toda a empresa ou todas as unidades de negócios da empresa direcionadas ao negócio da organização e sua estratégia.

Mais especificamente, os objetivos são promover o alinhamento, a normalização, a reutilização de ativos de TI existentes, bem como compartilhar os métodos comuns para a gestão de projetos e desenvolvimento de *software* em toda a organização. O resultado final, teoricamente, é que a arquitetura corporativa fará a TI da organização mais barata, mais alinhada estrategicamente à organização e mais ágil.

A finalidade da arquitetura corporativa é criar um mapa de ativos de TI e processos de negócios e um conjunto de princípios de governança que impulsionem uma discussão em curso sobre a estratégia de negócio e como ela pode ser expressa através da TI.

02

Na literatura é possível encontrar diferentes sugestões de como uma arquitetura corporativa deve ser elaborada. No entanto, em todas estas sugestões, ou pelo menos na maioria, **quatro domínios** fundamentais são comuns:

- **Arquitetura de negócios**

Documentação que descreve os processos de negócios mais importantes da empresa.

- **Arquitetura de informação**

Identifica onde os blocos de informação importantes, como um registro de cliente, são mantidos e como um tipicamente acessá-los.

- **Arquitetura do sistema de aplicação**

Um mapa das relações de aplicações de *software* para o outro.

- **Arquitetura de tecnologia de infraestrutura**

Um modelo para a gama de *hadware*, sistemas de armazenamento e redes. A arquitetura de negócios é o mais crítico, mas também o mais difícil de implementar, de acordo com profissionais da indústria.

Os domínios apresentados para a arquitetura empresarial, principalmente quando organizados em camadas, provaram ser úteis porque têm a vantagem de proporcionar uma visão mais contida e mais focada em um domínio, o que evita uma visão sobreposta do ambiente de TI da organização.

03

Há uma série de modelos de arquitetura corporativa disponíveis no mercado. Veja alguns deles.

No entanto, não existe um consenso sobre qual modelo é o mais completo e qual deve ser utilizado pelas empresas. Para tornar a missão de selecionar um modelo a ser adotado ainda mais difícil, novos modelos estão surgindo ao longo do tempo. Existe inclusive um livro com o título “Como sobreviver na selva de Frameworks para Arquitetura Empresarial: Criando ou Escolhendo um Framework de Arquitetura Empresa”.

Fundamentalmente, todos os modelos buscam, de alguma maneira fazer uso da noção de serviço e arquitetura genérica onde as funções estão agrupadas em módulos de serviços reutilizáveis. Os recursos mais complexos são construídos a partir da utilização adequada destes módulos mais básicos.

Alguns dos modelos de arquitetura empresarial têm sua origem no modelo **cliente-sevidor** desenvolvido no final de 1980 e início de 1990. No entanto, alguns dos conceitos foram modernizados; por exemplo, o cliente pode ser um dispositivo de acesso baseado em *browser*, o servidor pode ser um servidor Web, e o protocolo de comunicação poderia ser o HTTP (Hypertext Transfer Protocol); ou ambas as entidades poderiam ser um servidor executando algum Web Service.

No entanto, nós profissionais de TI, temos que ter uma visão mais pragmática e menos acadêmica sobre estes modelos. Caso contrário, pode-se acabar por gastar uma enorme quantidade de tempo ao longo de vários anos para desenvolver um modelo a ser utilizado e, pior, que tem pouco de concreto para mostrar.

Veja alguns deles.

A seguir é exibida uma lista de modelos de arquitetura corporativa:

- Zachman Enterprise Architecture Framework (ZIFA)
- The Open Group Architecture Framework (TOGAF)
- Extended Enterprise Architecture Framework (E2AF)
- Enterprise Architecture Planning (EAP)
- Federal Enterprise Architecture Framework (FEAF)
- Treasury Enterprise Architecture Framework (TEAF)
- Integrated Architecture Framework (IAF)
- Joint Technical Architecture (JTA)
- Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance (C4ISR) and DoD Architecture Framework (DoDAF)
- Department of Defense Technical Reference Model (DoD TRM)
- Technical Architecture Framework for Information Management (TAFIM)
- Computer Integrated Manufacturing Open System Architecture (CIMOSA)
- Purdue Enterprise Reference Architecture (PERA)
- Standards and Architecture for eGovernment Applications (SAGA)
- European Union—IDABC & European Interoperability Framework
- ISO/IEC 14252 (IEEE Std 1003.0)
- IEEE Std 1471-2000 IEEE Recommended Practice for Architectural Description

A tentativa para modelar uma empresa inteira com qualquer um dos modelos disponíveis no mercado pode ser um esforço extremamente entediante e pode se tornar um ponto discutível quando o ambiente de negócios e mesmo o mercado muda constantemente.

Por causa do esforço envolvido, uma empresa pode investir grande quantidade de tempo na realização de sua arquitetura corporativa e, no momento que a concluir, pode descobrir que ela já está desatualizada. Assim, o que temos que buscar é uma boa **modelagem**. O esforço para se produzir um modelo perfeito, extremamente preciso pode se mostrar um grande desperdício.

Em virtude de todo o contexto apresentado, não são apenas as grandes organizações que adotaram a arquitetura empresarial. As pequenas organizações também estão adotando essa abordagem. No entanto, o prazo com que a arquitetura corporativa necessita ser revisado e atualizado também se torna menor que empresas de grande porte.

Toda organização que procura gerir a sua complexidade de TI de forma eficaz, em termos de custo para implantação do sistema rápido, deve considerar fazer os investimentos adequados na arquitetura corporativa.

05

2- MOTIVAÇÕES PARA TER UMA ARQUITETURA CORPORATIVA

A principal razão para o desenvolvimento de uma arquitetura corporativa é apoiar o lado comercial da organização, fornecendo a tecnologia fundamental e estrutura de processo para uma estratégia de TI. Este, por sua vez, torna um ativo sensível para uma estratégia de negócios bem sucedida e atualizada.

Como a tecnologia da informação estendeu seu alcance em todas as áreas da empresa e os sistemas, aplicativos, bancos de dados, servidores, redes e dispositivos de armazenamento tornam-se altamente interligados e interdependentes na lógica e no nível físico, os profissionais de TI precisam reconhecer a crescente importância da arquitetura empresarial para o contínuo sucesso e crescimento da empresa.

Quando feita corretamente, a arquitetura empresarial fornece uma avaliação sistemática e uma descrição de como a função de negócios opera no momento atual:

- ela fornece um "projeto" de como deve operar no futuro,
- e um roteiro para chegar ao estado de destino.

Trabalhar a arquitetura corporativa é importante por pelo menos **três razões**:

a) permite a comunicação entre os *stakeholders*,
b) facilita as decisões iniciais do projeto, e
c) cria uma abstração transferível de uma descrição do sistema / ambiente.

A arquitetura corporativa irá ajudar as empresas a gerenciar seus custos de TI; ele ajuda a organização a decidir onde fazer novos investimentos em TI, ou seja, o que deve ser conservado e o que deve ser aposentado ou reconstruído, sejam aplicativos ou infraestrutura.

06

Mesmo entendendo a importância e a motivação de se ter uma arquitetura comporativa, ter um conjunto de declarações escritas rotuladas como princípios não significa que os princípios sejam bons, ainda que as pessoas na organização concordar com ele. Um bom conjunto de princípios se baseia em crenças e valores da organização e é expresso numa linguagem que o lado comercial da empresa entende e usa. Por outro lado, é importante que o lado do negócio entenda o orçamento da área de TI e o motivo pelo qual os recursos estão sendo destinados a esta área.

Os princípios da arquitetura corporativa precisam conduzir o comportamento dentro da organização de TI. Princípios devem ser em número reduzido, ser voltada para o futuro, e ser aprovado e defendido pelos gestores de TI e de negócio. Eles fornecem uma base para a tomada de decisões de Arquitetura e Planejamento, elaboração de políticas, procedimentos e padrões, e apoiam a resolução de situações contraditórias.

Um conjunto de princípios ruim cairá rapidamente em desuso, e as políticas e normas soarão como arbitrárias e, portanto, carecerão de credibilidade. Existem vários **critérios que caracterizam um bom conjunto de princípios**. São eles:

- Compreensibilidade
- Robustez
- Abrangência
- Consistência
- Estabilidade

Compreensibilidade

Os princípios subjacentes podem ser rapidamente apreendidos e compreendidos por pessoas em toda a organização. A intenção do princípio é clara e inequívoca, de modo que as violações sejam reduzidas.

Robustez

Subsidia a tomada de decisão de boa qualidade sobre os planos a serem realizados, e sobre as políticas e normas a serem criadas. Cada princípio deve ser suficientemente preciso para apoiar a tomada de decisões consistentes em situações complexas e potencialmente controversas.

Abrangência

Todo princípio potencialmente importante que rege a gestão de informação e tecnologia para a organização está definido. Os princípios devem cobrir todas as situações percebidas.

Consistência

Os princípios não devem ser contraditórios ao ponto em que aderir a um princípio violaria o espírito de outro. Cada palavra em um princípio deve ser cuidadosamente escolhida para permitir uma interpretação coerente, mas flexível.

Estabilidade

Princípios devem ser duradouros, ainda capazes de acomodar as mudanças. Deve ser estabelecido um processo de emenda para adicionar, remover ou alterar princípios depois de serem ratificados inicialmente.

07

A estratégia de uma arquitetura corporativa, ainda traz vantagens técnicas capazes de proporcionar **vantagens competitivas importantes**, como:

- Uma operação de TI mais eficiente.
- Menores custos de desenvolvimento de *software*, suporte e manutenção.
- O aumento da portabilidade de aplicações.
- Melhor interoperabilidade e sistema mais fácil e gerenciamento de rede.
- Uma melhor capacidade para abordar questões críticas em toda a empresa, como a segurança.
- Atualização mais fácil e troca de componentes do sistema.
- Melhor retorno sobre o investimento existente e redução do risco para futuros investimentos.
- Complexidade reduzida em infraestrutura de TI.
- O máximo retorno sobre o investimento em infraestrutura de TI existente.
- A flexibilidade de fazer, comprar, ou terceirizar soluções de TI.
- Redução do risco global em novos investimentos e os custos de propriedade de TI.
- Aquisições mais rápidas, mais simples, mais baratas.

- Decisões de compra mais simples, porque a informação que rege os contratos está prontamente disponível em um plano coerente.

08

3- PRINCIPAIS FRAMEWORKS DO MERCADO

Atualmente, os modelos mais utilizados pela indústria são **Zachman**, seguido por **TOGAF** e **DoDAF**. Vamos explorar cada um destes modelos. É importante ressaltar que o objetivo é apenas apresentar brevemente cada um destes frameworks.

3.1 ZACKMAN

Este framework recebeu o nome de seu criador John Zachman, que foi o primeiro a desenvolver o conceito de arquitetura corporativa nos anos 80, enquanto trabalhava na IBM. Mas este modelo tem sido atualizado constantemente desde a sua criação.

O Framework Zachman é uma estrutura fundamental que define uma maneira estruturada e formal para descrever uma arquitetura corporativa.

Este modelo define a infraestrutura de informação de uma empresa a partir de uma matrix de duas dimensões. A primeira vem das questões primordiais: O que, Quem, Quando, Onde, e Por que. A segunda vem da perspectiva dos principais *stakeholders* (interessados):

- Planejador,
- Proprietário,
- Projetista,
- Construtores,
- Subcontratados, e
- Sistema de trabalho.

Não há orientação em sequência, processo ou aplicação do quadro. O foco está em garantir que todos os aspectos de uma empresa sejam bem organizados e apresentem relações claras que assegurem um sistema completo, independentemente da ordem em que estão estabelecidas.

09

3.1.2 Estrutura do Framework

O Framework Zachman pretende proporcionar uma compreensão de qualquer aspecto particular de um

sistema em qualquer ponto do seu desenvolvimento. A ferramenta pode ser útil na tomada de decisões sobre as alterações ou ampliações no que se refere à tecnologia de informação para a organização.

A ideia básica por trás do Framework Zachman é que a mesma coisa, seja ela simples ou complexa, possa ser descrita para fins diferentes de formas diferentes, inclusive utilizando diferentes tipos de descrições (por exemplo, textual, gráfica). Como uma matriz de duas dimensões com 6 linhas e 6 colunas, o Framework Zachman fornece trinta e seis categorias para descrever completamente a arquitetura corporativa.

Ele permite que pessoas diferentes olhem para a mesma coisa de diferentes perspectivas. Isso cria uma visão holística do ambiente.

	DADOS (O que?)	FUNÇÃO (Como?)	REDE (Onde?)	PESSOAS (Quem?)	TEMPO (Quando?)	MOTIVAÇÃO (Por que?)
ESCOPO	Dados importantes do Contexto de Atuação	Processos de Negócios com Entidades Externas	Locais de atuação da organização	Entidades Externas importantes para o Negócio	Eventos Significativos para o Contexto	Objetivos do Contexto de Negócio
MODELO DE NEGÓCIO	Dados importantes do Modelo de Negócio da organização	Processos da Arquitetura de Negócio (Ambiente Interno)	Áreas da organização – componentes da arquitetura de Negócio	Entidades Gestoras dos Processos de Negócios	ventos Significativos para o Negócio	Plano de Direcionamento Estratégico da Organização
MODELO DE SISTEMAS DE INFORMAÇÃO	Dados importantes para Arquitetura de Sistemas	Processos automatizados pela Arquitetura de Sistemas	Integração entre Sistemas	Entidades Gestoras dos Sistemas	Eventos importantes para os Sistemas	Objetivo dos Sistemas de Informação
MODELO DE TECNOLOGIA	Itens de Configuração da Arquitetura Tecnológica	Funcionalidades da Estrutura Tecnológica	Integração entre ambientes operacionais	Entidades gestoras dos Ambientes	Eventos importantes para a Arquitetura Tecnológica	Infra-estrutura do Ambiente Operacional
DESCRIÇÃO DETALHADA	Requisitos e Especificações	Processos de Gerenciamento	Cadeia de Fluxo de Valor	Gestores de Serviços	Acordos de Níveis de Serviço (Prazo)	Produtos e Serviços de Valor Agregado

Modelo Zachman

As duas primeiras linhas são intensamente orientadas para **negócios** e podem ser expressas em vocabulários orientados a negócios, enquanto que as quatro linhas inferiores estão no **domínio técnico**.

Conceitos de negócio da linha superior devem ser incorporados a objetos de negócios e componentes nas linhas de fundo. Os conceitos de negócio podem ser refinados ao longo do tempo, mas seus relacionamentos não devem ser alterados.

Os requisitos são capturados por coluna, e os agentes são associados com a coluna. Como a ordem das colunas não tem nenhum significado estabelecido, eles poderiam ser reorganizados para acompanhar a necessidade da empresa.

Escopo
Corresponde a um resumo executivo de um planejador que quer uma estimativa do tamanho, custo e funcionalidade do sistema.

Modelo de negócio
Mostra todas as entidades e processos de negócio e como eles interagem.

Modelo do sistema
Usado por um analista de sistemas que devem determinar os elementos de dados e funções de *software* que representam o modelo de negócio.

Modelo de tecnologia
Considera as limitações das ferramentas, tecnologia e materiais.

Componentes
Representam módulos individuais, independentes, que podem ser atribuídoss aos empreiteiros para a implementação.

Sistema de trabalho
Descreve o sistema operacional.

Quando
Representa o tempo, ou as relações de eventos que estabelecem critérios de desempenho e níveis quantitativos para os recursos da empresa. Isso é útil para a concepção do plano- mestre, a arquitetura de processamento, arquitetura de controle e dispositivos de temporização.

Quem
Representa os povos relacionamentos dentro da empresa. O projeto da organização empresarial tem a ver com a atribuição de trabalho e da estrutura de autoridade e responsabilidade. A dimensão vertical representa delegação de autoridade, ea horizontal representa a atribuição de responsabilidade.

Por que
Descreve as motivações da empresa. Isso revela os objetivos da empresa e objetivos, plano de negócios, arquitetura de conhecimento, e conhecimento de design.

O que
Descreve as entidades envolvidas em cada perspectiva da empresa. Exemplos incluem objetos de negócios, dados do sistema, tabelas relacionais ou definições de campo.

Como
Mostra as funções dentro de cada perspectiva. Exemplos incluem processos de negócio, função de aplicação de *software*, função de *hadware* de computador e malha de controle de idioma.

Onde
Mostra locais e interligações dentro da empresa. Isso inclui os principais locais de visita geográficas, seções separadas dentro de uma rede logística, alocação de nós do sistema, ou até mesmo endereços de memória dentro do sistema.

10

O Framework tem um conjunto de 7 regras básicas:

Regra 1 - **As colunas não têm Ordem**

As colunas são intercambiáveis, mas não podem ser reduzidas ou criadas.

Regra 2 - **Cada coluna tem um modelo genérico simples**

Cada coluna pode ter sua própria meta-modelo.

Regra 3 - **O modelo básico de cada coluna deve ser exclusivo**

O modelo básico de cada coluna, os objetos de relacionamento e a estrutura são únicas. Cada objeto de relacionamento é interdependente, mas o objetivo da representação é único.

Regra 4 - **Cada linha descreve uma perspectiva distinta, única**

Cada linha descreve o ponto de vista de um grupo empresarial em particular e é exclusivo para ele. Todas as linhas são geralmente presentes na maioria das organizações hierárquicas.

Regra 5 - **Cada célula é única**

A combinação de 2,3 e 4 devem produzir células únicas, onde cada célula representa um caso particular.

Regra 6 - **O composto ou a integração de todos os modelos de células em uma linha constitui um modelo completo a partir da perspectiva da referida linha**

Pela mesma razão, como por não adicionar linhas e colunas, mudando os nomes podem alterar a estrutura lógica fundamental do quadro.

Regra 7 - **A lógica é recursivo**

A lógica é relacional entre duas instâncias da mesma entidade.

11

3.2 The Open Group Architectural Framework - TOGAF

The Open Group Architectural Framework, ou simplesmente TOGAF, foi criado pelo The Open Group no início dos anos 90.

TOGAF desempenha um papel importante para ajudar a sistematizar o processo de desenvolvimento de arquitetura, permitindo que os usuários possam construir soluções baseadas em sistemas abertos para as suas necessidades de negócios. Empresas que não adotam em uma arquitetura corporativa acabam adotando uma solução de um único fornecedor na busca de uma solução integrada. Com isso a empresa deixa de obter os benefícios, inclusive financeiros, de ter diferentes fornecedores de sistemas possíveis de serem integrados através de uma arquitetura verdadeiramente aberta.

Normalmente, uma arquitetura é desenvolvida porque as pessoas têm preocupações fundamentais que precisam ser abordadas pelos sistemas de TI dentro da organização. Tais pessoas são comumente referidas como as partes interessadas no sistema.

O papel do arquiteto é observar e atender às preocupações externadas pelos usuários dos sistemas de TI. O arquiteto atende às preocupações dos usuários identificando e refinando seus requisitos, apresentando diferentes visões da arquitetura que mostram como as preocupações e as necessidades vão serão atendidas, e mostrando as mudanças e concessões necessárias para que preocupações potencialmente conflitantes possam ser atendidas. Sem a participação do arquiteto é improvável que todas as preocupações e requisitos sejam atendidos.

The Open Group

The Open Group é um consórcio de empresas de tecnologia. Atualmente, conta com mais de 400 membros. Os serviços prestados incluem estratégia, gestão, inovação e investigação, normas, certificação e desenvolvimento de testes. Entre os membros encontram-se empresas como IBM, HP, Oracle, SAP, Nasa e Departamento de Defesa Americano.

12

TOGAF é composto de três partes principais:

A) O TOGAF Architecture Development Method (ADM), que explica como obter uma arquitetura empresarial específica da organização que aborda os requisitos de negócios. A ADM fornece o seguinte:

- Uma maneira confiável e comprovada de desenvolvimento da arquitetura;
- Visões de arquitetura que permitem ao arquiteto garantir que um conjunto complexo de requisitos seja tratado de forma adequada;
- Vínculos a estudos de casos práticos;
- Orientações sobre ferramentas para arquitetura de desenvolvimento.

B) A Enterprise Continuum, um "repositório virtual" de todos os ativos de modelos de arquitetura, padrões, descrições arquitetura etc., que existem tanto dentro da empresa quanto na indústria de TI em geral, que a empresa considera ter disponível para o desenvolvimento de arquiteturas.

O TOGAF fornece dois modelos de referência para consideração:

Modelo de Arquitetura **Modelo de Solução**

C) O TOGAF Resource Base, que é um conjunto de recursos-diretrizes, modelos, informações de fundo etc., para ajudar o arquiteto no uso da ADM.

TOGAF é publicado pela The Open Group em seu Web site público, e pode ser reproduzido livremente por qualquer empresa que deseje utilizá-lo para desenvolver uma arquitetura corporativa para uso dentro dessa empresa. Basicamente, as informações sobre os benefícios e limitações da aplicação existente, juntamente com os requisitos para a mudança, são combinados usando os métodos descritos no TOGAF ADM, resultando em uma "arquitetura de destino" ou conjunto de arquiteturas de destino.

Modelo de Arquitetura

São serviços e funções genéricas que fornecem uma base sobre a qual arquiteturas específicas e blocos de construção de arquitetura podem ser construídos. Esta arquitetura, por sua vez inclui:

- ***Modelo de Referência Técnica*** (TRM), que fornece um modelo e taxonomia de serviços de plataforma genéricos; e
- ***Padrões Information Base*** (SIB), uma base de dados de padrões industriais abertos que podem ser usados para definir os serviços particulares e outros componentes de uma arquitetura específico da empresa.

Modelo de Solução

É especificamente destinado a ajudar o desenvolvimento de arquiteturas que permitem e apóiam a visão de "fluxo de informação sem fronteiras."

13

3.3- The Department of Defense Architecture Framework (DoDAF)

Em meados da década de 1990, em resposta às exigências relacionadas com a articulação multisserviço e operações militares multinacionais, o Departamento de defesa discernido com a necessidade de uma formulação de um padrão arquitetônico para garantir que os sistemas militar poderia interoperar.

O objetivo do DoDAF é verificar que as descrições arquitetônicas desenvolvidas pelos vários comandos, serviços e agências são compatíveis e que, dos pontos de vista técnicos, as arquiteturas são utilizáveis e integrável em vários domínios organizacionais. Este quadro aborda o domínio militar e é usado principalmente pelo Departamento de Defesa.

Semelhante a qualquer outra estrutura de arquitetura, ele fornece as regras e orientações para o desenvolvimento e apresentação de descrições de arquitetura corporativa, incluindo artefatos. Ele fornece contribuição sobre a forma de descrever arquiteturas, mas que não fornece mecanismos em como construir e implementar uma arquitetura específica ou como desenvolver sistemas ou um processo para aquisição de sistemas.

DoDAF está organizado em três partes:

Volume I

Fornece orientação geral sobre a necessidade e a utilização de denominações de arquitetura em DoD.

Volume II

Apresenta definições dos 26 produtos contidos nos três visões do modelo.

Volume III

É um Deskbook que fornece exemplos de arquiteturas que são compatíveis, abordagens para a realização de desenvolvimento arquitetura, e outras informações de suporte. Para estar em conformidade com o framework, descrições arquitetura deve incluir o conjunto adequado de produtos e usar os termos e definições comuns especificados no quadro.

14

Como se observa, as descrições arquitetônicas de DoDAF requerem o uso de múltiplas visões, cada uma das quais transmite diferentes aspectos da arquitetura. A arquitetura integrada do DoDAF dispõe das seguintes **visões**:

- Visão operacional;

- Visão de Sistemas;
- Visão de Normas Técnicas.

Todas as visões ampliam as outras visões, fornecendo contexto, descrições e um dicionário integrado de termos.

Visão operacional

Descreve o que está acontecendo no mundo real que está a ser apoiado ou habilitado por sistemas representados na arquitetura. Atividades realizadas como parte de missões do DoD e as trocas de informações entre os profissionais associados ou organizações são os itens principais modelados em vista operacional. O ponto de vista operacional revela requerimentos para a capacidade e interoperabilidade.

Visão de Sistemas

Descreve existentes e futuros sistemas e interligações físicas que suportam as necessidades DoD documentados na visão operacional.

Visão de Normas Técnicas

Composta por catálogos padrão (comercial off-the-shelf [COTS], o governo off-the-shelf [GOTS]), peças do sistema ou componentes e suas interconexões. Essa visão aumenta os sistemas de visualizar com detalhes técnicos e as previsões de evolução da tecnologia padrão.

15

4- Arquitetura Corporativa X Arquitetura de *Software*

O que difere a arquitetura corporativa da arquitetura de *software*? Conhecer a existência destes dois conceitos e identificar qual o escopo de cada um é um conhecimento importante para todo profissional de TI.

CEOs de hoje sabem que a gestão eficaz e a exploração da informação através da TI são a chave para o sucesso do negócio, e os meios indispensáveis para alcançar vantagem competitiva.

A **arquitetura empresarial** aborda esta necessidade, fornecendo um contexto estratégico para a evolução do sistema de TI em resposta às necessidades em constante mudança do ambiente de negócios. Além disso, uma boa arquitetura corporativa permite que se atinja o equilíbrio certo entre a eficiência de TI e inovação empresarial.

Ela permite às unidades de negócios individuais inovar em segurança na sua busca de vantagem competitiva. Ao mesmo tempo, assegura as necessidades da organização de uma estratégia integrada de TI, permitindo a sinergia mais próxima possível em toda a empresa.

Já a **arquitetura de *software*** é a disciplina que faz parte da esteira de desenvolvimento de *software* responsável por definir e descrever o comportamento do sistema.

A arquitetura de *software* serve como modelo para o sistema e do projeto que visa desenvolvê-lo, definindo as atribuições do trabalho que devem ser realizado pela equipe de design e implementação. A arquitetura é a principal responsável pelas dimenções da qualidade do sistema, tais como desempenho, facilidade em absorver as mudanças e segurança, nenhuma das quais pode ser alcançada sem uma visão de arquitetura unificada.

Arquitetura é a forma para realizar a análise, cedo no desenvolvimento de software, para se certificar de que uma abordagem de projeto irá produzir um sistema aceitável. Através da construção de arquitetura eficaz, você pode identificar os riscos do projeto e mitigá-los logo no início do processo de desenvolvimento.

16

4.1- A diferença do Arquiteto de *Software* e do Arquiteto Corporativo

Após verificar a diferença entre a arquitetura corporativa e a arquitetura de *software* é natural identificarmos que são necessários profissionais com perfis diferentes para exercer e executar cada tipo de arquitetura.

Arquiteto de *Software* (como o nome sugere) se concentra em *software*, ou seja, em um assunto específico. Este profissional é responsável pela qualidade do sistema, da solução a ser entregue para a empresa.

Já o **arquiteto corporativo** é um papel de planejamento que é responsável por identificar o estado futuro do ambiente de TI de uma organização e envolver os recursos necessários para ajudar a equipe do projeto guia para entregar em direção a ela.

Assim, o arquiteto de *software* e o arquiteto corporativo são papeis muito diferentes, onde o primeiro, foca na entrega de soluções e o seguindo concentra-se em apoiar os projetos através da documentação de estado futuro e está envolvido em atividades de governança.

17

Resumo

Vimos que a arquitetura corporativa é um modelo para o posicionamento ideal de recursos no ambiente de TI de uma empresa.

Existe uma série de modelos de arquitetura corporativa disponíveis no mercado. Entretanto, não existe um consenso sobre qual modelo é o mais completo e qual o que deve ser utilizado pelas empresas.

O investimento na definição de uma arquitetura corporativa e importantes, pois proporciona:

- Uma operação de TI mais eficiente;

- Menores custos de desenvolvimento de *software*, suporte e manutenção;
- O aumento da portabilidade de aplicações;
- Melhor interoperabilidade, sistema e gerenciamento de rede mais fáceis.
- Uma melhor capacidade para abordar questões críticas em toda a empresa, como a segurança;
- Atualização mais fácil e troca de componentes do sistema;
- Melhor retorno sobre o investimento existente e redução do risco para futuros investimentos;
- Complexidade reduzida em infraestrutura de TI;
- O máximo retorno sobre o investimento em infra-estrutura de TI existente;
- A flexibilidade de fazer, comprar, ou terceirizar soluções de TI;
- Redução do risco global em novos investimentos e os custos de propriedade de TI;
- Aquisições mais rápida, mais simples, mais barata;
- Decisões de compra mais simples.

Atualmente, os modelos mais utilizados pela indústria são Zachman, TOGAF e DoDAF.

Vimos que enquanto a arquitetura corporativa fornecendo um contexto estratégico para a evolução do sistema de TI em resposta às necessidades em constante mudança do ambiente de negócios a arquitetura de *software* tem a responsabilidade de definir e descrever o comportamento do sistema.

Por fim, foi estudado que o arquiteto de *software* e o arquiteto corporativo são papeis muito diferentes, onde o primeiro foca na entrega de soluções e o segundo, concentra-se em atividades de governança de TI.

UNIDADE 1 – CONTEXTO DA ARQUITETURA DE SOFTWARE
MÓDULO 4 – PADRÕES DE ARQUITETURA DE SOFTWARE
01

1- SISTEMAS CENTRADOS NOS DADOS

Já estudamos o que é a arquitetura de *software* e sua importância. Durante nosso estudo vimos que sistemas de *software* precisam ser cuidadosamente arquitetados e avaliados a partir de várias

perspectivas para abordar adequadamente várias preocupações que afetam a qualidade do produto final.

Durante o processo para definir a arquitetura de um sistema é necessário resolver problemas de *design* de diferentes naturezas, como problemas que lidam com a estrutura do sistema lógico ou problemas que lidam com o sistema dinâmico, simultaneamente. Em todos os casos, é essencial identificar os componentes necessários, as interfaces e as responsabilidades de cada componente. É importante também modelar interações comportamentais entre eles antes de passar para o projeto detalhado.

A partir desta perspectiva, é importante usar a experiência passada com decomposições lógicas juntamente com suas interfaces ao projetar sistemas de *software*. Para este fim, os conceitos de **estilos arquitetônicos** e **padrões arquitetônicos** surgiram como a principal abordagem para alcançar a reutilização de arquitetura. Estes conceitos são fundamentais para a criação eficiente de arquiteturas de *software*, fornecendo uma estratégia global para a concepção de famílias de sistemas de *software*.

Os diferentes estilos fornecem soluções arquitetônicas, genéricos e reutilizáveis de uma forma que possa ser facilmente compreendida e aplicada a novos problemas que exigem características arquitetônicas semelhantes.

Desta forma, um padrão descreve uma solução para um problema que ocorre com frequência durante o desenvolvimento de software, podendo ser considerado como um par "problema/solução". Um padrão é um conjunto de informações instrutivas que possui um nome e que capta a estrutura essencial e o raciocínio de uma família de soluções comprovadamente bem sucedidas para um problema repetido que ocorre sob um determinado contexto e um conjunto de repercussões.

A escolha da aplicação de padrões de arquitetura para projetar algum elemento arquitetônico depende do tipo particular de sistema, requisitos e atributos de qualidade desejados. Estas características ajudam a orientar a seleção de um padrão específico em detrimento de outro. Em outros casos, **diferentes padrões podem ser utilizados em conjunto** para satisfazer as características de um sistema deixando a cargo da equipe de *design* escolher o padrão mais apropriado para o projeto. Projetistas familiarizados com certos padrões podem inclusive aplicá-los imediatamente a problemas de projeto, sem ter que redescobri-los.

Alguns padrões arquitetônicos são encontrados no mais alto nível de decomposição do sistema, sendo muito abstratos para produzir um projeto concreto do sistema; portanto, esses padrões não estão vinculados a uma implementação do sistema particular, mas podem ser associados a tipos (ou famílias) de sistemas de modo que sua solução possa ser reutilizada em vários sistemas do mesmo tipo.

As diversas classificações de padrões são apresentadas na tabela a seguir.

Tipo	Descrição
Centrado nos Dados	Sistemas que servem como um repositório centralizado para dados, permitindo que os clientes acessem e mantenham os dados.
Fluxo de Dados	Sistemas orientados em torno do transporte e transformação de um fluxo de dados.
Distribuído	Os sistemas que envolvem principalmente a interação entre várias unidades de processamento independentes ligados através de uma rede.
Interativo	Sistemas que servem ao usuário ou são voltados ao usuário.
Hierárquico	Os sistemas nos quais componentes podem ser estruturados como uma hierarquia (vertical e horizontal) para refletir diferentes níveis de abstração e responsabilidade.

Para melhor entendermos esta classificação, vamos explorar cada um destes tipos de padrões.

Abstratos

Existem padrões arquiteturais em que o nível de abstração é bastante alto, tais como: padrões de análise, padrões de projeto, padrões de código, entre outros.

02

Sistemas centrados nos dados são sistemas decompostos principalmente em torno de repositório central de dados. Portanto, as responsabilidades típicas encontradas nos componentes do sistema centrado em dados incluem um gerenciador de dados centralizado e vários componentes de trabalho. Os controles componente gerenciador de dados, fornece e gerencia o acesso aos dados do sistema, enquanto que os componentes de trabalho executam operações e executam o trabalho com base nos dados.

A comunicação em sistemas centrados nos dados é caracterizada por uma **comunicação um para um bidirecional** entre os componentes de trabalho e os componentes do gerenciador de dados. Assim, os componentes de trabalhado não interagem uns com os outros diretamente. Toda a comunicação passa pelo gerenciador de dados.

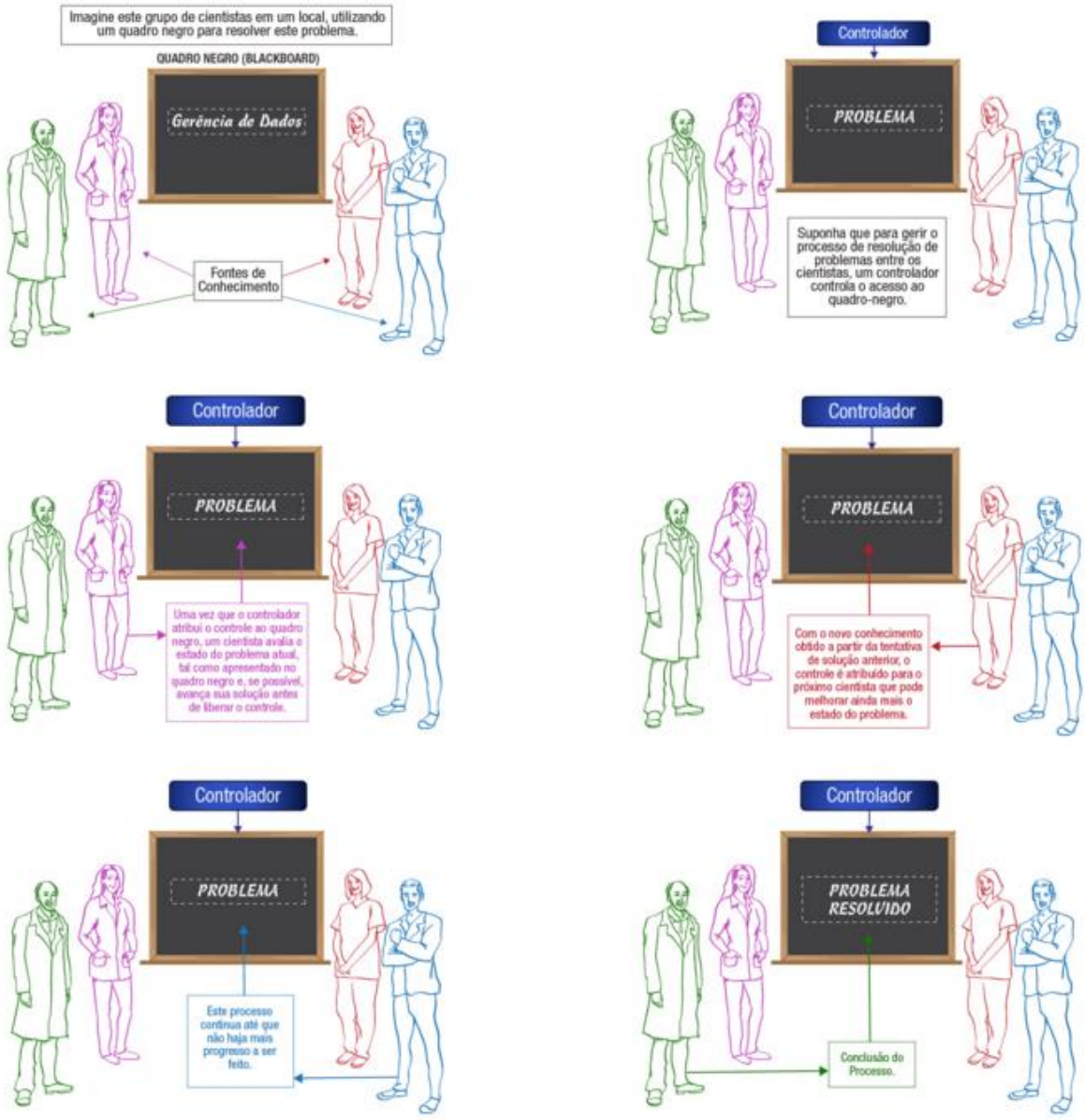
Exemplos de sistemas centrados nos dados incluem sistemas especialistas, que interagem com um sistema de gerenciamento de banco de dados para armazenar e recuperar informações de base de conhecimento.

Um exemplo de um padrão de arquitetura para sistemas centrados nos dados inclui o padrão de arquitetura quadro-negro (Blackboard Pattern), que será apresentado a seguir.

03

1.1- Blackboard Pattern – arquitetura quadro-negro

Para entendermos o funcionamento do padrão de arquitetura quadro-negro vamos imaginar um grupo de cientistas tentando resolver um problema complexo.



04

De posse deste entendimento, podemos definir que o padrão de arquitetura quadro negro se decompõe em componentes de sistemas de *software* que funcionam em torno de um componente central de dados, o quadro negro, para fornecer soluções para problemas complexos. Estes componentes funcionam independentemente uns dos outros para fornecer soluções para problemas. Não há uma sequência de operações pré-determinada ou correta para alcançar a solução do problema. O acesso ao componente central de dados pode ser feito por meio de uma referência direta à memória ou consultas ao banco de dados.

Esse estilo faz uso de um repositório central de dados circundado por um conjunto de componentes (ou células) de informações. Esses componentes contêm informações necessárias à solução de problemas. Os dados da solução de problemas são mantidos na base de dados compartilhada.

A figura abaixo demonstra como este padrão é organizado em termos de componentes.

05

As principais vantagens da utilização do padrão de arquitetura quadro negro incluem sua mutabilidade, reutilização e facilidade de manutenção. Estas propriedades de qualidade vêm como resultado de compartimentar fontes de conhecimento e estabelecer um método normalizado para a utilização do repositório central. Uma vez que cada agente precisa saber só a forma de comunicar com o quadro negro, novos agentes podem ser introduzidos sem muito esforço para melhorar as capacidades do sistema.

Além disso, por compartimentar fontes de conhecimento, mudanças no sistema são também compartimentadas. Portanto, as mudanças em um agente não afetam outros agentes do sistema. Finalmente, compartimentalização de agentes suporta fácil reutilização em futuros sistemas.

As propriedades de qualidade do padrão de arquitetura quadro-negro são apresentadas a seguir:

- **Mutabilidade**

Os agentes são compartimentados e independentes uns dos outros. Assim, é fácil de adicionar ou remover os agentes para ajustar os novos sistemas.

- **Reutilização**

Componentes especializados podem ser reutilizado facilmente em outras aplicações.

- **Facilidade de manutenção**

Devido à independência dos agentes, a manutenção de componentes existentes torna-se mais fácil.

06

2 - Sistema de Fluxo de Dados

Um sistema de fluxo de dados tem como premissa o transporte e a transformação de dados para atender aos requisitos específicos de um sistema. Desta forma, este tipo de sistema pode ser decomposto em

- **componentes de trabalho** e
- **componentes de transportes**.

Como o próprio nome sugere, os componentes de trabalho são responsáveis por realizar as transformações dos dados que necessitam ocorrer antes que estes dados possam ser transportados para os outros compoentes.

Exemplos de trabalhos ocorridos nestes tipos de componentes são: criptografia, descriptografia, compressão e descompressão.

Os componentes de transporte, por sua vez, são responsáveis por realizar a gestão e controle dos mecanismos de transporte de dados, que podem incluir a comunicação entre processos, comunicação baseada em soquete e interfaces seriais.

Os componentes de trabalho e os componentes de transporte se combinam para formar elementos arquitetônicos dos sistemas de fluxo de dados. Sistemas de fluxo de dados fornecem os meios para a transformação de dados, que ocorrem em série ou em paralelo, o que ajuda a melhorar o desempenho do sistema através da adição de simultaneidade para o sistema. Um exemplo de um padrão de

arquitetura para sistemas de fluxo de dados é o padrão de arquitetura Pipe e Filter, que veremos a seguir.

07

2.1- Pipe e Filter Pattern

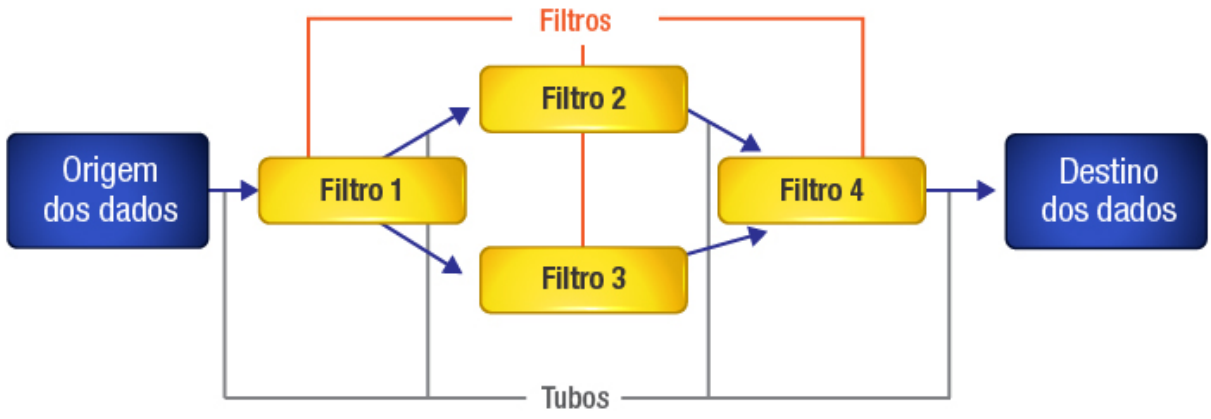
O padrão de arquitetura Pipe (tubo) e Filter (filtro) se decompõe em componentes que realizam duas funções principais:

- **processamento e transformação de dados e**
- **transferência de dados entre componentes.**

Componentes responsáveis pelo processamento de dados e de transformação são referidos como **filtros**, enquanto componentes que transferência de dados entre componentes são referidos como **tubos**.

Em conjunto, estes componentes são combinados de várias formas para criar uma família de sistemas relacionados que processam correntes de dados. Os tubos e filtros como padrão de arquitetura são comumente vistos em sistemas de fluxo de dados, onde as entradas de dados precisam ser transformadas em saída de dados através de uma série de componentes de cálculo ou manipuladores.

A estrutura de Pipe e Filter está presentada na figura abaixo.



As principais características que constituem as **vantagens** do padrão arquitetural Pipe (tubo) e Filter (filtro) são apresentadas a seguir:

- **Extensibilidade**

Filtros de processamento podem ser adicionados facilmente para proporcionar mais capacidades.

- **Eficiência**

Ao ligar os filtros em paralelo, simultaneidade pode ser alcançada para reduzir a latência no sistema.

- **Reutilização**

Por compartimentar tubos e filtros, eles podem ser reutilizados em outros sistemas.

- **Modificabilidade**

Os filtros são compartimentados e independentes um do outro; por conseguinte, é fácil de adicionar ou remover filtros para melhorar o sistema.

- **Segurança**

Em qualquer ponto durante o fluxo de dados, os componentes de segurança podem ser injectados para o fluxo de trabalho para proporcionar diferentes tipos de mecanismos de segurança para os dados.

- **Manutenibilidade**

Permite a separação de preocupações e independência dos filtros e tubos; por conseguinte, a manutenção de componentes existentes torna-se mais fácil.

08

3 - Sistema Distribuído

Os sistemas distribuídos são vulgarmente conhecidos como **sistemas decompostos** em vários processos que colaboram através da rede. Este padrão é amplamente difundido atualmente graças a tecnologias como: internet móvel e sem fio.

Em alguns sistemas distribuídos, um ou mais processos distribuídos executam o trabalho em nome de usuários do cliente e fornece uma ponte para algum computador servidor, geralmente localizado remotamente e executam o trabalho delegado a ele por parte do cliente. Uma vez concluído, os resultados são normalmente retornados para os clientes para visualização e processamento posterior.

Outros sistemas distribuídos podem ser compostos por nós com capacidades semelhantes e colaborando entre si para fornecer serviços avançados. Estas formas de sistemas distribuídos são bem conhecidas, no sentido de que a sua arquitetura de implantação envolve, tipicamente, múltiplos nós. No entanto, com o advento de múltiplas arquiteturas de CPU, arquiteturas distribuídas também são relevantes para *software* que é executado em um único nó com capacidade de multiprocessador.

As principais preocupações para sistemas distribuídos podem incluir:

- desempenho,
- confiabilidade,
- disponibilidade,
- segurança e
- interoperabilidade.

Padrões arquitetônicos comuns para estes sistemas incluem: Cliente Servidor e Broker.

09

3.1- Padrão Cliente-Servidor

A arquitetura cliente-servidor é um padrão popular, presente na arquitetura de sistemas modernos de hoje. Ele decompõe sistemas de *software* em dois componentes principais: o **cliente** e o **servidor**. Esses componentes são manifestados como processos individuais que podem ser distribuídos através da rede ou dentro de um único nó.

Sistemas cliente-servidor não são determinados apenas por processos de separação ou através da distribuição de processos em toda a rede, mas por ter um processo, o cliente, dependente dos serviços prestados por outro processo, o servidor.

O exemplo mais difundido de um sistema cliente-servidor hoje inclui o cliente navegador e o servidor web. Ao procurar um determinado site usando o navegador web, uma conexão é feita para o servidor, solicitações são enviadas e recebidas e o servidor processa as solicitações e envia as respostas ao cliente. Note que isto é verdadeiro independentemente do local onde se encontra o cliente, podendo inclusive estar no mesmo nó que o servidor. *O importante é que este cliente* possa se conectar ao servidor. A figura a seguir apresenta o padrão de arquitetura cliente-servidor.

10

Sistemas cliente-servidor são particularmente úteis para sistemas distribuídos a uma grande base de clientes, uma vez que fornecem a localização de dados em um lugar central. Portanto, fazer atualizações ou adicionar novas informações num local centralizado é o suficiente para uma multidão de clientes receber tais informações. As vantagens associadas aos sistemas cliente-servidor são identificadas a seguir:

- **Interoperabilidade**

Permite que os clientes em diferentes plataformas possam se comunicar com servidores de diferentes plataformas.

- **Modificabilidade**

Permite mudanças centralizadas no servidor e distribuição rápida entre muitos clientes.

- **Disponibilidade**

Ao separar os dados do servidor, vários nós de servidor podem ser conectados como cópia de segurança para aumentar os dados do servidor ou a disponibilidade dos serviços.

- **Reutilização**

Ao separar servidor de clientes, serviços ou dados fornecidos pelo servidor podem ser reutilizados em diferentes aplicações.

11

3.2- Padrão Broker

O padrão de arquitetura Broker fornece mecanismos para alcançar uma melhor flexibilidade entre clientes e servidores em um ambiente distribuído. No padrão de arquitetura cliente-servidor, os clientes acessam diretamente os serviços de servidores, o que pode obrigá-los a estabelecer uma conexão direta ou utilizar outros mecanismos de comunicação entre processos para se comunicar com o servidor.

Isso resulta em um maior grau de acoplamento entre clientes e servidores, o que leva à complexidade dos sistemas esperados para evoluir, fornecendo serviços a partir de diferentes servidores hospedados em locais diferentes. Em alguns casos, terminais de cliente precisam ser capazes de acessar os serviços de vários servidores sem conhecer as suas localizações reais ou detalhes particulares de comunicação para ter acesso a esses serviços. Isso leva a sistemas com maior interoperabilidade e flexibilidade.

O padrão de arquitetura Broker tem como objetivo diminuir a dependência entre os clientes e os servidores para que um cliente possa acessar de maneira transparente os serviços de diferentes servidores.

Em vez de acessar diretamente um servidor, os clientes acessam suas funcionalidades através de um componente do broker, que localiza servidores apropriados, encaminha as solicitações, e transmite respostas (incluindo exceções) de volta para clientes. Com este mecanismo, os clientes podem solicitar serviços como se eles fossem fornecidos localmente no mesmo nó que o servidor, quando estão de fato em nós diferentes.

12

Os principais participantes do padrão arquitetônico Broker são apresentados a seguir:

Cliente	As aplicações que utilizam os serviços prestados por um ou mais servidores.
Clientproxy	Componente que fornece transparência (pelo cliente) entre os componentes remotos e locais para que os componentes remotos apareçam como locais.
Corretor	Componente que faz a mediação entre os componentes de cliente e servidor.
ServerProxy	Componente que fornece transparência (no servidor) entre os componentes remotos e locais para que os componentes remotos apareçam como locais.
Servidor	Prestação de serviços aos clientes; também pode atuar como cliente para o corretor.
Ponte	Componente opcional para encapsular a interoperação entre os corretores

12

As **vantagens** deste padrão arquitetural estão listadas a seguir:

- **Interoperabilidade**

Permite que os clientes em diferentes plataformas para interoperar com servidores de diferentes plataformas; também permite que os clientes para interoperar (transparente) com vários servidores

- **Modificabilidade**

Permite mudanças centralizadas no servidor e distribuição rápida entre muitos clientes

- **Portabilidade**

Por portar o corretor para diferentes plataformas, os serviços prestados pelo sistema podem ser facilmente adquiridas por novos clientes em diferentes plataformas

- **Reutilização**

Corretores muitas chamadas do sistema abstratos necessários para a prestação de comunicação entre os nós; ao usar corretores, muitos serviços complexos podem ser reutilizados em outras aplicações que requerem operações distribuídas semelhantes

13

4 - SISTEMAS INTERATIVOS

Sistemas interativos são sistemas que suportam interações do usuário, geralmente por meio de interfaces de usuário.

Ao projetar sistemas interativos, alternativas de projeto concentram em dois atributos de qualidade principais: **usabilidade** e **modificabilidade**.

Usabilidade refere-se à meta de qualidade que visa minimizar o grau de complexidade envolvido no aprendizado e na utilização do sistema. Sistemas são projetados de tal forma que os usuários possam rapidamente tornar-se proficientes no sistema; eles também respondem a solicitações de usuários rapidamente para suportar os requisitos de alta interatividade.

Para maximizar a modificabilidade em sistemas interativos, a interface gráfica de usuário deve ser desacoplada do núcleo do sistema funcional. Ao fazer isso, o núcleo-funcional se torna mais estável uma

vez que a interface de usuário é mais propensa à mudança. O padrão de arquitetura iterativo é o MVC – Model-View-Controller, que será apresentado a seguir.

14

4.1- Padrão Modelo-Visão-Controlador (Model-View-Controller)

O padrão de arquitetura MVC é usado em aplicações interativas que exigem flexibilidade na interface do sistema. Com o MVC, os sistemas são decompostos em três componentes principais que lidam de forma independente:

- componente de entrada,
- processamento e
- saída do sistema.

Ao separar a saída do sistema de suas funções de processamento centrais, diferentes representações do núcleo do sistema podem ser facilmente suportadas.

Os principais **componentes** do padrão de arquitetura MVC são:

- **Modelo;**
- **Visão;**
- **Controlador.**

Os compoentes visão e controlador trabalham em conjunto como parte da interface do usuário para aceitar a entrada do usuário e transformar essa entrada para um formato compatível com o componente do modelo. Em algumas variantes do MVC, a responsabilidade dos controladores e visões é fundida em um componente.

Modelo

Componente que representa o núcleo dos sistemas.

Visão

Componente que representa a apresentção do sistema. A interface do usuário.

Controlador

Componente (associado com uma visão) que lida com entradas do usuário

15

A relação entre os componentes modelo, visão e controlador pode variar dependendo da aplicação; no entanto, no mínimo, projetos MVC fornecem relações que permitem mudanças no modelo para adequar-se ao seu ponto de vista e, quando necessário, aos controladores. Desta forma, os sistemas MVC fornecem uma abordagem sistemática, flexível e controlada por aceitar entradas e proporcionar saídas do sistema.

Como um padrão de arquitetura, MVC define as interfaces necessárias para os mecanismos de propagação entre os componentes modelo, visão e controlador. No entanto, não são especificados os detalhes de como o mecanismo de propagação de mudança é de fato implementado. Os pormenores de tais mecanismos são deixados a cargo do projeto.

16

5 - Sistemas Hierárquicos

Sistemas hierárquicos são sistemas nos quais componentes podem ser estruturados de forma hierárquica, de modo que os componentes existem em diferentes níveis de abstração e cada nível aborda uma preocupação particular do sistema de *software*.

Conceitualmente, os componentes que residem em níveis mais altos da hierarquia encaminham suas requisições para os componentes de níveis abaixo. Em alguns casos, o acesso aos serviços prestados nos diferentes níveis podem ser unificados, permitindo compartimentar e aumentar a capacidade de reutilização dos seus serviços. Em outros casos, a estrutura hierárquica é mapeada para o tratamento de dados, resultando em um conjunto de componentes funcionais coesos em níveis adequados de abstração para a criação de sistemas modulares.

Em qualquer caso, a concepção de sistemas em forma hierárquica normalmente leva a sistemas bem estruturados e modulares. Dois padrões arquitetônicos comuns para os sistemas hierárquicos são:

- Programa principal e sub-rotina;
- Camadas.

Ambos os padrões serão apresentados a seguir.

5.1- Programa principal e sub-rotinas

O padrão arquitetônico de programa principal e sub-rotina é popular em sistemas que são projetados usando a estratégia de design estruturado (ou funcional).

Nestes sistemas, um componente principal contém os principais dados para o programa, que é compartilhado entre os componentes que residem em níveis mais baixos da hierarquia. Cada nível da hierarquia representa refinamentos do sistema, de modo que o nível n fornece o nível principal; nível n + 1 fornece refinamentos de serviços; n + 2 fornece mesmo refinamentos, e assim por diante. Este processo continua até que o sistema seja decomposto em um conjunto apropriado de componentes ou sub-rotinas.

Os principais componentes identificados no padrão de arquitetura de programa principal e sub-rotina incluem um **componente principal**, que armazena todos os dados, e vários **subcomponentes** que realizam operações detalhadas do sistema.

Os sistemas baseados nesse padrão de arquitetura têm como principal benefício sua decomposição estruturada. Esta decomposição proporciona componentes independentes e com uma única finalidade, o que torna mais fácil de entender, gerenciar, depurar e reutilizar.

Os principais **benefícios** associados ao padrão de arquitetura de programa principal e sub-rotina principal são:

- **Modificabilidade.**
- **Reusabilidade.**

Modificabilidade

Ao se decompor o sistema em componentes de finalidade única independentes, cada componente torna-se mais fácil de entender e gerenciar.

Reusabilidade

Componentes independentes podem ser reutilizados em outros sistemas.

18

5.2- Padrão de Camadas

O padrão de arquitetura de programa principal e sub-rotina conduz a estruturas hierárquicas que se expandem verticalmente e horizontalmente, com cada nível de hierarquia contendo um ou mais componentes que podem interagir com um ou mais componentes com níveis mais baixos da hierarquia.

Uma forma mais restrita da estrutura hierárquica envolve que cada **camada** pode ter um componente principal, que pode ser composto internamente de vários componentes, que fornece uma interface unificada para comunicação com componentes que residem imediatamente abaixo na estrutura de hierarquia. Esta forma de colaboração restrita é definida como **arquitetura em camadas**.

Com o padrão de arquitetura em camadas, o trabalho realizado para implementar uma função do sistema é mais compartimentada do que no programa principal e sub-rotina. Ele é usado quando os sistema pode ser decomposto em camadas coesas com uma forma estruturada de interface entre as camadas.

A arquitectura em camadas é amplamente utilizada em sistemas como a pilha de comunicação de um sistema operacional, onde cada camada é uma abstração das principais funções de comunicação do sistema operacional. Cada camada também depende do serviços de outras camadas diretamente abaixo para criar pacotes de comunicação e para proporcionar qualidade de serviço, serviços de roteamento, comunicação ponto a ponto e transmissão usando as diferentes camadas físicas. Desta forma, as regras podem ser impostas a arquitetura lógica do sistema para restringir o acesso entre os componentes, diminuindo o acoplamento e aumentando a sua manutenibilidade e portabilidade.

19

As principais **vantagens** do padrão em camadas são:

- **Modificabilidade**

As dependências são mantidas localmente dentro de componentes de uma mesma camada. Como os componentes podem acessar outros componentes somente através de uma interface bem definida e unificada, o sistema pode ser facilmente modificado, trocando componentes da camada por outros componentes melhorados.

- **Portabilidade**

Serviços que lidam diretamente com uma interface de programação de aplicativo (API) podem ser encapsulados usando um componente de camada de sistema. Camadas de nível superior contam com

este componente para a prestação de serviços do sistema para a aplicação. Portanto, por dispor de uma camada de API para outras plataformas, os sistemas se tornam mais portáteis.

- **Segurança**

A estrutura hierárquica controlada de sistemas em camadas permite a fácil incorporação de componentes de segurança para criptografar ou descriptografar os dados de entrada ou saída.

- **Reutilização**

Ao compartimentar os serviços de cada camada, torna-se mais fácil reutilizar.

20

RESUMO

Este módulo estudamos os padrões de arquitetura Centrados nos Dados, Fluxo de Dados, Sistema Distribuído, Sistemas interativos e Sistemas Hierárquicos.

Para atingir este objetivo, iniciamos entendendo o que são estilos arquitetônicos e padrões arquitetônicos. Os estilos arquitetônicos fornecem soluções arquitetônicas, genéricos e reutilizáveis de uma forma que possa ser facilmente compreendida e aplicada a novos problemas que exigem características arquitetônicas semelhantes. Já os padrão descreve uma solução para um problema que ocorre com frequência durante o desenvolvimento de software, podendo ser considerado como um par "problema/solução".

Nos **Sistemas centrados nos dados** vimos que estes sistemas são decompostos principalmente em torno de repositório central de dados. Portanto, as responsabilidades típicas encontradas nos componentes do sistema centrado em dados incluem um gerenciador de dados centralizado e vários componentes de trabalho. Como exemplos deste tipo de padrão foi apresentado o padrão quadro-negro. Neste padrão, componentes de sistemas funcionam em torno de um componente central de dados, o quadro negro, para fornecer soluções para problemas complexos.

Nos **Sistema de Fluxo de Dados** vimos que tem como premissa o transporte e a transformação de dados para atender aos requisitos específicos de um sistema. Desta forma, ete tipo de sistema pode ser decomposto em componentes de trabalho e nos componentes de transportes. Como exemplos deste tipo de padrão foi apresentado o padrão Filtro e tupo. O padrão de arquitetura Pipe (tubo) e Filter (filtro) se decompõe em componentes que realizam duas funções principais: processamento e transformação de dados e transferência de dados entre componentes.

21

Os **Sistemas Distribuídos** são conhecidos como sistemas decompostos em vários processos que colaboram através da rede. Este padrão é amplamente difundido atualmente graças a tecnologias como: internet móvel e sem fio. Neste padrão estudamos o Padrão Cliente-Servidor, que decompõe sistemas de software em dois componentes principais: o cliente e o servidor. Esses componentes são manifestados como processos individuais que podem ser distribuídos através da rede ou dentro de um único nó (computador). Vimos também o Padrão Broker, que fornece mecanismos para alcançar uma melhor flexibilidade entre clientes e servidores em um ambiente distribuído.

Os **Sistemas interativos** são sistemas que suportam interações do usuário, geralmente por meio de interfaces de usuário. Neste padrão estudamos o Padrão Modelo-Visão-Controlador (Model-View-Controller). O MVC é usado em aplicações interativas que exigem flexibilidade na interface do sistema. Com o MVC, os sistemas são decompostos em três componentes principais que lidam de forma independente: componente de entrada, processamento e saída do sistema.

Os **Sistemas hierárquicos** são sistemas nos quais componentes podem ser estruturados de forma hierárquica, de modo que os componentes existem em diferentes níveis de abstração e cada nível aborda uma preocupação particular do sistema de software. Estudamos aqui o padrão arquitetônico de programa principal e sub-rotina muito popular em sistemas que são projetados usando a estratégia de design estruturado (ou funcional). Nestes sistemas, um componente principal contém os principais dados para o programa, que é compartilhado entre os componentes que residem em níveis mais baixos da hierarquia.

Ainda em sistemas hierárquicos, estudamos, por fim, a arquitetura em camadas onde uma forma mais restrita da estrutura hierárquica envolve que cada camada pode ter um componente principal, que pode ser composto internamente de vários componentes, que fornece uma interface unificada para comunicação com componentes que residem imediatamente abaixo na estrutura de hierarquia.