Componentização (CBD) e Arquitetura

Descritivo Conceitual

Sumário

1. Component Based Development

1.1. O que são componentes ?

Diferente da orientação a objetos, onde a teoria foi apresentada formalmente, a componentização não obteve uma fundamentação teórica semelhante. Sendo assim, na prática temos várias abordagens do que realmente é um componente.

A Componentização é um processo proposto por engenheiros de software, não possui uma raiz acadêmica como a orientação a objetos. Sua força está no resultado prático da aplicação de suas técnicas e no resultado superior à orientação a objetos. Principalmente no âmbito gerencial, onde o ROI fala mais alto. O purismo é facilmente abandonado quando temos um mesmo produto com mais qualidade e de menor preço.

Por outro lado, não havia razões comerciais que justificassem as grandes empresas mundiais a enxergarem a componentização como algo diferente da orientação a objetos. Sendo assim, um componente é um objeto grande e ponto final.

Contudo, uma grande comunidade vem apoiando firmemente o denominando Component Based Development - CBD. Talvez para a surpresa de alguns, um software produzido sob a visão ou paradigma de componentes não precisa se utilizar do paradigma de orientação a objetos. Ou seja, componentização não estende orientação a objetos. Na verdade, podemos dizer que componentização e orientação a objetos são teorias independentes entre si.

No entanto, embora independentes, são complementares. Construir um componente utilizando orientação a objetos é a melhor das opções.

Infelizmente nossa definição de software terá que sofrer grandes modificações se quisermos definir um componente de forma adequada. Se quisermos realmente entrar no mundo da componentização, o software, em sua forma tradicional, ou seja, sendo visto como um programa que gera um executável, está com seus dias contados.

Iniciaremos pela redefinição do que é um software e na mesma seqüência apresentaremos nossa definição de componentes. Cada **componente** possui um conjunto de **artefatos** gerados a partir de um processo de desenvolvimento e tendo como objetivo alvo um framework em particular.

> **Software**: Conjunto de **componentes** gerados na solução de um problema computacional. Cada **componente** possui um conjunto de **artefatos** gerados a partir de um processo de desenvolvimento e tendo como objetivo alvo um framework em particular.

Se existe um centro em um software, na componentização o centro é o componente. Assim não existe o conceito de programa, muito menos de executável. Um componente faz com que gravite ao seu redor toda a documentação necessária, todos os artefatos, todos os níveis de abstração; passa pela implementação e finalmente chega à sua distribuição (deployment) em um ambiente de execução de componente (AEC).

O conceito-chave da componentização é a distribuição. Ou seja, um componente teoricamente não possui dependência interna de outra entidade qualquer. Isso não significa que externamente, para seu funcionamento, ele não necessite de outros componentes, contudo ele é independente e autônomo.

Outra característica fundamental é a centralização do processo de desenvolvimento em componentes e não em artefatos gerais. Isso significa que um componente possui várias dimensões, ou seja, vários aspectos, e que todo o processo de desenvolvimento serve apenas para refiná-lo até sua distribuição final em um AEC. Além disso, os artefatos gravitam em torno dos componentes, e não em torno do software como um todo.

O conceito de componente, verdadeiramente, é recursivo. Ou seja, nada impede que um componente incorpore outro componente em sua definição. Nesse sentido, um software nada mais é do que um componente; seu conceito passou a ser relativo e dependente do contexto.

Assim, podemos dizer que um componente é um software em miniatura. Se esse conceito está difícil para o leitor, basta direcionar sua percepção para as outras Engenharias, como a Civil ou Eletrônica por exemplo. Um transistor é um exemplo de um componente eletrônico. Uma Porta ou uma Janela também é um exemplo de um componente. Uma casa pode ser vista como um grande componente.

1.2. Definindo um componente

Um componente engloba todos os aspectos de um software. Assim, podemos falar de modelagem de componente, arquitetura de componente, padrões de componente, etc. Portanto, de forma geral, um componente é um conceito típico de Engenharia de Software que visa fundamentalmente a garantir o reuso e diminuir os custos, além de aumentar a qualidade dos artefatos gerados.

Contudo, paralelamente a esta visão geral de um componente, existe uma visão realmente de especificação ou implementação, a qual iremos nos ater agora.

> Um **Componente de Software** é uma unidade encapsulada de código binário produzida especificamente para um framework, e cujos serviços são externados via interfaces denominadas Provided.
>
> Um **Componente de Software** pode ser substituído a qualquer tempo, desde de que o seu substituto ofereça no mínimo os mesmos serviços do componente original e seja binariamente compatível com o mesmo framework.
>
> Um **Componente de Software** poderá depender de outros componentes. Neste caso, a dependência será sinalizada via interfaces denominadas Required.
>
> A construção de um **Componente de Software** deverá obedecer a todas as fases de desenvolvimento. Portanto, será gerado não só o modelo de especificação, como também todos os artefatos necessários ao seu uso correto, manutenção e instalação.

Obviamente poderemos fazer uso da proposta MDA, visando especificar um componente abstrato (PIM) e implementa-lo em várias arquiteturas alvos (PSM). No entanto, estritamente falando, cada componente PIM dará origem a vários componentes diferentes. Pois em última análise o que vale é o framework alvo, e não a especificação. No entanto, iremos conciliar estas visões em um todo coeso em nosso capítulo de Análise de Negócio.

1.3. Compromissos da componentização

Como já dito anteriormente, a Componentização é fundamentalmente defendida por Engenheiros de Software e não por Cientistas da Computação. Portanto, existem alguns compromissos práticos aos quais a Componentização espera atender, são eles:

- **Reuso** – A principal motivação da componentização. Reuso aqui não está ligado à codificação, e sim a uma visão industrial. Ou seja, reuso de todas as fases e processos de desenvolvimento de um novo componente, bem como dos respectivos "segredos" industriais de sua produção.
- **Latência zero** – Resposta rápida a mudanças do negócio.
- **Adaptabilidade** – O software realmente veste o negócio, se tornando único e específico.
- **Qualidade** – Reinventar a roda significa abrir as portas para os erros. Portanto, reuso de componentes aumenta consideravelmente a qualidade do produto final.
- **Diminuição de Custos** – Obviamente o reuso de componente afeta diretamente a diminuição de custos, junto com a diminuição do tempo de desenvolvimento.
- **Escalabilidade** – Construções efetivas de Engenharia são possíveis, pois a complexidade está gerenciada internamente nos componentes.
- **Integração** – Possibilidade ilimitada de integrar legado com novas tecnologias via Enterprise Application Integration–EAI.

Embora existam outros fatores de menor ordem ou valor, os sete itens acima representam os aspectos que os defensores do chamado CBD acreditam poder alcançar. É importante observar que estes mesmos defensores do CBD acreditam firmemente que a orientação a objetos não conseguiu nos oferecer o devido conforto na solução destes problemas.

1.4. Aspectos de um Componente

Aspecto ou dimensão é um conceito extremamente poderoso e necessário no contexto atual da Engenharia de Software. A produção de software em camadas representa uma visão inicial ou básica do que seria um software construído utilizando-se do conceito de aspectos. Iremos apresentar agora várias subdivisões possíveis na exposição de um componente. Todas elas representam na verdade aspectos presentes na construção de qualquer componente.

Poderemos subdividir um componente utilizando-nos da visão de artefatos, assim teríamos:

1. Componente de Especificação.
2. Componente de Implementação.
3. Componente de Distribuição.

Quando utilizarmos o termo componente de especificação, estamos nos referindo aos modelos e regras que definem a especificação de um componente. Um componente de implementação refere-se ao código-fonte que implementa um determinado componente. E, por fim, componente de distribuição refere-se ao código binário de um determinado componente. Uma outra subdivisão possível para um componente está relacionada à sua proximidade ou nível de abstração em relação ao negócio. Assim, teremos:

1. Componente de Negócio.
2. Componente de Sistema.
3. Componente de Infra-Estrutura .

Os componentes de negócio expressam naturalmente as necessidades modeladas durante o estudo dos processos da empresa. Os componentes de negócio ainda são divididos em componentes de entidade e componentes de processo.

Os componentes de sistema representam os componentes conhecidos e presentes nas paletas das linguagens de programação.

Os componentes de infra-estrutura representam os componentes presentes em um framework aos quais todos os outros componentes devem se submeter; por exemplo: forma de se fazer persistência de objetos, forma de interfacear com páginas web (JSP), forma de fazer distribuição (EJB), em suma, regras impostas pelo framework e pelo CEE que o implementa.

Uma outra forma de subdividir um componente está fundamentada em seu local de execução. Usaremos aqui o termo direto em inglês, pois sua tradução nos traria problemas:

1. User Component – Componente presente na interface do usuário.
2. Workspace Component – Componente presente entre a interface do usuário e os componentes que efetivamente rodam no CEE.
3. Enterprise Component – Componente real. Camada onde efetivamente roda o componente.
4. Resource Component – Componente persistente. Na prática um RBDMS, na maioria dos casos.

Veja que a divisão acima com o aspecto de local de execução implica um conceito lógico e independente de tecnologia. Por exemplo, o J2EE denomina Web Container o que acima seria identificado com o aspecto Workspace. Amanhã, quem sabe, poderá não mais existir o conceito de Web nem de Container. Portanto, a divisão em aspecto está mais corretamente apoiada em Engenharia do que as visões proprietárias de diversos fabricantes.

1.5. Component Execution Environment (CEE)

O CEE representa um ambiente tecnológico onde os componentes são executados. Ele é construído através da implementação de um framework em particular.

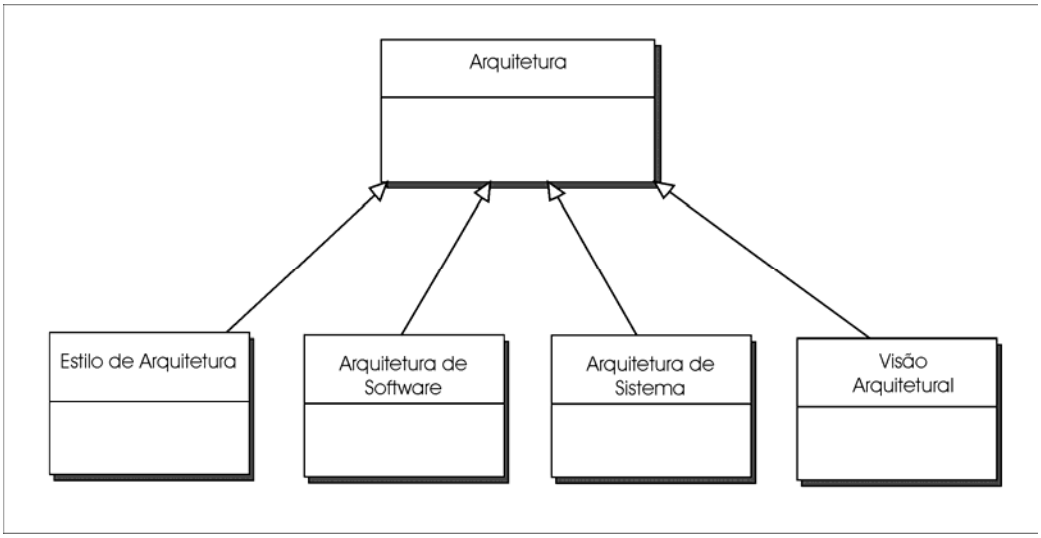
Um CEE contém binariamente a infra-estrutura tecnológica necessária para a execução dos componentes. Iremos daqui em diante utilizar o termo Ambiente de Execução de Componentes (AEC) para referenciar o CEE.

Podemos classificar a tecnologia DCOM da Microsoft, o CORBA da OMG e mais verdadeiramente o padrão EJB e .NET como ambientes de execução de componentes. Como toda engenharia, primeiro vem o produto, depois sua definição teórica. Acreditamos que futuramente, em uma camada acima do sistema operacional, teremos ambientes robustos de execução de componentes. Na verdade, a proposta .NET caminha nessa direção.

2. Arquitetura em Engenharia de Software

Na Engenharia de Software existem quatro usos distintos para o conceito de arquitetura em software, são eles:

1. Estilo de Arquitetura
2. Arquitetura de Software
3. Arquitetura do Sistema
4. Visão Arquitetural



Especialização de Arquitetura

O conceito de arquitetura aqui é semelhante ao conceito de uma classe abstrata chamada animal. Embora, o conceito animal delineie um certo sentimento em nossa mente, enquanto não especializarmos este conceito, para cachorro, por exemplo, nossa conversa ficará no ar.

Portanto, o termo arquitetura sem especialização, não possui nenhum uso prático, e na verdade, não significa muita coisa.

Para que haja um pleno entendimento destes conceitos, iremos apresentá-los em separado, nos próximos tópicos.

2.1. Estilo de Arquitetura

Um estilo arquitetural está intimamente ligado com o conceito de framework e padrões. Este último pode ser omitido, se lembrarmos que o conceito de framework utiliza internamente padrões em sua construção.

Alguns consideram um framework como um sinônimo de estilo arquitetural. Isto porque várias decisões de projeto são realizadas na construção de um framework.

O leitor deve notar, que praticamente partilhamos desta definição, e creditamos ser um framework praticamente um sinônimo de estilo de arquitetura.

Um Componente está estritamente relacionado aos conceitos de composição e distribuição. Está preocupado com vários aspectos de um problema, e pode até mesmo não ser construído utilizado as técnicas de orientação a objetos. Portanto, embora haja grandes semelhanças, existem também diferenças fundamentais.

Por esta mesma razão, um estilo arquitetural é diferente de um framework. Embora, estes conceitos estejam intimamente relacionados como um objeto e um componente, existem também diferenças fundamentais.

O conceito de estilo arquitetural geralmente é referenciado via metáforas com outras engenharias. Por exemplo, o estilo barroco é um exemplo da engenharia civil. Automóveis pickups representam um exemplo de estilo arquitetural da indústria automobilista.

Embora estes exemplos nos ajudem muito a entender teoricamente o conceito, em termos práticos, são de pouca utilidade. Portanto, de forma clara e objetiva: O que é um estilo arquitetural?

> **Estilo Arquitetural**: Representa um conjunto de princípios e práticas de engenharia de software, aplicadas a um *processo de desenvolvimento* em particular, utilizando-se de um conjunto de *ferramentas* em particular visando construir um software através da extensão e personalização de um *framework*.

Nossa definição entrelaça os conceitos de processo de desenvolvimento, ferramentas de engenharia de software e framework.

O framework é o núcleo de um estilo arquitetural. Para um mesmo framework, podemos ter vários estilos arquiteturais. Isto significa que poderemos estender um framework através de vários processos diferentes e várias ferramentas diferentes.

A importância do estilo arquitetural está no uso ótimo de um framework. Obviamente nem todas as ferramentas são úteis na extensão de um framework, e nem todos os processos se adequam corretamente a tarefa de especializar um framework.

Portanto, quando falamos de famílias arquiteturais, subtendem-se famílias de estilos arquiteturais, que compartilham o mesmo framework mais possuem variações em seus processos ou ferramentas.

Voltemos rapidamente para a engenharia civil. Embora exista o estilo barroco, existem várias escolas diferentes com seus próprios processos e ferramentas. Cada escola possui um caminho concreto e prático, de como por exemplo, construir uma igreja barroca.

Um estilo arquitetural não é, portanto, uma abstração. Alguns chamam um estilo abstrato de meta-arquitetura. O estilo barroco definido acima seria uma meta-arquitetura, e cada escola concreta que a implementa seria um estilo arquitetural real ou concreto. Contudo, embora a palavra meta esteja na moda, não iremos partilhar desta visão e discutir estes conceitos teóricos.

Portanto, quando falamos de um estilo arquitetural devemos ter em mente, um framework, um processo e um conjunto de ferramentas e obviamente os princípios e caminhos de como estender o framework utilizando este processo e estas ferramentas.

A especificação do J2EE é simplesmente a definição de um framework. A implementação deste framework, pelo seu suporte à componentização, é reconhecida como um AEC.

A produção de software para o J2EE utilizando por exemplo o processo APD, algumas ferramentas específicas e os princípios personalizados para este framework caracterizam um estilo arquitetural. Obviamente, o framework J2EE poderá ser a origem de várias familiais arquiteturais, se trocarmos o processo ou as ferramentas. No entanto, se não existir uma personalização deste processo e destas ferramentas para este framework, não temos um estilo arquitetural. Pois, estará faltando justamente a cola que permite que o todo conviva harmonicamente.

Uma importante questão que fortalece a necessidade de estilos arquiteturais é o reuso de práticas de sucesso reconhecidas no uso e implementação de softwares. Estamos aqui nos referindo ao reuso não só de modelos e códigos fonte. Estamos falando de reuso arquitetural, no sentido de trilharmos as melhores práticas e técnicas visando atingir com segurança a construção de um software de sucesso.

Na construção de um navio, por exemplo, muito mais que a correta aplicação teórica dos conceitos de engenharia naval são necessários. O mesmo pode-se dizer de um avião em relação a engenharia aeronáutica.

O que estamos dizendo é que o uso de um framework não é suficiente para garantir o sucesso de um software, embora seja praticamente necessário. Um framework pode ser utilizado erroneamente pelos desenvolvedores através de processos e ferramentas inadequadas.

Portanto, um estilo de arquitetura de software representa a velha tabuada presente em outras engenharias, e que ao final dão segurança ao processo. Pois, nenhum engenheiro utiliza somente cálculos diferenciais e integrais para realizar uma construção, quando e se na prática as utiliza. Na verdade, se o fizesse poderia chegar ao valor de 5 unidades, por exemplo. Contudo, se um estilo arquitetural ou simples experiência de outras construções de sucesso do mesmo estilo dissessem que o valor correto seria 4, duvido muito que ele utilizaria o valor 5.

Portanto, um estilo arquitetural está relacionado com a soma sendo maior que as partes. A teoria necessária está embutida na construção do framework e no correto uso do processo e das ferramentas. Contudo, somente a

experiência de grandes maestros do passado nos dará a segurança necessária de forma a garantir que a harmonia do todo possa ser alcançada através da correta regência das partes.

2.2. Arquitetura de Software

Arquitetura de Software está diretamente relacionada com a aplicação dos princípios internos e indiretamente relacionada à aplicação dos princípios externos na construção de um software. Alguns destes princípios estão mais relacionados com arquitetura de software do que outros. Para que o leitor tenha em mente os princípios que estamos nos referindo, transcrevemos para a figura abaixo o resumo dos princípios já estudados em outro capítulo.

Princípios Externos	Princípios Internos
Corretude	Rigorismo
Confiabilidade	Separação de Aspectos
Robustez	Modularidade
Desempenho	Abstração
Facilidade de Uso	Generalidade
Verificabilidade	Antecipação a Mudanças
Facilidade de Manutenção	Refinamento
Reutilização	Classificação
Portabilidade	
Compreensibilidade	
Interoperabilidade	
Produtividade	
Desvio de Planejamento	
Visibilidade	
Segurança	

Princípios de Software

Processo e framework são os locais ideais para garantir um correto atendimento dos princípios de engenharia de software. Jamais um programador irá atentar para estes princípios, pois não há tempo. Portanto, ou o processo e o framework forçam o uso correto destes princípios, ou certamente eles serão atendidos de forma deficiente. Quando digo forço quero dizer de forma implícita e não explicita.

Por exemplo, um processo deverá definir suas técnicas de forma a aplicar corretamente o conceito de abstração, sem a obrigatoriedade de revelar que a técnica se destina a atender este princípio. O que um processo não deve fazer é definir como uma das suas técnicas, as técnicas de abstração. Isto e nada são a mesma coisa. Contudo, embora necessários no conceito tautológico da questão, os processos e as técnicas não são suficientes para garantir uma boa arquitetura de software.

Para que possamos explorar diretamente as principais questões envolvendo uma arquitetura de software vamos defina-la.

> **Arquitetura de Software**: Direcionamento dado à construção de um software visando atender diretamente os princípios internos e indiretamente os princípios externos de engenharia de software.

Assim uma boa arquitetura é sinônimo de um bom atendimento aos princípios internos e externos em um software. Uma arquitetura ruim significa um pobre ou escasso atendimento a estes mesmos princípios.

Vejam que de propósito, estamos utilizando neste tópico a palavra arquitetura como sinônimo de arquitetura de software. Portanto, como já dito, é do contexto que inferimos o uso correto desta palavra.

Neste momento, percebemos que nem todo processo deve ser utilizado em conjunto com um determinado framework. Pois, às vezes, estes princípios se anulam. Ou seja, um processo poderá ser construído ressaltando ou atendendo explicitamente a alguns princípios, e um framework a outros princípios, o resultado final é a anulação da maioria. Portanto, um estilo arquitetural de sucesso é fundamental para garantir a correta junção entre framework, processo e ferramenta.

A arquitetura de software deverá fazer parte das técnicas do processo de desenvolvimento utilizado.

Embora a definição genérica de arquitetura de software seja endereçada a aplicação de todos os princípios de engenharia de software; usualmente o conceito de arquitetura de software está mais relacionado aos seguintes princípios:

1. Princípios Externos

1.1 Facilidade de Manutenção

1.2 Reutilização

2. Princípios Internos

2.1 Modularidade

2.2 Abstração

2.3 Generalidade

2.4 Classificação

Em termos de orientação a objetos e componentização quando falamos em arquitetura de software, geralmente estamos nos referindo a correta divisão do software em subsistemas e componentes com seus relacionamentos.

A divisão de um sistema em subsistemas é fundamental para o sucesso de um software. E geralmente é feita por todos os desenvolvedores de software. Contudo, o baixo valor fornecido a esta tarefa reflete na baixa qualidade do produto final.

Isto porque a subdivisão de um sistema em subsistemas é a pedra angular que permitirá ao software atender um número elevado de nossos princípios. O desprezo usual desta tarefa reflete a inocência de nossa engenharia.

Os softwares usualmente são divididos em subsistemas por questões gerencias ou então utilizando apenas o princípio de classificação. Esta abordagem não é suficiente e metaforicamente é como se estivéssemos em uma guerra, e ao deparar com o inimigo, gritássemos para que os baixinhos fossem para esquerda e os mais altos para direita, ou os que possuem botas para esquerda e os que usam sandálias a direita. O que queremos dizer? Que aqui o comandante do exército não acredita que a correta subdivisão de seu exército é importante, e simplesmente grita para que todos arrumem uma posição.

Já o outro exército, que acredita na arquitetura, manteve a frente os soldados com pontaria a longa distância. A um passo atrás manteve os soldados atiradores de elite de média distância. A um passo atrás destes os soldados com expertise em combate corpo a corpo. Ou seja, organizou seu exército estrategicamente e não simplesmente visando organizar e agrupar soldados.

Se ouve falar que um processo de desenvolvimento deve estar centrado na arquitetura. O que isso quer dizer ?

A idéia central de um processo centrado em arquitetura segundo esta visão está relacionado ao emprego de casos de uso. A pergunta é: Como um caso de uso, um elemento de modelagem como outro qualquer, pode ser considerado o elemento mestre de uma arquitetura?

Neste enfoque em particular, um caso de uso é visto como a linha mestra que guiará todo desenvolvimento de um software. Os casos de uso serão os objetos de agrupamento e darão origem aos subsistemas e componentes do software sendo construído.

Os processos de negócio e as regras oriundas da política do negócio representam um ponto mais seguro para ancorarmos nossa arquitetura de software. Embora ainda iremos defender mais arduamente o porquê desta afirmação, podemos adiantar que um caso de uso está limitado a uma visão sistêmica do negócio.

No entanto, um sistema nasce para atender um processo e é filho deste. Um processo nasce da política do negócio, criado pelos diretores da empresa. Portanto, o ponto de mutação de um software, ou seu centro de gravidade, não está nos casos de uso, e sim nos processos e os porquês dos processos. Somente entendendo estes processos e seus porquês, poderemos aplicar com extremo sucesso o princípio de antecipação a mudanças e vários dentre outros, não menos importantes, princípios em nossa arquitetura de software.

Para finalizar, iremos discutir alguns pontos importantes na definição de uma arquitetura de software restrita, ou seja, endereçada a atacar mais diretamente ao subconjunto dos princípios definidos acima.

O subsistema será definido no tempo apropriado. Contudo, neste instante seria interessante encarar um sistema como um grande quebra cabeça. Um subsistema seria uma destas peças do quebra-cabeça. O problema é que em um quebra-cabeça o todo é criado e o fabricante então divide as partes. Em nosso caso, a situação é um

pouco mais complicada, porque dividimos aquilo que não sabemos o que é, pois ainda está em construção. O nosso problema, obviamente, é chegarmos ao final do jogo percebendo que várias peças não se encaixam com várias outras, e existem vários buracos na figura. Se este for ocaso, teremos o famoso sistema FDB mas conhecido como Flag Based Development. Pois, nestas condições só os flags salvam.

Os subsistemas devem ter entre si um baixo acoplamento, alta coesão dos componentes.

Uma boa arquitetura de software restrita não deverá possuir recursividade entre subsistemas ou componentes. A fim de garantir a facilidade de manutenção do sistema.

A criação de um subsistema de abstração ou núcleo abstrato (kernel) do domínio do negócio também é importante, de forma a isolar os tipos de negócio mais estáveis do sistema..

A correta separação de aspecto deve ser realizada.

2.3. Arquitetura de Sistema

Nossa terceira especialização do conceito de arquitetura é relativamente simples. O termo arquitetura de sistema é utilizado quando nos referimos a toda infra-estrutura necessária para implantação e uso do sistema.

Assim, servidores, roteadores, hubs, impressoras, coletores de dados etc., são inseridos no contexto do software visando dar uma idéia completa e total do ambiente de produção.

A UML possui o diagrama de distribuição visando apresentar estes diagramas junto com os principais componentes de um software. Contudo, na prática, outros softwares mais completos são utilizados para construção de uma arquitetura de sistema.

O diagrama de distribuição deve ser utilizado para representar os aspectos mais importantes da arquitetura e seus relacionamentos. É neste diagrama que são representados os relacionamentos entre os principais nós de processamento do sistema, geralmente servidores e os principais componentes do software.

Vamos então a definição de arquitetura de sistema.

> **Arquitetura de Sistema**: Estudo contendo as informações chave da estrutura física e lógica do parque tecnológico atual, bem como todos os pareceres técnicos envolvendo a correta configuração e atualização do parque visando atender aos requisitos do software em construção.

Uma possível visão da arquitetura de sistema de um software seria:

1. Arquitetura atual
   1.1 Topologia Atual de Rede e seus Servidores
   1.2 Resumo dos principais grupos de trabalho que utilizarão o sistema e seus respectivos parques tecnológicos
2. Arquitetura Proposta
   2.1 Topologia de Rede e seus Servidores
   2.2 Grupos de Trabalho (Clientes do sistema)
   2.3 Camadas (Física e Lógica)
   2.4 Contingência
   2.5 Segurança
   2.6 Upgrades
   2.7 Banco de Dados
3. Ferramentas de Desenvolvimento
   3.1 FrontEnd
   3.2 BackEnd
   3.3 Web Design

A arquitetura do sistema representa uma grande variável de risco. Portanto, é importante que paralelamente ao desenvolvimento de um software, uma equipe esteja dedicada a construção deste ambiente.

2.4. Visão Arquitetural

Finalmente chegamos a última especialização de nosso conceito de arquitetura. Uma visão arquitetural representa uma espécie de resumo executivo do software em construção.

Neste sentido, podemos utilizar o termo arquitetura de negócio, tendo como significado uma visão ou corte executivo das outras arquiteturas definidas acima, tendo como filtro somente os principais artefatos referentes ao domínio do negócio.

O mesmo se aplica a arquitetura de análise. Teríamos uma visão ou corte executivo das outras arquiteturas definidas acima, tendo como filtro somente os principais artefatos referentes à análise de sistema.

Utilizamos a seguinte visão arquitetural:

1. Arquitetura de Negócio
2. Arquitetura de Análise
3. Arquitetura de Projeto
4. Arquitetura de Padrões

Estas visões arquiteturais fornecem um maior benefício ao longo do projeto. Obviamente a arquitetura de sistema definida acima completaria nossa visão arquitetural de um software.

> **Visão Arquitetural**: Visão executiva de um software contendo seus principais artefatos agrupados por uma semântica privilegiada.

A partir da visão arquitetural de um software podemos entender o software como um todo. Podemos dizer que uma visão arquitetural representa uma visão abstrata do sistema contendo aproximadamente 5% de suas principais funcionalidades, as principais soluções utilizadas, e como os principais riscos do sistema estão sendo atacados.

A construção da melhor visão arquitetural de um sistema está a cargo da gerência e coordenação do projeto. Geralmente o próprio processo de desenvolvimento utilizado, como o APD, por exemplo, define seu conceito de visão arquitetural.

Para pensar em uma visão arquitetural imagine, por exemplo, uma possível apresentação a um grupo de investidores. Uma apresentação deste tipo não pode durar mais do que 45 minutos, no máximo. Portanto, nestes 45 minutos você terá que apresentar todo sistema, mesmo os sistemas enormes. Este é o desafio.