Reengenharia de Software: Um Estudo de Caso com a Metodologia OpenUP e Aplicação do Zend Framework

Thales V. Maciel, Alex Sandro E. Valério

***Resumo*— Eventualmente, más práticas de desenvolvimento de software, como a não adoção de um processo bem definido, resultam em complexidade desnecessária na atividade de manutenção. Com o passar do tempo, essas dificuldades tornam a manutenibilidade de um sistema praticamente nula, circunstância sob a qual é aberto precedente à ação de reengenharia do mesmo. Este trabalho objetivou apresentar os impactos oriundos da fundamentação teórica em engenharia de software em um estudo de caso de reengenharia e os resultados da adoção da metodologia OpenUP no processo e o framework de aplicação Zend como base estrutural para implementação, atingindo melhoramento em manutenibilidade e reduzindo custos para a organização.**

***Palavras Chave*—Reengenharia de Software, Engenharia de Software, Desenvolvimento Ágil, OpenUP, Zend Framework.**

I. INTRODUÇÃO

NOS dias atuais, devido à dinamicidade dos negócios, é usual que modificações em produtos *software* sejam solicitadas com urgência. Contudo, isto comumente implica na ausência de tempo hábil para a documentação adequada dessas modificações no momento e na velocidade em que elas acontecem.

Nestes casos, o perigo está em os requisitos, o código e a documentação do *software* se tornarem inconsistentes, pois a prioridade pode ser dada apenas à efetivação das atualizações solicitadas. Com o passar do tempo, torna-se impraticável a recuperação desta consistência.

Este trabalho tem como objetivo o estudo da atividade de reengenharia de *software* quando motivada pela inviabilidade da continuação da manutenção em um produto desta natureza. São apresentados problemas recorrentes, causados pela não consideração de boas práticas no desenvolvimento que, eventualmente, levam uma equipe de profissionais a esta situação.

Para efeitos desta pesquisa, foi tomado o estudo de caso do portal de serviços acadêmicos utilizado na Universidade da Região da Campanha (URCAMP). É analisada sua situação perante a habitual necessidade de manutenção e, então, proposta uma estratégia de reengenharia do sistema com a adoção de uma metodologia ágil de desenvolvimento de *software* no processo, bem como o emprego de um *framework* de aplicação no desenvolvimento do produto.

É apresentada fundamentação teórica sobre manutenção e reengenharia de *software* e agilidade no processo, além de considerações sobre as tecnologias utilizadas. A Seção III expõe as propostas do estudo de caso e como a fundamentação teórica apresentada se aplica ao mesmo.

Thales V. Maciel é bacharel em sistemas de informação pela Universidade da Região da Campanha, especialista em sistemas distribuídos com ênfase em banco de dados pela Universidade Federal do Pampa, mestrando em Engenharia de Computação na Universidade Federal do Rio Grande, onde integra o Grupo de pesquisa em Automação e Robótica Inteligente - NAUTEC e professor do ensino básico, técnico e tecnológico no Instituto Federal Sul-rio-grandense Câmpus Bagé (e-mail: thales.maciel@furg.br).

Alex Sandro E. Valério é bacharel em informática pela Universidade da Região da Campanha, especialista em engenharia de software pela Universidade Federal de Lavras e professor do ensino básico, técnico e tecnológico no Instituto Federal Sul-rio-grandense Câmpus Bagé (e-mail: alexvalerio@ifsul.edu.br).

II. FUNDAMENTAÇÃO TEÓRICA

O trabalho é fundamentado sobre as matérias de reengenharia de *software* e desenvolvimento ágil. Dentro destes contextos, nesta seção, também são apresentadas questões básicas sobre a metodologia OpenUP, o *Zend Framework* e o padrão de projeto *Model-View-Controller*.

A. Manutenção e Reengenharia de Software

Manutenção de *software* é definida, pela norma IEEE 1219, como o conjunto de modificações efetuadas em um produto de *software* após sua disponibilização, seja para a correção de falhas, melhoramento em desempenho ou outros atributos, ou adaptação do produto a um ambiente modificado [1].

Entende-se que estas modificações devem ser efetuadas com a motivação de que o sistema continue a satisfazer os requisitos dos usuários e mantenha um determinado padrão de qualidade.

A modificação de um produto de *software*, para adição de novas funcionalidades, após sua entrega é uma atividade mais complexa (e consequentemente mais custosa) do que a etapa regular de desenvolvimento [2]. Isto acontece em razão da necessidade de se dedicar tempo ao entendimento do sistema e analisar o impacto da realização de tais modificações.

Portanto, esforços, durante o desenvolvimento, para fazer com que o *software* seja fácil de entender e modificar, em ocasiões futuras, podem implicar na redução dos custos de manutenção quando ela for necessária. O uso de boas práticas

de Engenharia de *Software* (ES) como a especificação documentada e o desenvolvimento orientado a objetos são fatores que contribuem para a redução destes custos [3].

Neste contexto, cabe a menção da abordagem caótica de desenvolvimento de *software* [2], usual na construção de aplicações voltadas para a *web*. Nela ocorre a integração de diversas tecnologias, geralmente no mesmo código-fonte, de forma intrínseca, de modo que, ao ler o produto desenvolvido, não fica evidente o propósito de seu desenvolvimento.

A má endentação de código e a ausência de comentários nos artefatos implementados, por exemplo, implicam, quando da atividade de manutenção, na necessidade de tempo adicional para o entendimento do código apresentado [4].

É entendido que nem todo sistema de *software* é desenvolvido com técnicas modernas de engenharia, pode ter sido estruturado de maneira inadequada e configurado para priorizar outros aspectos, que não a compreensibilidade e eficiência deste. Adicionalmente, outro fator que dificulta a manutenção de *software* é a falta de documentação, que pode ter sido perdida, estar mal formulada ou inconsistente.

Estas características podem resultar em desperdício de tempo em função da necessidade do conhecimento de detalhes sobre um projeto que, geralmente, são encontrados em sua documentação. Quando da inviabilidade de manutenção, a reengenharia de *software* é indicada para a resolução desses conflitos [3].

A reengenharia de *software* é definida, no *Software Engineering Body Of Knowledge* (SWEBOK), como a inspeção e alteração de *software* para reconstituí-lo em uma nova forma, incluindo a subsequente implementação desta nova forma [5]. Considerando isto, entende-se que a reengenharia é a mais radical e mais dispendiosa forma de manutenção de *software*.

Com o passar do tempo no ciclo de vida do *software*, a tendência é que sua estrutura passe a ser gradualmente comprometida pelas alterações realizadas, o que torna cada vez maior o esforço para entender e efetuar novas modificações. A tomada de decisão pela reengenharia, após certo tempo de manutenção, é uma situação bastante comum e chegar à conclusão que determinado produto de *software* não é mais passível de manutenção não é um problema novo [2].

Devido à esta problemática, uma organização pode optar pela reengenharia de um sistema legado tendo como objetivo o melhoramento de sua estrutura e manutenibilidade [3]. Neste contexto, o SWEBOK explica que, estatisticamente, a reengenharia de *software* não é tão usada apenas para o melhoramento de manutenibilidade, como é para a substituição de sistemas legados em virtude de precariedade nesta característica [5].

B. Metodologias de Desenvolvimento de Software

A adoção de uma metodologia ou processo de *software* no desenvolvimento depende do tipo de aplicação a ser desenvolvida e isto se torna indispensável ao se considerar os níveis de qualidade necessária e a criticidade atribuída a um projeto.

Apesar da existência de diversos processos de desenvolvimento propostos, há um consenso na comunidade de desenvolvimento de *software* sobre a inexistência do melhor processo de desenvolvimento, ou aquele que melhor se aplica a todas as situações de engenharia. Entende-se que cada processo tem suas particularidades em relação ao modo de distribuir e encadear as atividades que propõe [6].

Considerando a dinâmica da indústria nos dias de hoje, é dificultada a escolha por uma metodologia tradicional, como o modelo em cascata (clássico) ou o Processo Unificado (UP). Isto acontece porque estes, muitas vezes, se apresentam lentos e rígidos demais para suprir as demandas atuais de negócios e economia [2]. Nestas hipóteses, considera-se mais coerente a aplicação de um processo com ênfase no melhor aproveitamento do tempo, como uma metodologia ágil.

A propriedade que melhor caracteriza metodologias da escola ágil é a melhor resposta a mudanças. Isto acontece como um benefício de um de seus princípios, onde é proposta a aceitação de mudanças de requisitos, mesmo que sejam em uma etapa avançada do desenvolvimento [7]. Processos ágeis protegem a contínua expectativa de mudanças para beneficiar a competitividade do produto de *software* no mercado.

Metodologias desse padrão devem priorizar a construção de *software* funcional sobre a formulação de documentação extensa, assim como a dinamização de respostas a mudanças sobre continuidade de um plano e a colaboração entre indivíduos sobre a dependência de processos e ferramentas, entre outros aspectos [7].

Esta definição de prioridades não significa que boas práticas de engenharia e desenvolvimento de *software* devam ser desencorajadas ou ignoradas. De fato, métodos ágeis foram desenvolvidos como uma essencial evolução para superar pontos fracos na ES convencional [2].

Exemplos de processos ágeis são a Programação Extrema (XP), *Scrum*, Desenvolvimento Dirigido à Funcionalidades (FDD), Processo Ágil Unificado (AUP) e o *Open Unified Process* (*OpenUP*). Dentre eles, o *OpenUP* é apreciado positivamente no contexto deste estudo devido a sua proposta primordial. Trata-se da criação de uma síntese ou combinação das boas práticas e características de ambos o UP e metodologias ágeis [8].

C. OpenUP

O *Open Unified Process* (*OpenUP*) é baseado no UP e assemelha-se com o *Rational Unified Process* (RUP), sua implementação mais conhecida [9]. Porém, exibe uma apresentação mais leve de produtos de trabalho, tarefas e artefatos [10]. As fases de seu ciclo de desenvolvimento são quatro: Concepção, Elaboração, Construção e Transição, conforme a Fig 1.

O *OpenUP* é especificado como um processo minimalista, pois propõe o mínimo necessário de especificações formais, ou seja, produzir somente os artefatos que forem julgados essenciais dentro do contexto para a captura e comunicação de ideias; completo o suficiente para cobrir todas as disciplinas essenciais num ciclo de vida de desenvolvimento de *software*; e extensível, pois foi projetado para permitir sua extensão de acordo com as necessidades e peculiaridades de cada projeto,

individualmente [9].

A abordagem iterativa e incremental do *OpenUP* (Fig. 1) provoca os desenvolvedores, de maneira saudável para o projeto, de forma que o foco seja direcionado à entrega de produtos valorizados pelas partes interessadas (*stakeholders*), ou seja, de maneira que a maior parte do trabalho executado seja com o objetivo de suprir os objetivos destes [8].

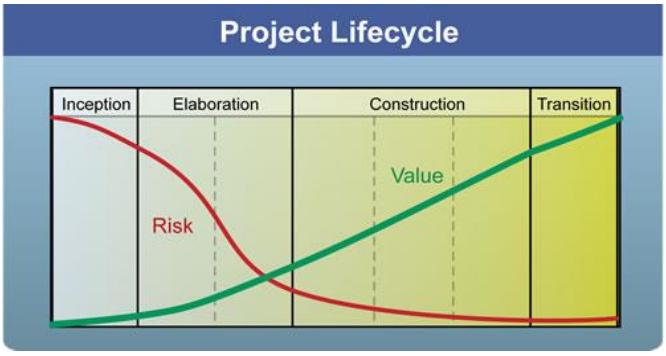

Fig. 1. Modelo de ciclo de vida do *OpenUP* [9].

Um efeito desta característica é a diminuição considerável do período de tempo gasto na chamada parálise de análise. Esta situação é caracterizada pela estagnação na produção dos artefatos de análise, onde os profissionais passam tempo demasiado na produção de modelos [6].

Outra característica baseada na natureza ágil do *OpenUP*, é a estimulação dos esforços individuais, cujos produtos são denominados micro incrementos ou unidades de trabalho. Com a aplicação de intensa colaboração dos *stakeholders* no desenvolvimento, é reduzido o tempo para o recebimento de *feedback* sobre o trabalho realizado, o que contribui para a agilidade do processo.

Considera-se o *OpenUP*, portanto, uma alternativa leve e ágil aos processos tradicionais de *software*, pois, além de mostrar agilidade na metodologia com ênfase nos resultados, mostra-se capaz de documentar satisfatoriamente os artefatos desenvolvidos. Adicionalmente, sua propriedade de extensibilidade o torna viável no que se trata da utilização em um processo de reengenharia.

D. Zend Framework

Um *framework* é uma estrutura genérica de *software* que pode ser estendida para a criação de uma aplicação específica, sendo implementado com o uso de orientação a objetos (OO) como uma coleção de classes e interfaces [3]. Um *framework*, sozinho, não é uma aplicação de *software*, mas sim uma base estrutural e lógica, ou seja, um arcabouço para as aplicações que venham a ser construídas sobre suas fundações.

A decisão pelo uso de um *framework* de aplicação no desenvolvimento de *software* depende da necessidade de estruturar um projeto desta natureza, pois em um projeto bem estruturado a qualidade do produto final é aumentada e o tempo e os custos de desenvolvimento são reduzidos [11].

O *Zend Framework* (ZF) é um *framework* de aplicação escrito em linguagem *PHP: Hypertext Preprocessor* (PHP) que se destaca por vantagens deste sobre outros *frameworks* da categoria. A flexibilidade alcançada pelo baixo acoplamento entre seus componentes, o rápido ciclo de desenvolvimento e lançamentos de atualizações, e o fato de que o ZF e sua documentação oficial são desenvolvidos e mantidos pela *Zend Technologies*, também desenvolvedora e mantenedora da linguagem PHP são características que garantem-no grande credibilidade na comunidade e no mercado [12].

Entende-se que o ZF é um conjunto de componentes orientados a objetos que têm o propósito da execução de tarefas de escopo amplo, podendo ser abordado pelo uso individual de seus componentes, ou pela utilização integral como base de uma aplicação [13]. Portanto, é considerado aplicável a projetos onde a metodologia adotada busca propriedades como a disciplina arquitetural do *software*, o controle dos riscos e, ao mesmo tempo, agilidade no processo de desenvolvimento, incluindo processos de reengenharia de *software*.

E. MVC

O ZF concretiza, através da implementação de seus componentes, especificações de soluções documentadas e heurísticas para diversos problemas em desenvolvimento de *software*, os chamados padrões de projeto (*design patterns*) [13].

O *Model-View-Controller* (MVC) é um desses padrões e, nele, é proposta a divisão arquitetural do produto de *software* em componentes lógicos de três categorias: os modelos, as visualizações e os controladores.

Os modelos são os objetos que representam informações contidas no domínio da aplicação [11]. Além disto, os modelos contêm a lógica de domínio ou negócio e, conceitualmente, não têm caráter visual para interface com o usuário. É a camada de *software* onde ocorre a abstração dos dados do sistema.

As visualizações comportam as lógicas de interface de uma aplicação, apenas [14]. No caso de uma aplicação para a *web*, é, geralmente, código escrito em *Hypertext Markup Language* (HTML) para a composição das páginas, frequentemente em conjunto com *Extensible Markup Language* (XML), *Cascading Style Sheets* (CSS) e *JavaScript*.

Os controladores, por sua vez, compõem a camada da aplicação que controla seu fluxo de dados. Determinam, também, qual ação deve ser tomada pela aplicação quando da ocorrência de requisições dos usuários. Os controladores podem ser o canal de comunicação entre os modelos e as visualizações, ou seja, a comunicação entre o conjunto de dados do domínio da aplicação e a interface com o usuário, mas também podem permitir a interação direta entre eles [12].

Engenheiros de *software* têm adotado este padrão porque ele oferece um método eficiente para a criação de aplicações com componentes fracamente acoplados, possibilitando maior facilidade no reaproveitamento destes componentes, assim como facilita também as possíveis manutenções da aplicação [4].

A Fig. 2 ilustra, de forma simplificada, o fluxo de interações entre as camadas de *software* no padrão MVC em aplicações para a *web*.

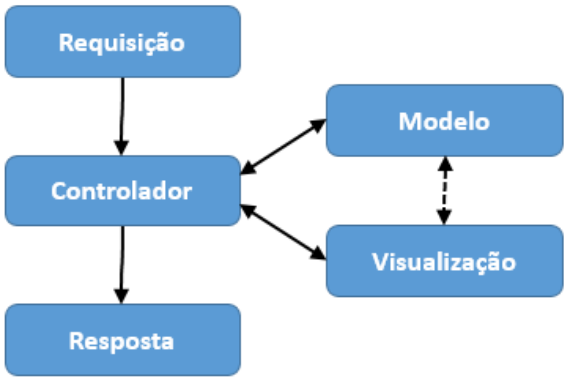

Fig. 2. Interação entre camadas de software no padrão MVC. Adaptação [14].

III. DESENVOLVIMENTO DO PROJETO DE REENGENHARIA ÁGIL

Sobre o sistema de *software* integrante do estudo de caso em foco nesta pesquisa, foi realizada uma atividade de identificação de módulos (ou "perfis" – conforme caracterizados em sua origem). Esta atividade, definida como a compreensão do estado atual de um sistema, é denominada análise de inventário [2].

Para fins de observação neste trabalho, o escopo do objeto em estudo foi delimitado em três módulos do sistema em projeto. Dentro deste número, foram considerados aqueles com nível mais alto de criticidade em relação às atividades do domínio do negócio, em acordo com os *stakeholders* da organização [15].

Nominalmente, portanto, além do núcleo do sistema, onde são tratadas autenticações, autorizações, gerenciamento de sessão de usuário, conexões com bancos de dados, entre outras funcionalidades básicas e inerentes a sistemas de informação, são os módulos de serviços disponíveis aos alunos de graduação, como visualização de notas e horários de aulas; serviços disponíveis aos professores acadêmicos, como informação do diário de classe; e os serviços públicos, como visualização de portarias.

Foi analisado que o núcleo do sistema de serviços acadêmicos deve abranger atividades genéricas inerentes a um sistema de informação. Basicamente, isto compreende responsabilidades como o controle de autenticação dos usuários e autorização de acesso aos recursos do sistema, gerenciamento de sessão de usuários, conexões com bancos de dados e determinação do *layout* principal da aplicação, entre outras.

Estas atividades são consideradas as mais críticas para o funcionamento adequado do sistema e, de acordo com as definições do *OpenUP* [9], suas produções devem compreender os primeiros incrementos e iterações do ciclo de desenvolvimento.

Examinando o sistema legado, compreendeu-se que havia um subsistema de permissões baseado em perfis, onde cada usuário teria acesso a diversos perfis, e cada perfil proporcionava acesso a diversas páginas comportadas por este [15]. O problema deste modelo de permissões de acesso fica aparente sob duas perspectivas: quando da necessidade de compartilhamento de recursos e também na abordagem técnica para a verificação de privilégios de acesso.

No caso da necessidade do compartilhamento de privilégios de acesso a determinado recurso do sistema por diversos usuários, isto dependeria da vinculação do usuário ao perfil sob o qual o recurso seria disponibilizado. Contudo, além de conceder acesso ao recurso solicitado, era automaticamente concedido acesso a todos os recursos disponibilizados pelo perfil. Isto, muitas vezes, não supriria a necessidade de maior granularidade nas definições de privilégios de acesso, expressa pela organização, e impossibilitava a atribuição de privilégios de acesso individuais aos usuários do sistema [15].

Na abordagem proposta, é possível o agrupamento lógico desses recursos por características em comum, mas passa a ser possível, em conjunto, a atribuição de permissões de acesso de forma individual aos usuários de forma mais específica.

Foi identificado no sistema legado, também, um problema na abordagem técnica utilizada para a verificação dos privilégios de acesso às páginas. No caso, a cada requisição executada pelo agente do usuário, seriam efetuadas múltiplas conexões simultâneas entre a aplicação e o banco de dados para o fim de executar a validação de acesso, apenas.

Este tipo de situação teria o potencial de causar lentidão e indisponibilidade do sistema quando em épocas de picos de acesso. Considerando ainda que, por se tratar de um sistema de informação, seriam estabelecidas outras conexões com fontes de dados com o objetivo de buscar os dados que, efetivamente, foram solicitados pelo usuário.

As propostas para o melhoramento nos aspectos de autenticação e permissões de acesso e da criação de conexões múltiplas com o banco de dados, baseado nos componentes providos pelo *Zend Framework* são apresentadas em detalhes, respectivamente, nas subseções A e B da presente seção.

A. *Autenticação e Autorização de Acesso*

Autenticação é definida como a determinação de veracidade de uma identificação perante a informação de credenciais [13]. Para este fim, o ZF provê uma *Application Programming Interface* (API) através do componente *Zend\Authentication*, para a implementação de casos de uso onde haja a necessidade de autenticação de usuários.

Este componente utiliza classes associadas a componentes do tipo *Zend\Db\Adapter* como dependência, para efeitos de autenticação. Esta, por sua vez, representa a abstração do acesso ao banco de dados, ou seja, uma conexão ativa pronta para o uso. No caso, especificamente, representaria o meio de acesso ao local onde estão contidas as informações de identificação dos usuários e suas credenciais (senhas).

O processo de autenticação pode retornar uma tentativa como válida ou inválida. Em ambos casos, são instanciados objetos contendo as informações de acessos permitidos, dado o contexto. Um usuário autenticado teria privilégios de acesso de aluno acadêmico ou professor, por exemplo. Um usuário não autenticado, por sua vez, teria privilégios de acesso

definidos pelo mesmo processo, porém, somente àqueles recursos definidos como para acesso público.

A permissão de acesso às seções do sistema aos usuários depende diretamente de sua autenticação perante este. Autorização é definida como a identificação de privilégios de acesso ao determinado recurso por determinada entidade [13].

Em ambas as situações, com o usuário autenticado ou não, as diretrizes de acesso são buscadas e encapsuladas por um objeto da classe *Zend\Permissions\Acl*, componente do ZF que implementa o padrão de projeto conhecido como *Access Control List* (ACL).

À cada requisição de acesso ao sistema, são verificados os privilégios de acesso definidos para o usuário requerente. Caso seja um usuário não autenticado, são atribuídos os privilégios de acesso aos recursos de livre acesso, apenas.

B. Gerenciamento de Serviços

Um dos padrões de projeto implementados no ZF é o da injeção de dependências [13]. Em relação ao problema da instanciação de múltiplas conexões com bancos de dados à cada requisição, a aplicação deste padrão permite que uma única conexão seja instanciada e utilizada por todas solicitações dentro de uma mesma requisição.

Na prática, dentro do escopo de cada requisição à aplicação, uma conexão com o banco de dados seria instanciada pelo gerenciador de serviços quando da primeira solicitação e, então, conservada por ele até o final da requisição. O gerenciador de serviços seria o responsável por determinar o uso das conexões (e outros serviços que venha a gerenciar), por injeção de dependência aos componentes que necessitem delas.

Desta forma, obedecendo a arquitetura proposta e o princípio da inversão de controle, elimina-se a instanciação simultânea de recursos idênticos e seu aproveitamento inadequado.

C. Interface e Layout da Aplicação

Em pesquisa realizada junto aos usuários do sistema legado [15], foi apurado que grande parte destes usuários tinha uma opinião negativa sobre a interface do sistema. Entre outras, algumas opiniões recorrentes davam referência a poluição visual na interface do sistema em função da apresentação de elementos desnecessários. Foi notada também a ausência de marca ou logotipo identificando a instituição aos usuários.

A aparência gráfica da interface do sistema legado é ilustrada na Fig. 3.

Fig. 3. Ilustração da interface do sistema legado.

O detalhamento dos requisitos de interface, conforme identificados para reengenharia, expressou [15]:

- A interface deve seguir um padrão em todas seções do sistema;
- Deve ser facilitado o acesso pelo usuário comum que, efetivamente, utiliza o sistema, com uma interface simples e amigável;
- A linguagem disposta deve ser orientada ao usuário, com acesso intuitivo e propôr eficiência na recuperação de informações;
- O usuário deve controlar todas suas ações no sistema;
- O usuário deve receber *feedback* sobre suas ações no sistema.

Com base nesta abordagem, foi proposta a reestruturação do *layout* principal do sistema, tendo em vista o desacoplamento entre a interface e as outras camadas do *software*, em conformidade com o padrão MVC.

Foi proposta uma estrutura bastante simples e limpa, com o objetivo de evitar poluição visual, e foi aproveitada a identificação visual da instituição em duas oportunidades: na ilustração do brasão extraoficial da instituição no canto superior esquerdo da interface e na utilização do logotipo histórico da instituição como plano de fundo à interface (ver Fig. 4).

Além disto, um problema enfrentado pelos usuários era a difícil acessibilidade aos recursos do sistema devido ao sistema de perfis, conforme descrito anteriormente. Para solucionar este problema de navegabilidade, foi proposta uma mudança no método de acesso a eles. Diferentemente do proposto no sistema legado, na nova interface não há seleção de perfis de acesso anteriormente ao acesso propriamente dito. Desta maneira, os recursos disponíveis são apresentados todos juntos em formato de lista ao usuário na interface.

Neste contexto, o foi utilizado o componente *Zend\Navigation* do ZF que, em associação com um objeto do tipo *Zend\Permissions\Acl*, é capaz de produzir uma lista de recursos, de acordo com as permissões de acesso ao usuário como um menu de acesso ou mapa do sistema, para integração na interface.

Com o uso da linguagem *JavaScript* e a possibilidade de manipulação sob perspectiva *Document Object Model* (DOM), foi desenvolvido, como parte do estudo neste trabalho, um artefato capaz de filtrar por palavras-chave os recursos presentes na lista descrita acima. Portanto, um usuário, ao confrontar-se com diversos itens representantes de recursos para utilização do sistema, poderia informar uma expressão relevante ao contexto do recurso solicitado. Assim, a lista apresentaria um comportamento de auto filtragem, mostrando somente os recursos nos quais seja determinada relevância, de acordo com o termo inserido pelo usuário.

Um protótipo de interface para a abordagem proposta é apresentado na Fig. 4.

D. Modelagem e Documentação do Projeto

Conforme exposto anteriormente, a proposta do *OpenUP*, no que se trata da extensibilidade do processo de desenvolvimento, permite boa flexibilidade na especificação

das suas atividades. Esta, em conjunto com a ideia de minimização de documentação, implica na possibilidade da determinação dos artefatos que são essenciais, dado o contexto de cada projeto.

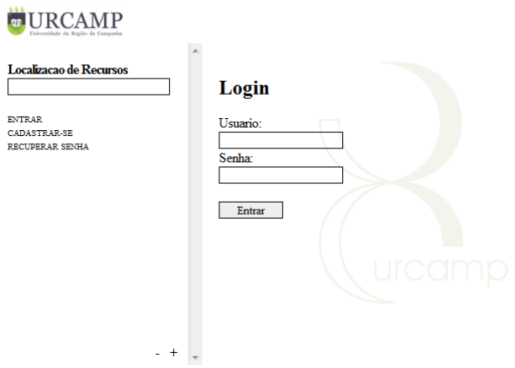

Fig. 4. Protótipo de interface em acordo com o contexto proposto.

A *Unified Modeling Language* (UML) é uma linguagem visual utilizada para modelar sistemas computacionais orientados a objetos [16]. Esta linguagem disponibiliza diversos elementos gráficos para utilização na modelagem de sistemas, e se tornou, ao longo dos anos, a linguagem padrão para a modelagem e documentação de *software*, sendo adotada internacionalmente pela indústria de ES.

Uma das principais características da metodologia *OpenUP* é que as atividades desempenhadas no desenvolvimento são direcionadas aos casos de uso do sistema [9]. Portanto, entende-se que a formulação correta desta especificação implica diretamente no bom andamento e sucesso de um projeto.

Neste contexto, a UML disponibiliza notações para a formulação do diagrama de casos de uso, componente do modelo de casos de uso (MCU) de um sistema. A criação deste modelo faz parte da atividade de análise e apresenta uma representação das funcionalidades observáveis externamente a um sistema, assim como os elementos externos que interagem com este [6]. Entende-se que o MCU apresenta, essencialmente, a perspectiva da figura do usuário sobre como ele age perante o sistema.

A Fig. 5 ilustra um exemplo de modelo de casos de uso correspondente as ações realizadas por usuários do sistema no estudo de caso presente neste estudo. O exemplo denota como alunos acadêmicos visualizam notas, horários e histórico acadêmico; professores informam diários de classe e; todos usuários têm acesso à visualização de portarias e solicitação de credenciais de acesso. Também fica caracterizado o requisito de alunos e professores passarem por autenticação no sistema antes de realizarem suas atividades.

Citado como o diagrama da UML mais utilizado na modelagem de sistemas de *software* orientados a objetos [17] e, potencialmente, com maior relevância na expressão de informações relevantes e comunicação de ideias durante o desenvolvimento [18], o Modelo de Classes é representado pelo Diagrama de Classes [16].

A Fig. 6 apresenta um exemplo da diagramação das classes

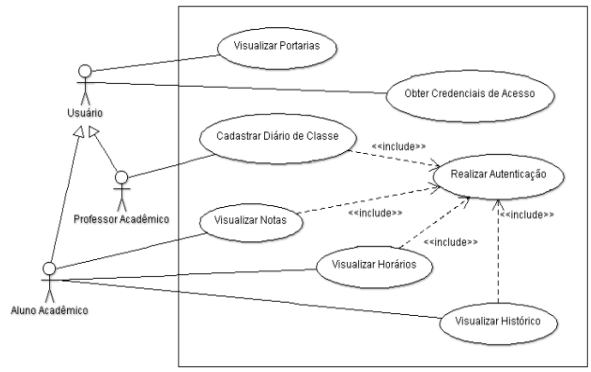

Fig. 5. Exemplo de diagrama utilizado na modelagem de casos de uso.

em estágio de especificação projetada de acordo com as necessidades de uma seção da aplicação no presente estudo de caso deste trabalho. Esta ilustração apresenta classes referentes ao domínio do diário acadêmico e indica como, dentro da arquitetura MVC, o controlador pode manipular o modelo (com as regras de negócio) e a visualização (para fazer interface com o usuário). Também é expresso como um diário de classe é composto por uma oferta de disciplina que, por sua vez, é composta pela disciplina e pelo professor ministrante, além de atributos próprios da classe.

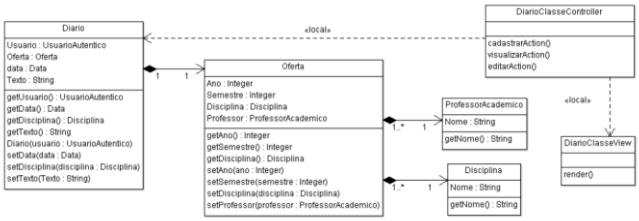

Fig. 6. Exemplo de diagrama utilizado na modelagem classes.

IV. CONCLUSÃO

Como resultado deste estudo, foi verificado que, com a aplicação de uma metodologia de desenvolvimento de *software* em um ambiente onde a prática anterior era considerada indisciplinada, foram positivamente alterados indicadores de qualidade de *software* anteriormente negligenciados, como o entendimento do sistema, em virtude da documentação produzida, e sua manutenibilidade, devido à proposição de uma estrutura de *software* bem definida (arquitetura MVC).

A adoção da metodologia ágil *OpenUP* de desenvolvimento de *software* em um estudo de caso de reengenharia permitiu uma abordagem minimalista na determinação de quais documentos deveriam ser produzidos durante o processo e, desta maneira, foi possível observar a prática satisfatória da documentação do sistema sem grandes interferências nas estimativas de entrega dos produtos.

Verificou-se, também, o impacto positivo da aplicação do *Zend Framework* na estrutura do sistema, pela utilização de padrões de projeto implementados em seus artefatos. Foi observado o melhoramento na produtividade dos

desenvolvedores pela alta reusabilidade de componentes oferecida pelo *framework*, o que implicou na desnecessidade do projeto, especificação e implementação de componentes básicos que compõem o núcleo de um sistema de informação.

Finalmente, a execução do projeto decrito no presente trabalho teve, como consequência, a eliminação de diversos problemas de manutenibilidade apresentados pelo sistema legado.

Com base nos resultados obtidos e considerando que produtos de *software* não são considerados soluções permanentes a qualquer que seja a atividade a qual sirvam de suporte, foi possível concluir que a adoção de um processo de *software* ágil, compatível com o contexto o qual esteja inserido, se torna essencial para o desenvolvimento de um produto desta natureza, o que converge com a proposta de minimalismo, completude e extensibilidade da metodologia *OpenUP* (ver seção II subseção C).

Não menos importante, a proporção em larga escala de reusabilidade de componentes e a inibição de atividades de um processo consideradas demasiadamente burocráticas vêm a prolongar e sustentar a manutenibilidade de produtos de *software*, de maneira que seus ciclos de desenvolvimento e manutenção, mesmo em reengenharia, possam ser desempenhados com agilidade e qualidade devida para a continuidade na satisfação dos requisitos dos usuários.

REFERÊNCIAS

[1] IEEE. (1998) "IEEE Standard for Software Maintenance". IEEE.
[2] Pressman, R. (2010) "Software Engineering: A Practitioner's Approach". 7.ed, New York, McGraw-Hill.
[3] Sommerville, I. (2007) "Software Engineering". 8.ed, Pearson.
[4] Padilla, A. (2009) "Beginning Zend Framework", New York, Apress.
[5] IEEE. (2014) "Guide to the Software Engineering Body of Knowledge". IEEE.
[6] Bezerra, E. (2007) "Princípios de Análise e Projeto de Sistemas com UML". 2.ed, Rio de Janeiro, Elsevier.
[7] Beck, K. et al. (2001) "Manifesto for Agile Software Development". http://www.agilemanifesto.org.
[8] Kroll, P. (2011) "OpenUP In a Nutshell". http://www.ibm.com/developerworks/rational/library/sep07/kroll/.
[9] Eclipse Foundation. (2014) "OpenUP". http://epf.eclipse.org/wikis/openup/.
[10] Santos, S. (2009) "OpenUP: Um Processo Ágil". http://www.ibm.com/developerworks/br/local/rational/open_up/.
[11] Waswani, V. (2010) "Zend Framework: A Beginner's Guide", New York, McGraw-Hill.
[12] Lisboa, F. (2009) "Zend Framework: Componentes Poderosos para PHP", São Paulo, Novatec.
[13] Zend Technologies. (2014) "Programmer's Reference Guide". http://framework.zend.com/manual/en/.
[14] Allen, R.; Lo, N.; Brown, S. (2009) "Zend Framework In Action". Greenwich, Manning.
[15] De Lima, D. A. (2011) "Portal Coorporativo em Universidades: uma Metodologia para o Processo de Projeto". Universidade Federal do Rio Grande do Sul.
[16] Guedes, G. (2007) "UML 2: Guia Prático", São Paulo, Novatec.
[17] Forward, A. e Lethbridge, T. (2008) "Perceptions of Software Modeling: A Survey of Software Practicioners". University of Ottawa, Canada.
[18] Reggio G. et al. (2013) "What Are the Used UML Diagrams? A Preliminary Survey". Università di Genova, Italy.