PUC Minas
Virtual

Arquitetura de Software Distribuído

Princípios de Arquitetura de Software

SÍNTESE DA DISCIPLINA

Marcelo Werneck Barbosa

Sumário

1. Unidade I – Introdução à Arquitetura de Software

A arquitetura de software pode ser vista de forma análoga à arquitetura na construção civil. Ela é responsável por definir a base da construção do software, fazer seu projeto, pensar em toda a estrutura que precisa ser feita para sustentar o software ao longo de seu projeto e manutenção. O papel responsável por garantir que isso aconteça é o Arquiteto de Software.

O Arquiteto de Software deve ser capaz de:

- reconhecer estruturas comuns em sistemas já desenvolvidos;
- usar o conhecimento sobre arquiteturas existentes para tomar decisões de projeto em novos sistemas;
- realizar uma descrição formal da arquitetura de um sistema a fim de analisar as propriedades do sistema e apresentar a arquitetura para outras pessoas;

São habilidades esperadas de um Arquiteto de Software:

- Compreensão profunda do domínio e das tecnologias pertinentes. O Arquiteto é normalmente um profissional mais experiente. Muitos veem inclusive a carreira de Arquiteto como o final da carreira de um bom desenvolvedor. Como ele avalia possibilidades de solução para os sistemas, ele precisa conhecer as tecnologias disponíveis, suas vantagens e desvantagens;
- Técnicas de modelagem e metodologias de desenvolvimento;
- Entendimento das estratégias de negócios da instituição onde atua para poder fazer propostas de solução alinhadas às demandas e restrições deste negócio;
- Conhecimento de produtos, processos e estratégias de concorrentes;

São atividades típicas do Arquiteto de Software:

- Modelagem de software;
- Análise de trade-offs e viabilidade entre possíveis soluções e tecnologias;
- Prototipação, simulação e realização de experimentos. Em muitos casos, provas de conceito são necessárias para confirmar a viabilidade técnica de alguma solução ou componente;
- Análise de tendências tecnológicas. O Arquiteto precisa sempre conhecer as tecnologias mais recentes;
- Atuação como mentor de arquitetos novatos ou desenvolvedores da equipe;

Um Arquiteto trabalha procurando entender os requisitos que foram levantados, principalmente os requisitos que têm impacto na arquitetura do sistema. Mas precisa compreender de qualquer forma quais são as necessidades do cliente para o produto. O responsável por coletar estas necessidades do cliente é tipicamente um papel chamado de Analista de Requisitos. Este papel pode e muitas vezes precisa trabalhar em conjunto com o Arquiteto na definição dos requisitos, até mesmo para verificar a viabilidade técnica dos mesmos e se eles podem ser feitos dentro das restrições do projeto. Assim, é papel do Analista de Requisitos identificar junto ao cliente e envolvidos as necessidades ou requisitos. Podemos definir um requisito como uma característica que pode ser observada no sistema desenvolvido, algo que o sistema deverá fazer, seja por necessidades do cliente seja por cumprimento a um contrato ou lei, por exemplo.

Os requisitos podem ser classificados, em um sentido, em duas categorias: funcionais e não funcionais. Requisitos funcionais indicam o que o sistema deve fazer, que resultados devem trazer para os usuários. Eles representam as funções executadas pelo sistema para que as necessidades do cliente sejam satisfeitas sem considerar quaisquer restrições. Os requisitos não funcionais indicam as restrições do siste-

ma e os critérios de qualidade para sua aceitação. São exemplos de categorias ou tipos de requisitos não funcionais (as classificações são variadas entre os diversos autores e modelos):

- Usabilidade: Está relacionada com características como capacidade de ser compreendido, aprendido, usado e atrativo para o usuário. Está diretamente ligada ao esforço de uso;
- Confiabilidade: Está relacionada com características como disponibilidade, precisão dos cálculos do sistema e a capacidade de se recuperar de falhas caso ocorram;
- Desempenho: Está relacionada com características como tempo de resposta, tempo de recuperação, taxa de transferência, Disponibilidade e eficiência;
- Suportabilidade: Está relacionado com características como testabilidade, adaptabilidade, manutenibilidade, compatibilidade, configurabilidade, instabilidade, escalabilidade e localização.
- Requisitos de desenho: são também chamados de restrição de desenho, especificam ou restringem o design de um sistema.
- Requisitos de Implementação: especificam ou restringem o código ou a construção do software. Exemplos: Padrões obrigatórios, linguagem de programação, política de integridade de banco de dados, limites de recursos e ambientes operacionais.
- Requisitos de Interface: especificam ou restringem itens externos que o sistema deve interagir, ou restrições em formatos ou outros fatores utilizados dentro de uma interação.
- Requisitos Físicos: especificam ou restringem uma característica física que o sistema deve ter.

Requisitos não funcionais muitas vezes determinam ou compõem a arquitetura de um sistema. Requisitos que influenciam a arquitetura podem ser chamados de requisitos arquiteturais. Requisitos arquiteturais são requisitos que possuem importância para a arquitetura do sistema, tais como requisitos funcionais e requisitos não funcionais (desempenho, usabilidade, confiabilidade, portabilidade, manutenibilidade e reusabilidade). Um requisito pode ser chamado de requisito arquitetural sempre que expressar algum significado para a arquitetura.

O Arquiteto de Software, em suas atividades de modelagem, trabalha com alguns diagramas da UML. Ele pode precisar compreender diagramas da UML tipicamente elaborados por um Analista de Requisitos, como os digramas de casos de uso, classes, atividades ou estado, mas é responsável muitas vezes por elaborar diagramas como os de componentes, pacotes e implantação (também chamado de distribuição). O Arquiteto precisa conhecer os principais elementos destes diagramas. As Figuras 1, 2 e 3 apresentam exemplos respectivamente de diagramas de pacotes, componentes e distribuição.

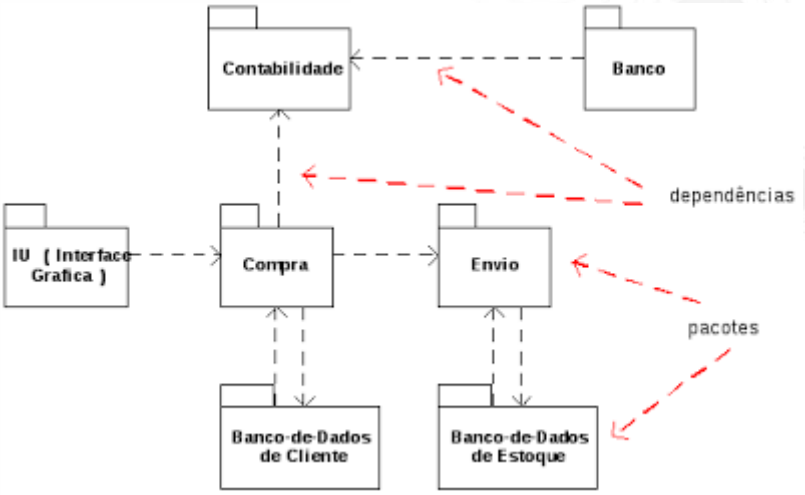

Figura 1. Exemplo de diagrama de pacote

Fonte: http://tecnologialocal.blogspot.com.br/2009/07/uml-pratico-uma-introducao-para.html

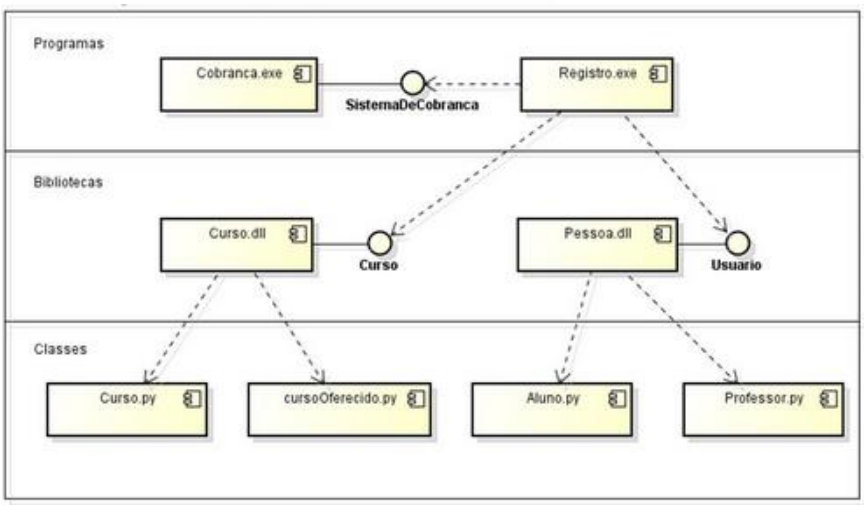

Figura 2. Exemplo de diagrama de componente

Fonte: http://slideplayer.com.br

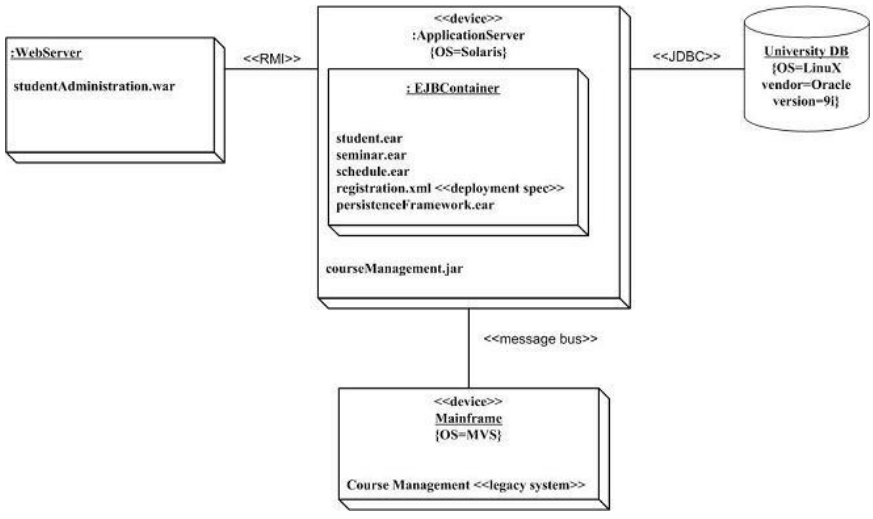

Figura 3. Exemplo de diagrama de implantação

Fonte: http://agilemodeling.com

2. Unidade II – Práticas de Arquitetura de Software

Um dos trabalhos do Arquiteto de Software é produzir o Documento de Arquitetura do projeto. Este documento, muitas vezes, é baseado no modelo de visões proposto por Krutchen, chamado de 4+1. A ideia é que cada visão aborda um determinado conjunto de questões específicas dos envolvidos no processo de desenvolvimento: usuários finais, projetistas, gerentes, entre outros. As visões são: casos de uso, lógica, implementação, distribuição e processos.

A Visão de Casos de Uso contém casos de uso e cenários que abrangem comportamentos significativos em termos de arquitetura, classes ou riscos técnicos. A Visão Lógica contém as classes de design mais importantes e sua organização em pacotes e subsistemas. Contém algumas realizações de caso de uso. É um subconjunto do modelo de design. A Visão de Implementação contém uma visão geral do modelo de implementação e sua organização em termos de módulos em pacotes e camadas. A Visão de Processos contém a descrição das tarefas (processo e threads) envolvidas, suas interações e configurações, e a alocação dos objetos e classes de design em tarefas. Essa visão só precisará ser usada se o sistema tiver um grau significativo de simultaneidade. A Visão de Implantação contém a descrição dos vários nós físicos da maior parte das configurações comuns de plataforma e a alocação das tarefas (da Visão de Processos) nos nós físicos. Essa visão só precisará ser usada se o sistema estiver distribuído.

Os trabalhos do arquiteto de software podem ser utilizados como evidências da realização de boas práticas presentes em modelos de maturidade como o MPS.Br. Um modelo de maturidade de processos de software contem um conjunto de processos que representam boas práticas e estes processos estão organizados em níveis que representam um caminho a ser seguido pelas empresas. Além disso, um modelo de maturidade permite a comparação entre empresas.

O MPS.BR é um programa em desenvolvimento desde 2003 e é coordenado pela Associação para Promoção da Excelência do Software Brasileiro (SOFTEX), e conta com o apoio do Ministério da Ciência e Tecnologia (MCT), da Financiadora de Estudos e Projetos (FINEP) e do Banco Interamericano de Desenvolvimento (BID). O MPS.BR está dividido em três componentes: Modelo de Referência (MR-MPS), Método de Avaliação (MA-MPS) e Modelo de Negócio (MN-MPS). O MPS.BR apresenta níveis de maturidade no qual estabelecem patamares de evolução de processos, caracterizando estágios de melhorias de implementação de processos na organização. O Modelo de Referência para melhoria de processo de software define sete níveis de maturidade:

- A – Em Otimização
- B – Gerenciado Quantitativamente
- C – Definido
- D – Largamente Definido
- E – Parcialmente Definido
- F – Gerenciado
- G – Parcialmente Gerenciado

Os níveis de maturidade do MPS.Br possibilitam um caminho de evolução em etapas menores uma vez que possui mais níveis de maturidade que o modelo CMMI. Essa maior divisão dos níveis também facilita a implantação dos processos e mudança da cultura organizacional.

O propósito do processo Projeto e Construção de Produto é projetar, desenvolver e implementar soluções para atender aos requisitos. A seguir, são discutidos os resultados esperados deste processo.

PCP1 - Alternativas de solução e critérios de seleção são desenvolvidos para atender aos requisitos definidos de produto e componentes de produto.

A definição da arquitetura do software é o primeiro passo a ser tomado durante o projeto (design) de software. Normalmente existem diferentes formas de se resolver um problema. Este resultado espera procu-

ra fazer com que a organização não selecione simplesmente a primeira solução pensada, mas sim a melhor de várias soluções identificadas. Sendo assim, o modelo exige que soluções alternativas sejam identificadas. Além disso, como estamos aqui no nível D, a melhor solução deve ser identificada com base em critérios objetivos, pois se trata de uma organização com processos padrão.

As alternativas de solução devem ser identificadas com base nos requisitos levantados até o momento, em restrições impostas pelo cliente ou em sugestões dadas pela equipe. As alternativas de solução normalmente são definidas através da seleção de componentes arquiteturais e irão estabelecer a arquitetura do sistema. Critérios de seleção destas alternativas devem considerar restrições impostas pelo cliente assim como restrições de custo, de prazo e probabilidade e impacto dos riscos associados com a implantação de cada solução.

Para seleção da melhor alternativa, pode-se apoiar no processo GDE – Gerência de Decisões, que define um método para escolha da melhor solução para um dado problema com base em critérios bem definidos. Pode ser utilizado um método de pontuação (*scoring method*). O resultado também possui associação com o resultado esperado PCP5, que trata de escolhas do tipo comprar, reusar ou fazer internamente. Essa é uma decisão que também impacta a escolha das alternativas de solução uma vez que as diferentes alternativas podem considerar a compra, reuso e produção interna de componentes arquiteturais.

PCP2 - Soluções são selecionadas para o produto ou componentes do produto, com base em cenários definidos e em critérios identificados.

Identificados os critérios de seleção a serem utilizados, é necessário avaliar as soluções alternativas com base nestes critérios e selecionar a solução mais adequada. Para este resultado então, espera-se que seja aplicado o método de seleção de alternativas previsto no resultado anterior. Assim, os critérios indicarão a solução mais adequada para o produto ou para seus componentes. A partir da seleção de uma alternativa, novas decisões podem precisar ser tomadas e por isso a abordagem de seleção de alternativas necessita ser novamente executada. Deve-se lembrar ainda de que o resultado da tomada de decisão deve ser documentado de alguma forma para que possa ser avaliado posteriormente.

PCP3 - O produto e/ou componente do produto é projetado e documentado.

Com o atendimento adequado aos resultados do processo PCP até o momento, foram selecionadas alternativas de solução e as mesmas foram selecionadas com base em critérios definidos. Sendo assim, poderíamos dizer que já se tem uma solução arquitetural definida para o produto e para seus componentes. O próximo passo deve ser então projetar o produto de acordo com os requisitos especificados e de acordo com a solução escolhida.

Entende-se projeto aqui como algo geralmente constituído do projeto da arquitetura do sistema e o do software. A arquitetura definida deve identificar itens de hardware, software e operações manuais. O atendimento a este resultado pode ser evidenciado com a elaboração de documentos técnicos como um Documento de Arquitetura, Documento de Desenho ou de Análise e ainda um Modelo do Banco de Dados.

PCP4 - As interfaces entre os componentes do produto são projetadas com base em critérios predefinidos.

Ao se definir a solução para o produto e seus componentes e ao realizar a documentação e o projeto do sistema, é necessário muitas vezes definir interfaces entre os próprios componentes do produto e entre o sistema e o mundo externo. É considerado um recurso fundamental dos componentes a sua capacidade de definir interfaces.

Um componente precisa expor alguns dos meios para os outros componentes se comunicarem com ele. Assim, uma interface declara o conjunto de serviços que são fornecidos ou exigidos pelo componente.

Existem diversas formas de se definir a interface entre componentes. Componentes podem ser comunicar através de web services, arquivos textos, de maneira síncrona ou assíncrona, por processos batch, etc …

Por ser um processo do nível D, é preciso estabelecer o mecanismo de comunicação com base em critérios bem definidos. Exemplos destes critérios poderiam ser: experiência da equipe, restrições de ambiente, quaisquer restrições técnicas, restrições impostas pelo cliente ou por contrato, arquitetura existente, entre outras. Assim, os critérios devem nortear a seleção do mecanismo de comunicação das interfaces.

PCP5 - Uma análise dos componentes do produto é conduzida para decidir sobre sua construção, compra ou reutilização.

Caso os quatro resultados esperados anteriores tenham sido atingidos, já existe uma solução definida, seu projeto está documentado, assim como o projeto das interfaces de seus componentes. Assim, uma vez identificados os componentes do produto, é necessária agora a existência de uma abordagem que permita a uma organização decidir o que lhe é mais vantajoso:

- desenvolver um determinado componente internamente;
- contratar uma outra organização para fazer este desenvolvimento; ou
- reutilizar um componente já disponível na organização.

Caso a organização possua uma base de ativos reutilizáveis, ela deve consultar esta base para verificar se possui algum componente similar ou que apresente as mesmas capacidades dos componentes necessários para o projeto corrente. Claramente, a análise de fazer internamente, reutilizar ou comprar deve ser feita com base em critérios muito bem definidos. Além disso, o resultado desta análise deve ser documentado.

PCP6 - Os componentes do produto são implementados e verificados de acordo com o que foi projetado.

Uma vez projetado o sistema, suas interfaces e decididos se os componentes serão comprados, reusados ou construídos internamente, aqueles componentes do produto que devem ser construídos, evoluídos ou adaptados são implementados de acordo com o que foi especificado no projeto. Este resultado prevê ainda que a documentação destes componentes deva ser desenvolvida. Além disso, é necessário revisar cada componente do produto, ou seja, os mesmos devem ser verificados. Esta verificação pode acontecer por uma revisão por pares ou testes de unidades, por exemplo. Mais detalhes sobre estas técnicas podem ser vistos no processo VER – Verificação.

Assim, uma vez que as unidades sejam implementadas é necessário verificá-las em relação aos requisitos e ao projeto (*design*). Esta verificação também deve ser feita de acordo com critérios definidos. Tais critérios irão definir, por exemplo, como os componentes devem ser verificados, se o serão em sua totalidade ou apenas parcialmente, o perfil do profissional de verificação ou ainda em que momento devem ser verificados.

PCP7 - A documentação é identificada, desenvolvida e disponibilizada de acordo com os padrões estabelecidos.

Após implementar e verificar os componentes de produto produzidos, deve-se garantir que a documentação dos mesmos seja produzida e disponibilizada para os interessados. Deve-se desenvolver a documentação do projeto e para o usuário final, de acordo com padrões. Este resultado prevê a elaboração de um pacote de dados técnico que descreva o produto. Neste pacote técnico, podem estar todos os artefatos necessários para seu entendimento, evolução e documentação, tais como, documentos técnicos (arquitetura, desenho e análise), scripts, modelos, diagramas e código fonte. A documentação dos componentes do produto permite a realização das tarefas de desenvolvimento ou de manutenção posteriores.

PCP8 - A documentação é mantida de acordo com os critérios definidos.

A documentação necessária para a manutenção, operação e instalação do produto, produzida nos resultados esperados anteriores deve ser mantida e revisada, de acordo com critérios previamente definidos. Estes critérios podem definir, por exemplo, formato da documentação, local ou estrutura de armazenamento, mecanismos de disponibilização, entre outros. O objetivo deste resultado é garantir consistência em relação aos requisitos e ao projeto.

3. Unidade III – Métodos Ágeis

Os modelos prescritivos de processo definem um conjunto distinto de atividades, ações, tarefas, marcos e produtos de trabalho que são necessários para fazer Engenharia de Software com alta qualidade. Os modelos prescritivos de software mais conhecidos são: Modelo em Cascata, Modelos Incrementais, Modelos Evolucionários e, por fim, o Processo Unificado, propondo o Modelo Espiral.

Os modelos ágeis são aqueles que visam tornar menos burocráticos os processos de desenvolvimento e com isso melhorar a interface usuário e software. Os modelos ágeis permitem cada um, a seu modo, atingir pontos importantes do processo de desenvolvimento de sistemas, tornando-os menos expressivos, como custo, complexidade, dificuldade de manutenção e a discordância entre o usuário e o produto solicitado. As metodologias ágeis de desenvolvimento de software se destacam dos processos de desenvolvimento tradicionais devido, principalmente, ao fato de darem prioridade ao desenvolvimento de funcionalidades através de código executável ao invés da produção de extensa documentação escrita, e ainda, respostas rápidas às mudanças e colaboração com o cliente ao invés de seguir planos rígidos e negociações contratuais.

Embora cada envolvido tivesse suas próprias práticas e teorias sobre como conduzir um projeto de software ao sucesso, cada qual com suas particularidades, todos concordavam que, em suas experiências prévias, um pequeno conjunto de princípios sempre parecia ter sido respeitado quando os projetos davam certo. Com base nisso, eles criaram o Manifesto para o Desenvolvimento Ágil de Software, frequentemente chamado apenas de Manifesto Ágil. O termo Desenvolvimento Ágil passou a descrever a abordagem de desenvolvimento que seguiu quatro princípios que passam a valorizar:

- Indivíduos e interação entre eles mais que processos e ferramentas;
- Software em funcionamento mais que documentação abrangente;
- Colaboração com o cliente mais que negociação de contratos;
- Responder a mudanças mais que seguir um plano.

Uma das metodologias ágeis bastante usada atualmente é o Scrum. O Scrum parece uma metodologia simples, mas tem práticas para influenciar profundamente a experiência do trabalho com chave para se obter um processo adaptativo e ágil. O grande diferencial do Scrum entre os métodos é a forte promoção de equipes auto-dirigidas, com avaliações diárias da equipe e prevenção de processo prescritiva. O Scrum é um framework estrutural para desenvolver e manter produtos complexos. Scrum consiste nas equipes do Scrum associadas a papéis, eventos, artefatos e regras. Cada componente dentro do framework serve a um propósito específico e é essencial para o uso e sucesso. O Scrum emprega uma abordagem iterativa e incremental para aperfeiçoar a previsibilidade e o controle de riscos.

O Scrum prevê a existência de alguns papéis, entre eles:

- O Time Scrum é composto pelo *Product Owner* (Gerente do Produto), a Equipe de Desenvolvimento e o *Scrum Master*. Times Scrum são auto-organizáveis e multifuncionais. Equipes auto-organizáveis escolhem qual a melhor forma para completarem seu trabalho, em vez de serem dirigidos por outros de fora da equipe. Equipes multifuncionais possuem todas as competências necessárias para completar o trabalho sem depender de outros que não fazem parte da equipe.
- O *Product Owner* (PO), ou gerente do produto, é o responsável por maximizar o valor do produto e do trabalho da equipe de Desenvolvimento. Como isso é feito pode variar amplamente através das organizações, Times Scrum e indivíduos. O PO é a única pessoa responsável por gerenciar o *Backlog* do Produto.
- O *Scrum Master* é responsável por garantir que o Scrum seja entendido e aplicado. Ele faz isso para garantir que o Time Scrum adere à teoria, práticas e regras do Scrum. O *Scrum Master* é um apoio para o Time Scrum, removendo impedimentos e dificuldades do projeto, para que a equipe possam concentrar em suas tarefas. O *Scrum Master* ajuda aqueles que estão fora do Time Scrum a entender quais as suas interações com o Time Scrum são úteis e quais não são.

O coração do Scrum é o Sprint, um período de um mês ou menos, durante o qual uma versão incremental potencialmente utilizável do produto é criada. Sprints tem durações coerentes em todo o esforço de desenvolvimento. Um novo Sprint inicia imediatamente após a conclusão do Sprint anterior. Durante um Sprint a equipe Scrum se organiza para produzir um incremento do produto.

Os Sprints são compostos por uma reunião de planejamento da Sprint, reuniões diárias, o trabalho de desenvolvimento, uma revisão da Sprint e a retrospectiva da Sprint. O trabalho a ser realizado no Sprint é planejado na reunião de planejamento. Este plano é criado com o trabalho colaborativo de todo o Time Scrum. A reunião de planejamento do Sprint possui duração de oito horas para uma Sprint de um mês de duração. Para Sprints menores, este evento deve ser proporcionalmente menor. Por exemplo, um Sprint de duas semanas terá uma reunião de planejamento de Sprint de quatro horas. A reunião de planejamento da Sprint consiste em duas partes, cada uma sendo um período de metade do tempo de duração da reunião de planejamento do Sprint.

A Reunião Diária do Scrum é um evento time-boxed de 15 minutos, para que a Equipe de Desenvolvimento possa sincronizar as atividades e criar um plano para as próximas 24 horas. Esta reunião é feita para inspecionar o trabalho desde a última Reunião Diária, e prever o trabalho que deverá ser feito antes da próxima Reunião Diária. Ela é mantida no mesmo horário e local todo dia para reduzir a complexidade. A Reunião Diária do Scrum tem por objetivo disseminar o conhecimento sobre o que foi feito no dia anterior, identificar quais são os problemas, impedimentos existentes e priorizar o trabalho a ser realizado no dia que se inicia. Três questões-chaves são formuladas e respondidas por todos os membros da equipe:

- O que você fez desde a última reunião da equipe?
- Que obstáculos você está encontrando?
- O que você planeja realizar até a próxima reunião da equipe?

Outras reuniões que acontecem no ciclo de vida Scrum:

- Sprint Retrospective (Retrospectiva da Iteração): reuniões com objetivo de identificar o que funcionou bem, o que precisa ser melhorado e que medidas para melhorar.
- Sprint Planning Meeting (Reunião de Planejamento de Iteração): é uma reunião na qual estão presentes o Product Owner, o Scrum Master e todo o Scrum Team, bem como qualquer pessoa interessada que esteja representando a gerência ou o cliente. São descritas as funcionalidades de maior prioridade para equipe. São realizadas perguntas pela equipe com objetivo de quebrar as funcionalidades em tarefas técnicas.
- Sprint Review Meeting (Reunião Revisão de Iteração): Durante esta reunião, o Scrum Team mostra o que foi alcançado durante o Sprint. Tipicamente, isso tem o formato de um demo das novas funcionalidades.

Os artefatos do Scrum representam o trabalho ou o valor das várias maneiras que são úteis no fornecimento de transparência e oportunidades para inspeção e adaptação. Os artefatos definidos para o Scrum são especificamente projetados para maximizar a transparência das informações chave necessárias para assegurar que o Time Scrum tenha sucesso na entrega do incremento “Pronto”.

O Backlog do Produto é uma lista ordenada de tudo que deve ser necessário no produto, e é uma origem única dos requisitos para qualquer mudança a ser feita no produto. O PO é responsável pelo Backlog do Produto, incluindo seu conteúdo, disponibilidade e ordenação. O Backlog da Sprint é um conjunto de itens do Backlog do Produto selecionados para a Sprint, juntamente com o plano de entrega do incremento do produto e atingir o objetivo da Sprint. O *Backlog* da Sprint é a previsão da Equipe de Desenvolvimento sobre qual funcionalidade estará no próximo incremento e do trabalho necessário para entregar a funcionalidade. O *Backlog* da Sprint define qual trabalho a Equipe de Desenvolvimento realizará para converter os itens do Backlog do Produto em um incremento “Pronto”. O *Backlog* do Produto torna visível todo o trabalho que a Equipe de Desenvolvimento identifica como necessário para atingir o objetivo da Sprint.

Histórias de usuário se tornaram mais conhecidas inicialmente, como uma unidade de funcionalidade no XP. Outra definição mais atual seria: Uma Sentença curta de intenção que descreve algo que o sistema deve fazer para o usuário. A grande diferença aqui é a palavra intenção. Esta palavra mostra

que a história não é algo rígido, não é um contrato, mas sim algo que pode ser negociável e principalmente, ainda não está claro o suficiente para o usuário.

As histórias de usuário devem ser escritas de maneira a serem compreendidas por usuários e desenvolvedores. Elas focam no valor definido pelo usuário em vez de uma decomposição hierárquica funcional, como é comum quando trabalhamos com requisitos em projetos tradicionais. Muitas vezes as histórias são escritas em cartões pequenos, justamente para limitar o volume de informação que pode ser registrada. Mas também ferramentas específicas podem ser usadas. Detalhes do comportamento do sistema não aparecem na história. As histórias servem de base para a discussão dos requisitos, ou melhor, das intenções junto aos usuários. Assim, histórias de usuários não são especificações detalhadas de requisitos, mas sim expressões de intenção negociáveis. Representam pequenos incrementos de funcionalidade que podem ser desenvolvidos em dias ou semanas e por serem pequenas, são relativamente fáceis de se estimar seu esforço.

Existe um formato específico de escrita geralmente para as histórias de usuário. A sugestão de documentação segue o seguinte padrão de escrita.

Como um <papel>, eu posso/devo <ação> para que <valor para o negócio>

Este padrão foca no papel do usuário, a história é escrita para atender a um usuário e também no valor para o negócio. Desta maneira, evitamos que requisitos desnecessários sejam trabalhados. Seguindo este padrão sugerida para escrita de histórias de usuário, podemos citar um exemplo:

Eu, como Gerente de Equipe, devo visualizar as tarefas realizadas pelos membros da minha equipe para que possa acompanhar seu desempenho.

4. Unidade IV – Estilos e padrões arquiteturais

Estilos e padrões arquiteturais são princípios de desenho (projeto) reutilizáveis. A ideia é prover um conjunto de soluções reutilizáveis que identificam um conjunto de problemas que podem ser resolvidos dentro de um contexto específico. Esses estilos expressam organizações arquiteturais fundamentais para software. Eles proveem um conjunto de subsistemas pré-definidos, especificam suas responsabilidades e incluem regras e diretrizes para organizar as relações entre eles. Não podem ser considerados uma arquitetura de software completa, mas são uma orientação, um tipo de arquitetura, que pode ser usado como base na definição da arquitetura de um sistema e ainda para realizar comparações entre diferentes arquiteturas. A escolha de um estilo arquitetural a ser adotado em um sistema pode ser considerada o primeiro passo ao desenhar a arquitetura de um sistema, mas ainda é necessário detalhar e especificar componentes e formas de comunicação e integração entre eles.

Uma das formas comuns de se visualizar e documentar a arquitetura de um sistema é em camadas. As camadas ajudam a decompor aplicações que podem ser subdivididas em grupos de subtarefas no qual cada subgrupo está em um nível de abstração particular. Ao se definir um sistema em camadas, devem ser definidos o formato, conteúdo e significado de todas as mensagens trocadas entre as camadas. A divisão em camadas tem o propósito de se fazer uma divisão do trabalho. Assim, cada camada lida com um aspecto da comunicação. São utilizadas para sistemas grandes e complexos que demandam decomposição. O uso de camadas traz vários benefícios como reuso de camadas, suporte para padronização, manter dependências localmente, manutenibilidade, facilidade na trocada da camada. Como desvantagem, em função da necessidade de comunicações e verificações entre camadas, pode haver alguma perda de desempenho.

Um tipo de sistema que demanda um estilo arquitetural diferenciado são os sitemas interativos. Estes sistemas são baseados em um conjunto de requisitos funcionais, que podem ser mais estáveis, mas também em interfaces de usuário, que podem ser alteradas com maior frequência. Neste caso, a arquitetura deve possibilitar que alterações na interface sejam realizadas sem impactar o atendimento aos requisitos funcionais.

Um dos modelos conhecidos para sistemas interativos é o MVC (Model-View-Controller), que consiste em dividir a aplicação em três componentes: o Modelo, responsável por gerenciar as funcionalidades básicas e dados; a Visão, responsável por exibir informação ao usuário e o Controle, responsável por gerenciar entrada e dados. Com a adoção do MVC, mudanças na interface devem ser fáceis de fazer, o modelo suporta distintos "*look-and-feel*" (visualizações) e a interface consegue refletir mudanças de dados imediatamente.

As responsabilidades entre os componentes do modelo é bem dividida. Os componentes de Visão exibem informação ao usuário, sendo que cada visão tem ao menos um componente de controle associado. Recebem entrada como eventos relacionados a movimento do mouse, botões e teclas. O Controle recebe a requisição do usuário e a valida; determina o que o usuário deseja fazer e obtém dados do modelo. É responsável ainda por selecionar a visão que o cliente deve ver. O Modelo representa os objetos da aplicação e define estes objetos abstratamente. Instâncias e valor dos objetos constituem o estado da aplicação. Como benefícios do uso do MVC temos a possibilidade de ter múltiplas visões do mesmo modelo, visões sincronizadas (mecanismo de propagação), controles e visões "plugáveis", substituição fácil de "look and feel". Com o MVC, os desenvolvedores de interfaces gráficas focam exclusivamente em seu trabalho, sem se preocupar com lógica do negócio, e na implementação da lógica do negócio sem se preocupar com recursos de interface gráfica.

Outro tipo de sistema que merece destaque do ponto de vista arquitetural são os sistemas distribuídos. Para este tipo de sistema, um padrão possível é o chamado Broker. Broker pode ser utilizado para estruturar sistemas de software distribuídos que interagem por chamadas remotas. Um componente broker é responsável por coordenar comunicação, como, enviar solicitações e transmitir resultados e exceções. Ele atua como intermediário entre clientes e servidores. É usado em um contexto de sistema distribuído e heterogêneo com cooperação entre componentes independentes. São requisitos para uso deste padrão: Flexibi-

lidade, escalabilidade; independência do mecanismo de comunicação. O broker precisa ainda adicionar, remover, localizar e ativar componentes.

A interação entre clientes e servidores é baseada em um modelo dinâmico: os servidores podem atuar como clientes e os clientes não precisam conhecer a localização dos servidores que eles acessam. Assim, o componente broker atua para separar melhor clientes de servidores e realizar a comunicação entre eles. Os servidores se registram no broker e tornam seus serviços disponíveis a clientes através de interfaces definidas. Os clientes, por sua vez, acessam funcionalidades dos servidores através de requisições ao broker. A arquitetura broker é flexivel. Os objetos podem ser alterados facilmente. O componente Broker deve possuir algum modo de localizar solicitante de requisições, oferecer APIs a clientes e servidores, registrar servidores e chamar métodos de servidores.

Outro tipo de arquitetura discutida e utilizada atualmente é a arquitetura orientada a serviços (SOA – Service Oriented Architecture). Um serviço é um componente que atende a uma função de negócio específica e recebe requisições e as responde ocultando todo o detalhamento do processamento. Na arquitetura SOA, as unidades (serviços) existem de forma autônoma não isoladas umas das outras. Precisam se adequar a princípios e padronização, mas estas convenções trazem padrões benéficos aos consumidores sem influenciar cada negócio.

SOA é então um estilo ou paradigma de desenvolvimento cujo objetivo é criar módulos funcionais – serviços – com baixo acoplamento e permitindo reutilização. Acoplamento pode ser definido como a medida da interdependência entre dois ou mais módulos. Quanto mais acoplados, mais dependentes e vulneráveis a mudanças. Coesão está relacionada à ligação entre elementos internos com os outros para executar a função do módulo. Quanto mais coesos, mais fáceis de manter e menos afetados por mudanças. Web Services devem ser componentes fracamente acoplados. Devem poder continuar a funcionar mesmo com alterações na comunicação como adição de novos parâmetros, omissão de parâmetros, mudança de ordem de parâmetros. Outro conceito relacionado ao projeto de serviços é o de granularidade. Se pensamos em granularidade fina, descemos em nível de classes. Trata-se de um projeto muito detalhado. Ao pensar em granularidade grossa, ocultamos o modelo de classes básico. Os componentes executam mais de uma função. SOA visa criar componentes de granularidade grossa. Por fim, outra característica importante de serviços é a capacidade de manutenção de estado. Se saída de um componente é afetada pelo seu estado interno (não somente pela entrada), diz-se que é um componente stateful, como por exemplo, armazenar código do cliente para mostrar produtos de que gosta. São problemáticos, exigem maior acoplamento, de difícil manutenção e baixa reutilização. Componentes web services são ditos stateless.

Podemos assim resumir os seguintes princípios do Projeto de Serviços:

- Fracamente acoplados: Mantém uma relação que minimiza dependências e apenas têm conhecimento de outros.
- Contrato do serviço: Serviços concordam com um contrato definido por uma ou mais descrições de serviço e documentos relacionados.
- Autonomia: Possuem controle sob a lógica que encapsulam.
- Abstração: Escondem lógica do mundo externo.
- Reusabilidade: Lógica dividida em serviços para promover reuso.
- Ausência de estado: Serviços minimizam armazenamento sobre informações específicas de suas atividades.
- Serviços são projetados para serem encontrados e acessados por mecanismos específicos.