2
Frameworks: Conceitos Gerais

Um dos principais objetivos da Engenharia de Software é o reuso. Através da reutilização de software obtém-se o aumento da qualidade e redução do esforço de desenvolvimento (Gimenes & Huzita, 2005). A orientação a objetos fornece funcionalidades para que classes possam ser reutilizadas, bem como métodos por meio de seus mecanismos de herança e polimorfismo (Lundberg & Mattsson 1996). Componentes de software definem unidades reutilizáveis que oferecem serviços através de interfaces bem definidas (Gimenes & Huzita, 2005). Padrões de Projeto (*Design Patterns*) (Gamma et al., 1995) é outra abordagem mais abstrata que objetiva a reutilização de projetos de soluções para problemas recorrentes.

Além destas formas de reutilização, a tecnologia de frameworks possibilita que uma família de produtos seja gerada a partir de uma única estrutura que captura os conceitos mais gerais da família de aplicações (Pinto, 2000).

Este capítulo apresenta os principais conceitos sobre frameworks encontrados na literatura. Primeiramente, são revistas algumas definições sobre frameworks. Logo após, os frameworks são classificados em duas categorias: frameworks de aplicação orientado a objetos e frameworks de componentes. São apresentados os papéis envolvidos no desenvolvimento e uso de frameworks e os benefícios e desafios decorrentes da adoção de frameworks. O capítulo é encerrado com uma comparação entre frameworks e outras abordagens que visam o reuso.

2.1.
Definições de Frameworks

A literatura é repleta de definições de frameworks. As principais são descritas a seguir.

Para Fayad et al (1999b) e Johnson & Foote (1988), um framework é um conjunto de classes que constitui um projeto abstrato para a solução de uma família de problemas.

Segundo Mattsson (1996, 2000), um framework é uma arquitetura desenvolvida com o objetivo de atingir a máxima reutilização, representada como um conjunto de classes abstratas e concretas, com grande potencial de especialização.

Para Johnson (1991) e Gamma et al (1995), um framework é um conjunto de objetos que colaboram com o objetivo de atender a um conjunto de responsabilidades para uma aplicação específica ou um domínio de aplicação.

Já segundo Buschmann et al. (1996), Pree (1995) e Pinto (2000) um framework é definido como um software parcialmente completo projetado para ser instanciado. O framework define uma arquitetura para uma família de subsistemas e oferece os construtores básicos para criá-los. Também são explicitados os lugares ou pontos de extensão (*hot-spots*) nos quais adaptações do código para um funcionamento específico de certos módulos devem ser feitas.

Apesar de diferentes, as definições encontradas na literatura não são contraditórias. A definição adotada nesta dissertação é a de Buschmann et al. (1996), Pree (1995) e Pinto (2000).

2.2. Classificação dos Frameworks

Frameworks podem ser classificados de diversas formas. Inicialmente, são classificados em dois grupos principais: frameworks de aplicação orientado a objetos (Fayad et al., 1999a, b; Fayad & Johnson, 2000) e framework de componentes (Szyperski, 1997).

2.2.1. Frameworks de Aplicações Orientado a Objetos

Frameworks de aplicação orientado a objetos, também chamados de frameworks de aplicação, geram famílias de aplicações orientadas a objetos. Seus pontos de extensão são definidos como classes abstratas ou interfaces, que são estendidas ou implementadas por cada instância da família de aplicações. Segundo

Fayad et al (1999b), frameworks de aplicação são classificados quanto ao seu escopo em frameworks de infra-estrutura de sistemas, frameworks de integração de middleware e frameworks de aplicações corporativas.

Frameworks de infra-estrutura de sistemas simplificam o desenvolvimento de sistemas de infra-estrutura portáveis e eficientes, como sistemas operacionais, frameworks de comunicação, de interfaces gráficas e ferramentas de processamento de linguagens.

Frameworks de integração de middleware são usados para integrar aplicações e componentes distribuídos. Estes frameworks escondem o baixo nível da comunicação entre componentes distribuídos, possibilitando que os desenvolvedores trabalhem em um ambiente distribuído de forma semelhante a que trabalham em um ambiente não distribuído. São exemplos de frameworks de integração de middleware Java RMI (RMI-IIOP, 2005) e frameworks ORB (*Object Request Broker*).

Frameworks de aplicações corporativas, ao contrário dos frameworks de infra-estrutura e de integração de middleware, são voltados para um domínio de aplicação específico, como por exemplo, os domínios da aviação, telecomunicações e financeiro. Frameworks de aplicações corporativas são mais caros de serem desenvolvidos ou comprados quando comparados com os de integração de middleware e de infra-estrutura, pois são necessários especialistas do domínio de aplicação para construí-lo. Entretanto, eles podem prover um retorno substancial já que eles encapsulam o conhecimento sobre o domínio onde a instituição atua (Fayad et al., 1999b).

Frameworks de infra-estrutura e de integração de middleware fornecerem mecanismos para integrar componentes distribuídos e tratar aspectos de infra-estrutura, o desenvolvedor da aplicação abstrai os detalhes em baixo nível e concentra-se no objetivo principal da aplicação. Estes frameworks tratam aspectos comuns a diversos domínios de aplicação de forma que quase sempre são encontradas soluções disponíveis no mercado e na maior parte das vezes é mais vantajoso buscar uma solução existente do que desenvolvê-la internamente.

Além da classificação por escopo, segundo Fayad et al (1999b) frameworks podem ainda ser classificados quanto à forma usada para estendê-los. Frameworks caixa branca, ou *white box*, são instanciados usando herança, através da implementação de *Template Methods* (Gamma et al., 1995), métodos abstratos

que são implementados na subclasse, ou da redefinição de métodos ganchos (*hook methods*) (Pree, 1995), métodos com uma implementação padrão que pode ser redefinida na subclasse. Frameworks caixa preta, ou *black box*, são instanciados através de configurações e composições, através da definição de classes que implementam uma determinada interface ou contrato. Os pontos de extensão dos frameworks caixa preta freqüentemente são definidos seguindo o padrão de projetos *Strategy* (Gamma et al., 1995).

Frameworks caixa branca exigem que o desenvolvedor responsável pela instanciação do framework possua um bom conhecimento sobre sua estrutura interna e normalmente são mais difíceis de usar. Por outro lado, frameworks caixa preta não exigem tanto conhecimento sobre a estrutura interna do framework, mas são menos flexíveis. Frameworks caixa cinza, ou *gray box*, possuem as características conjuntas dos frameworks caixa branca e preta, de forma a tentar evitar as desvantagens dos dois.

2.2.2. Frameworks de Componentes

Segundo Szyperski (1997), um framework de componentes é uma entidade de software que provê suporte a componentes que seguem um determinando modelo e possibilita que instâncias destes componentes sejam plugadas no framework de componentes. Ele estabelece as condições necessárias para um componente ser executado e regula a interação entre as instâncias destes componentes. Um framework de componente pode ser único na aplicação, criando uma ilha de componentes ao seu redor, ou pode cooperar com outros componentes ou frameworks de componentes. Alguns exemplos de frameworks de componentes são o OpenDoc (2005) e o BlackBox (Oberon, 2005).

A principal diferença entre frameworks de aplicação orientado a objetos e framework de componentes é que, enquanto frameworks de aplicações definem uma solução inacabada que gera uma família de aplicações, um framework de componentes estabelece um contrato para plugar componentes. Usando uma analogia, um framework de aplicação equivale ao quadro de uma bicicleta. Há lugares específicos (os pontos de extensão) onde o guidão, as rodas, o banco e os pedais (implementações dos pontos de extensão) podem ser encaixados. Bicicletas

diferentes (instâncias do framework) podem ser compostas variando estes itens: bicicletas com marchas, com amortecedores, etc., mas apenas os pontos previstos podem variar e o produto final será sempre uma bicicleta. Por outro lado, frameworks de componentes podem ser vistos como uma placa de circuito integrado com *slots* onde os chips (componentes) podem ser encaixados para criar uma instância. Como os frameworks de componentes, as placas são utilizadas para compor soluções para diversos domínios como por exemplo, uma placa de vídeo ou um modem.

2.3. Papéis Envolvidos no Uso e Desenvolvimento de um Framework

Há vários profissionais envolvidos na criação, manutenção e uso (instanciação) de um framework. Estes papéis, que não precisam necessariamente ser exercidos por pessoas diferentes, são segundo Froehlich et al. (1997a), Froehlich et al (1997b) e Pinto (2000): projetista do framework, mantenedor do framework e o desenvolvedor de aplicações (usuário do framework).

O projetista do framework é responsável pelo desenvolvimento da estrutura básica do framework. O projetista determina os pontos de extensão do frameworks (*hot spots*) e realiza o levantamento de requisitos.

O mantenedor do framework é responsável por redefinir e acrescentar novas funcionalidades ao projeto do framework, possibilitando que novas características sejam incorporadas ao framework no momento da instanciação pelos desenvolvedores de aplicação.

O desenvolvedor de aplicações usa o framework. Ele satisfaz os pontos de extensão para gerar a aplicação, instância do framework.

2.4. Conseqüências da Adoção de Frameworks

A utilização de frameworks no desenvolvimento de aplicações traz benefícios, originados de suas características principais: são modulares, reusáveis, extensíveis e eventualmente assumem o controle da execução invocando métodos da aplicação quando necessário (inversão de controle). Por outro lado, há certos

desafios na criação e uso de frameworks que não podem ser ignorados. Esta seção lista as principais vantagens e desafios decorrente da adoção de frameworks.

2.4.1. Benefícios Decorrentes da Utilização de Frameworks

Os principais benefícios decorrentes da utilização de frameworks, segundo Fayad et al. (1999b) e Pinto (2000) advêm da modularidade, reusabilidade, extensibilidade e inversão de controle que os frameworks proporcionam.

Frameworks encapsulam detalhes de implementação voláteis através de seus pontos de extensão, interfaces estáveis e bem definidas, aumentando a modularidade da aplicação. Os locais de mudanças de projeto e implementação da aplicação construída usando o framework são localizados, diminuindo o esforço para entender e manter a aplicação.

Além disso, frameworks incentivam o reuso, pois capturam o conhecimento de desenvolvedores em determinado domínio ou aspecto de infra-estrutura ou de integração de middleware. As interfaces do framework promovem pontos de extensão que as aplicações estendem gerando instâncias que compartilham as funcionalidades definidas no framework. Desta forma, aumentam a produtividade dos desenvolvedores, pois o processo não começa do zero, a qualidade e a confiabilidade do produto final, pois idealmente as funcionalidades do framework já foram testadas em várias instâncias, e a interoperabilidade de software, pois as instâncias de uma mesma família compartilham características em comum.

Frameworks oferecem pontos de extensão explícitos que possibilitam aos desenvolvedores estenderem suas funcionalidades para gerar uma aplicação. Os pontos de extensão desacoplam as interfaces estáveis do framework e o comportamento de um determinado domínio de aplicação das variações requeridas por uma determinada aplicação em um dado contexto.

Por fim, alguns frameworks apresentam inversão de controle (*Inversion of Controll* – IoC). Inversão de controle, também é conhecida como princípio de Hollywood : "*Don't call us, we will call you*" (Larman, 2004). Trata-se de transferir o controle da execução da aplicação para o framework, que chama a aplicação em determinados momentos como na ocorrência de um evento. Através da IoC o framework controla quais métodos da aplicação e em que contexto eles

serão chamados em decorrência de eventos, como eventos de janela, um pacote enviado para determinada porta, entre outros.

2.4.2. Desafios Decorrentes da Utilização de Frameworks

Quando usado em conjunto com padrões de projetos, bibliotecas de classes, componentes, entre outros, frameworks de aplicação têm o potencial para aumentar a qualidade de software e reduzir o esforço de desenvolvimento. Contudo, instituições que tentam construir ou usar frameworks freqüentemente falham a menos que resolvam os seguintes desafios: esforço de desenvolvimento, curva de aprendizagem, integração, eficiência, manutenção, validação e remoção de defeitos e falta de padrões (Fayad et al., 1999b).

Destes desafios, o esforço de desenvolvimento afeta principalmente os projetistas de frameworks. A curva de aprendizagem, a integração e a eficiência afetam principalmente os desenvolvedores de aplicação. A manutenção e validação e remoção de defeitos afetam tanto desenvolvedores de aplicação quanto mantenedores de frameworks e a falta de padrões afetam a todos envolvidos no desenvolvimento e uso dos frameworks.

O desafio do esforço de desenvolvimento ocorre devido à alta complexidade envolvida no projeto de um framework. Se desenvolver aplicações complexas é difícil, desenvolver frameworks que sejam de alta qualidade, extensíveis e reusáveis para domínios de aplicações complexos ou para tratar aspectos de infra-estrutura ou de integração de middleware é ainda mais difícil. Projetistas de frameworks devem estar cientes dos custos decorrentes do desenvolvimento de frameworks de aplicação.

Quanto à curva de aprendizagem, aprender a usar um framework de aplicação leva tempo e incorre em custos. A menos que o esforço de aprendizado possa ser amortizado através do uso do framework em muitos projetos, ou em projetos duradouros, o investimento pode não compensar. Deve ser analisado se os benefícios trazidos pelo framework compensam os custos com o treinamento da equipe de desenvolvimento.

O desafio da integração ocorre quando diversos frameworks são usados com outras tecnologias e com sistemas legados não previstos pela arquitetura do

framework. Antes de usar um framework o desenvolvedor de aplicações deve analisar possíveis problemas que possam surgir ao incorporar novos frameworks à aplicação.

Além disso, aplicações usando frameworks podem apresentar desempenho inferior a aplicações específicas. Para tornarem-se extensíveis, frameworks de aplicações adicionam níveis de indireção. Aplicações genéricas tendem a obter desempenho inferior a aplicações específicas (Fayad et al., 1999b). Desta forma o desenvolvedor de aplicações deve considerar se os ganhos advindos do reuso compensam uma potencial perda de desempenho.

Frameworks inevitavelmente precisam ser mantidos. A manutenção em frameworks toma diversas formas como, correção de erros de programação, acréscimo ou remoção de funcionalidades, acréscimos ou remoção de pontos de extensão, etc. A manutenção é realizada pelos mantenedores do framework, que precisam ter um conhecimento profundo sobre os componentes internos e suas interdependências. Em alguns casos, a própria instância do framework sofre manutenção para acompanhar a evolução do framework. Cabe ao desenvolvedor da aplicação considerar o esforço de manutenção gasto para obter os benefícios de uma nova versão do framework.

Além disso, a validação e remoção de erros de um framework ou de uma aplicação que usa um framework é um desafio, pois o framework é uma aplicação inacabada, o que torna difícil tanto para os mantenedores de frameworks quando para os desenvolvedores de aplicação testar, respectivamente, o framework e a instância do framework isoladamente. Aplicações que usam framework costumam ser mais difíceis de depurar, pois devido a IoC, o controle da execução oscila entre a aplicação e o framework.

Por fim, ainda não há um padrão amplamente aceito para desenvolver, documentar, implementar e adaptar frameworks. Todos os papéis envolvidos no desenvolvimento e uso do framework são afetados por este desafio, sendo que os mais afetados são os mantenedores do framework e o desenvolvedor de aplicações. O primeiro, pois precisa ter um conhecimento profundo sobre os componentes internos e suas interdependências. O segundo, pois a falta de um padrão de documentação de frameworks dificulta o seu uso.

2.5. Frameworks e Outras Abordagens

Esta seção realiza uma breve comparação entre frameworks e outras técnicas de desenvolvimento de software que objetivam o reuso. Dentre elas, são analisadas o projeto de aplicações orientado a objetos, biblioteca de classes, padrões de projeto e componentes de software (Pinto, 2000).

2.5.1. Frameworks e Aplicações Orientado a Objetos

Uma aplicação orientada a objetos descreve um programa executável completo, atendendo aos requisitos definidos nos documentos de especificação. Já os frameworks são incompletos e não são executáveis. Eles capturam funcionalidades comuns a diversas aplicações, mas declaram pontos de extensão que devem ser preenchidos pela aplicação que instancia o framework.

Frameworks geram famílias de aplicações orientadas a objetos através do preenchimento dos pontos de extensão, mas aplicações orientadas a objetos não geram famílias de frameworks.

2.5.2. Frameworks e Biblioteca de Classes

Bibliotecas de classes são conjuntos de classes relacionadas com um propósito geral. Por outro lado, as classes de um framework estão relacionadas a um determinado domínio ou a algum determinado aspecto de infra-estrutura ou de integração de middleware. Além disso, as classes da biblioteca são reutilizadas individualmente enquanto que as classes de um framework são reutilizadas em conjunto.

Frameworks tendem a ter um impacto maior do que bibliotecas de classes na arquitetura da aplicação. Frameworks ativos (*calling frameworks*) (Matsson, 2000), que utilizam inversão de controle, podem assumir o controle de execução da aplicação ao contrário de bibliotecas de classes que são sempre passivas.

2.5.3. Frameworks e Padrões de Projeto

Padrões de projeto são soluções mais abstratas do que um framework. Frameworks são normalmente implementados utilizando linguagens orientadas a objetos como Java, C++, etc. Padrões de projeto descrevem o projeto de uma solução para um problema recorrente, podendo incluir um exemplo de implementação. Além disso, frameworks são mais específicos que padrões de projetos, pois estão relacionados a um domínio de aplicação, um aspecto de infra-estrutura ou de integração de middleware. Já padrões de projetos são mais gerais e podem ser usado em diversas situações independente de domínio de aplicação.

Por fim, frameworks são construções mais complexas que padrões de projeto. Enquanto um framework pode possuir vários padrões de projeto, o contrário não é verdadeiro.

2.5.4. Frameworks e Componentes de Software

Componentes de software podem ser definidos como unidades independentes, que encapsulam dentro de si seu projeto e implementação e oferecem serviços para o meio externo através de interfaces bem definidas (Gimenes & Huzita, 2005). Componentes podem fornecer dois tipos de interface: as interfaces fornecidas, que expõem seus serviços e as interfaces requeridas, por onde o componente explicita suas dependências.

Frameworks também fornecem interfaces bem definidas, os pontos de extensão, que as aplicações devem estender. Contudo, enquanto um framework possui necessariamente pontos de extensão, um componente totalmente autocontido não possui interfaces requeridas. Além disso, um componente deve ser plugado a uma interface requerida de outro componente enquanto que os pontos de extensão dos frameworks são menos exigentes, possibilitando que sejam plugados componentes, classes ou qualquer artefato que realiza o contrato definido.

Por fim, componentes podem ser desenvolvidos usando frameworks e frameworks podem ser desenvolvidos usando componentes. Estas duas

tecnologias são complementares de forma que frameworks podem ser usados para auxiliar o desenvolvimento de componentes e vice e versa (Szyperski, 1997).

2.6. Conclusão

Um dos principais objetivos da engenharia de software é o reuso. Através da reutilização de software obtém-se o aumento da qualidade e redução do esforço de desenvolvimento. Frameworks possibilitam o desenvolvimento de família de aplicações, geradas a partir de uma estrutura que captura os conceitos mais gerais das aplicações.

São encontradas na literatura várias definições de frameworks. A usada nesta dissertação é a de Buschmann et al (1996), Pree (1995) e Pinto (2000) na qual um framework é um software parcialmente completo projetado para ser instanciado. Isto define uma arquitetura para uma família de subsistemas e oferece os construtores básicos para criá-los. Também são explicitados os pontos de extensão nos quais adaptações do código para um funcionamento específico de certos módulos devem ser feitas.

Frameworks podem ser classificados segundo Fayad et al (1999a), (1999b) e (2000) em framework de aplicação orientado a objetos ou segundo Szyperski (1997) em framework de componentes. O primeiro define uma estrutura para o desenvolvimento de aplicações orientadas a objetos enquanto que o segundo define uma infra-estrutura de execução onde componentes podem ser plugados.

Frameworks de aplicação podem ser classificados quanto ao seu escopo como frameworks de infra-estrutura, que simplificam o desenvolvimento de sistemas de infra-estrutura portáveis e eficientes, de integração de middleware, usados para integrar aplicações e componentes distribuídos, e de aplicações corporativas, voltados para um domínio de aplicação específica. Ao longo desta dissertação é apresentado como frameworks de aplicação, em especial frameworks de infra-estrutura, podem ser combinados para fornecer uma infra-estrutura com serviços de gerenciamento de persistência de objetos, controle de transações, etc. para um groupware.

Frameworks de aplicação podem ainda ser classificados quanto ao seu uso como frameworks caixa branca, que exige um bom conhecimento da estrutura

interna do framework por parte do usuário do framework, caixa preta, que é menos flexível que o framework caixa branca, mas possibilita o seu uso sem que o desenvolvedor precise conhecer a estrutura interna do framework, e caixa cinza, que tenta combinar as vantagens dos frameworks caixa branca e preta.

Há três papéis envolvidos no uso e desenvolvimento de frameworks: o projetista do framework, responsável pela estrutura interna do framework, pelo levantamento de requisitos e pela definição dos pontos de extensão; o mantenedor do framework, responsável por redefinir e acrescentar novas funcionalidades ao projeto do framework e o desenvolvedor de aplicações que instancia o framework. Nesta dissertação os frameworks são analisados sobre a ótica do desenvolvedor de aplicações.

O desenvolvimento com frameworks traz benefícios. Frameworks diminuem o esforço para entender e manter a aplicação. Frameworks incentivam o reuso, pois capturam o conhecimento de desenvolvedores em determinado domínio ou aspecto de infra-estrutura ou de integração de middleware. Além disso, frameworks são expansíveis por construção e, por fim, o uso de inversão de controle simplifica o desenvolvimento de aplicações.

Ao longo desta dissertação são analisados frameworks de aplicação incorporados à arquitetura técnica do AulaNet 3.0. Estes frameworks são usados para compor uma base de serviços de infra-estrutura sobre a qual groupwares podem ser desenvolvidos. Dessa forma, o desenvolvedor de groupware pode concentrar-se no seu domínio enquanto que os frameworks de infra-estrutura cuidam de aspectos técnicos, como persistência de dados, por exemplo. No capítulo 3 são analisados frameworks para a camada de negócios enquanto que no capítulo 4 são analisados frameworks para a camada de apresentação.

Apesar dos benefícios decorrentes do uso de frameworks, também são impostos desafios ao desenvolvimento de frameworks e de aplicações que usem os frameworks. Dentre eles, o esforço de desenvolvimento, imposto aos projetistas de frameworks, a curva de aprendizagem, a integração e a eficiência, que afetam principalmente os desenvolvedores de aplicação, a manutenção e validação e remoção de defeitos, que afetam tanto desenvolvedores de aplicação quanto mantenedores de frameworks e o desafio da falta de padrões que afetam a todos envolvidos no desenvolvimento e uso de frameworks.

O desafio da curva de aprendizagem é o principal ao considerar as características do AulaNet, pois ele é desenvolvido por alunos de graduação, mestrado e doutorado da PUC-Rio. Além a empresa EduWeb desenvolve customizações no AulaNet para seus clientes. Para lidar com este desafio, é considerada a quantidade de documentação disponível para cada framework além da disponibilidade de suporte. Além disso, é considerada a quantidade de mão de obra disponível no mercado já apta a trabalhar com o framework, eliminando assim o custo com o treinamento, e a quantidade de ferramentas compatíveis disponíveis, pois elas auxiliam o uso do framework, suavizando a curva de aprendizado.

Quanto à falta de padrões, sob a ótica do desenvolvedor este não é um desafio crítico, desde que haja uma quantidade de documentação suficiente, mesmo que não padronizada, sobre o framework.

Como o AulaNet 3.0 é desenvolvido sem reutilizar o código legado do AulaNet 2.1, sistemas legados não são problema e o desafio da integração se resume a integração com outros frameworks. Para lidar com este desafio, os frameworks são analisados tecnicamente para que a compatibilidade possa ser verificada. Além disso, considera-se o grau de intrusão dos frameworks, pois, quanto menos intrusivos, mais facilmente os artefatos do framework podem ser usados em outro contexto.

Quanto ao desafio da eficiência, este não é um problema crítico para o AulaNet. Os principais problemas presentes na arquitetura do AulaNet 2.1 e que levaram à especificação de uma nova arquitetura referem-se à manutenibilidade que ao longo dos anos tornou-se muito custosa. Sendo assim, considera-se que vale a pena sacrificar um pouco a eficiência para ganhar na manutenibilidade. Além disso, como está sendo adotada uma abordagem de desenvolvimento baseada em componentes, um problema crítico de eficiência decorrente do uso de determinado framework em um componente pode ser resolvido substituindo o componente por outro que não use o framework, de maneira pontual sem afetar os outros componentes que obtém eficiência satisfatória.

Por fim, é importante lidar com os desafios da manutenção, validação e remoção de defeitos. Frameworks pouco usados correm o risco de serem descontinuados enquanto que frameworks muito usados tendem a permanecer em desenvolvimento, recebendo manutenção e tendo seus erros corrigidos. Isto é

particularmente verdade em se tratando de frameworks de código aberto onde, mesmo que os projetistas originais do framework decidam não realizar mais manutenção nele, haverá desenvolvedores dispostos a assumir a tarefa. Ao verificar o grau de aceitação de mercado dos frameworks, apenas frameworks de grande aceitação são incorporados à arquitetura do AulaNet 3.0.