UNIVERSIDAD MAYOR DE SAN SIMÓN

FACULTAD DE CIENCIAS Y TECNOLOGÍA

TRABAJO DE LICENCIATURA

METODOLOGÍA ÁGIL MODELING

Caso de Estudio

Revista de Divulgación Científica de Ciencias

AUTOR: Vasco Daniela Pedro

Mayo, Año de 2007

UNIVERSIDADE EDUARDO MONDLANE

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE MATEMÁTICA E INFORMÁTICA

Trabalho de Licenciatura

METODOLOGIA AGILE MODELING

Caso estudo

Emissão de Declarações e Certificados da UEM

AUTOR: Vasco, Penicela Pedro

SUPERVISORA: Prof. Dra Esselina Macome

Maputo, Maio de 2007

DEDICATÓRIA

> ***"A Ciência é um sistema de conhecimentos que abarca verdades gerais ou a operação de leis gerais especialmente obtidas e testadas através de métodos científicos"*** (Wikipédia, 2006c).

Dedico este trabalho:

Aos meus pais, que tanto os amo como os admiro pela força e confiança que em mim depositaram;

Aos meus irmãos que durante toda a minha carreira académica me acompanharam, esclarecendo-me as minhas dúvidas no momento oportuno;

Aos meus queridos sobrinhos, pois destes tive um aconchego merecido;

Aos meus amigos que souberam, durante toda a minha vida, acompanhar-me nos pequenos e grandes momentos;

À minha querida namorada, Elda, que me deu muita força, amor e carinho para que este trabalho se realizasse;

Aos meus colegas de turma e de curso, que sempre estiveram disponíveis para me apoiar no entendimento das matérias durante o curso.

AGRADECIMENTOS

Agradeço à Deus, por ter me dado coragem de enfrentar o mundo da ciência e tecnologia;

Agradeço à minha família, em especial aos meus pais, orgulho e razão da minha existência e que nenhuma palavra seria suficiente para dizer o quanto lhes sou grato pelo apoio, ensinamentos e sabedoria que me transmitem, e aos meus irmãos que me ensinaram bastante e deram-me um puxão de orelhas no devido momento;

O meu muito obrigado à minha supervisora, Prof. Doutora Esselina Macome pela paciência, atenção, críticas, ensinamentos e direccionamento ao longo do trabalho;

Agradeço bastante ao engenheiro Cameron Smith, que me ensinou a programar em *JSP* e despertou em mim a capacidade de investigação de novas tecnologias aliadas a *java,* e ao Ivan que me ajudou imenso com o desenho das imagens da página desenvolvida;

Agradeço aos meus colegas do DMI e amigos pelo imenso carinho que me demonstraram e pela ajuda na concretização do trabalho, especialmente ao Shang pelas críticas e ajuda no início do trabalho, ao Azarais, Rossana e Hélder pela força que me deram;

Agradeço aos funcionários da Direcção do Registo Académico (DRA), nomeadamente ao Frederico Guirugo, Sérgio Bata e Ivan Conninfon por toda ajuda ao longo deste trabalho e a todos os funcionários do DMI, pelo apoio e carinho que nunca me faltaram durante o curso;

Agradeço à minha querida namorada, Elda, que me deu todo o seu carinho e me apoiou bastante na concretização deste trabalho;

Um profundo agradecimento vai à todos os estudantes do DMI, em particular aos que dedicarem parte do seu tempo para a leitura deste trabalho. Espero sinceramente contribuir de alguma forma com os conhecimentos e experiências expostas.

DECLARAÇÃO DE HONRA

Declaro por minha honra, que este trabalho é fruto da minha investigação e nunca foi apresentado por nenhuma outra pessoa, e que o mesmo foi realizado para ser submetido para obtenção do grau de Licenciatura em Informática na Universidade Eduardo Mondlane.

Maputo, Maio de 2007

Penicela Pedro Vasco

Penicela Pedro Vasco

RESUMO

Nos princípios desta década, com o surgimento das metodologias ágeis, houve muitas mudanças no processo de desenvolvimento de software. O foco principal destas metodologias está no aumento da qualidade do software desenvolvido e a garantia da satisfação dos requisitos do cliente.

O uso destas metodologias tornou-se uma necessidade, pois elas trazem técnicas que visam garantir maior qualidade no software e, por conseguinte, maior satisfação do cliente. Em Moçambique, assim como em vários pontos do mundo, nota-se que a qualidade do software desenvolvido não tem alcançado a qualidade desejada pelo cliente, e em muitos casos não satisfaz os requisitos dos seus utilizadores.

Agile Modeling é uma destas metodologias. Ela é baseada em valores, princípios e práticas que visam o desenvolvimento ágil de software, garantindo desta forma, qualidade e entrega atempada do software produzido.

Para a consolidação dos conhecimentos desta metodologia, procurou-se aplicá-la no desenvolvimento do Sistema de Emissão de Declarações e Certificados da UEM, cuja escolha vem no âmbito da DRA, gerir os seus processos manualmente, o que tem trazido enormes transtornos, sobretudo aos estudantes que levam 10 dias para a aquisição de um documento que poderia ser emitido muito pouco tempo.

É neste contexto, que se propõe este sistema para a emissão de declarações e certificados, construído em ambiente *web*, de forma que um estudante possa requisitar os documentos a partir de qualquer ponto do mundo.

Para o efeito, usou-se uma modelação orientada a objectos, modelando em *UML*, e para o desenvolvimento da aplicação, usou-se a linguagem JSP e a base de dados em *MySQL*, e tratando-se de um sistema *web*, aborda-se também, ao longo da descrição do trabalho aspectos de segurança de modo a garantir a integridade e segurança dos dados.

SUMÁRIO

ABREVIATURAS **5**

GLOSSÁRIO **6**

1. INTRODUÇÃO **9**

1.1 Enquadramento do trabalho 9

1.2 Definição do Problema 12

1.3 Objectivos 13

1.3.1 Gerais 13

1.3.2 Específicos 13

1.4 Metodologia 13

1.5 Estrutura do trabalho 16

2. A METODOLOGIA AGILE MODELING **17**

2.1 AM realça outros processos de software 18

2.2 Motivações para a criação da metodologia AM 20

2.3 Definição de modelos ágeis 20

2.4 Valores da AM 22

2.4.1 Comunicação 23

2.4.2 Simplicidade 23

2.4.3 Feedback 23

2.4.4 Coragem 24

2.4.5 Humildade 24

2.5 Princípios da AM 25

2.5.1 Princípios fundamentais ou centrais 25

2.5.2 Princípios suplementares 30

2.6 Práticas da AM 31

2.6.1 Práticas fundamentais ou centrais 31

2.6.2 Práticas suplementares 37

2.7 Como é que as práticas de AM se encaixam? 40

2.7.1 As práticas fundamentais 40

2.7.2 As práticas suplementares 42

2.7.3 Como é que as categorias se relacionam 42

2.8 Documentação Ágil 45
2.8.1 Relação entre modelos documentos e código fonte 46
2.8.2 Razões para a criação de documentos 46
2.8.3 Quando é que um modelo se torna permanente 47
2.8.4 Quando é que um documento é Ágil? 48
2.8.5 Tipos de documentos a criar no seu projecto 49
2.8.6 Estratégias para aumentar a agilidade da documentação 50
3. O SISTEMA DE EMISSÃO DE DECLARAÇÕES E CERTIFICADOS 52
3.1 Sistema Actual 52
3.1.1 Descrição do sistema actual 52
3.2 Sistema Proposto 54
3.2.1 Acesso ao sistema 54
3.2.2 Boas razões para adopção deste sistema 55
4. MODELAÇÃO DO SISTEMA USANDO AM 56
4.1 RationalUnified Process 56
4.1.1 Fases do RUP 57
4.1.2 Princípios do RUP 58
4.2 Âmbito do SEDC 58
4.3 Requisitos do SEDC 58
4.3.1 Requisitos Funcionais 59
4.3.2 Requisitos Não funcionais 59
4.4 Diagrama de casos de uso 61
4.4.1 Casos de uso e actores 62
4.5 Diagrama de classes 63
4.6 Diagrama de Actividades 65
4.7 Diagrama de Interacção 66
4.8 Diagrama de estados 66
4.9 Desenho de Base de Dados 66
4.10 Arquitectura de Implementação Usada 67
5. CRITÉRIOS DE SEGURANÇA 72
5.1 Critérios de segurança implementados 72
5.2 Critérios de segurança por implementar 73

6. CONCLUSÕES E RECOMENDAÇÕES 74

6.1 Conclusões 74

6.2 Recomendações 76

7. BIBLIOGRAFIA 78

ANEXO A — O Manifesto para o Desenvolvimento Ágil de Software 84

ANEXO B — Fases do RUP 87

ANEXO C — Detalhamento de casos de uso 88

ANEXO D — Diagrama de actividades do pedido e emissão de documentos 107

ANEXO E — Diagrama de sequência do SEDC 108

ANEXO F — Diagrama de estados do objecto Pedido 110

ANEXO G — Algumas páginas do SEDC 111

LISTA DE FIGURAS E TABELAS

Figura 1 – AM realça outros processos de *software* (Fonte: Ambler, 2005).....19
Figura 2 – Como é que as práticas de AM se encaixam (Fonte: Ambler, 2006e).....40
Figura 3 – A relação entre Modelo, Documento, Código Fonte e Documentação (Fonte: Ambler, 2006f).....46
Figura 4 – Arquitectura geral da RUP (Fonte: Luiz, 2004).....57
Figura 5 – Diagrama de casos de uso do SEDC.....63
Figura 6 – Diagrama de classes do SEDC.....65
Figura 7 – Modelo Relacional de Base de Dados com Mapeamento de Tabelas.....67
Figura 8 – O uso de struts e spring na arquitectura MVC.....70
Figura 9 – Componentes arquitecturais e funcionamento do SEDC.....71
Figura 10 – Processo de pedido e emissão de um documento.....107
Figura 11 – Diagrama e sequência do processo de pedido de um documento.....108
Figura 12 – Diagrama de sequência do processo de emissão de um documento.....109
Figura 13 – Diagrama de estados do objecto pedido.....110
Figura 14 – Página de Login do SEDC.....112
Figura 15 – Opções relativas aos funcionários da reitoria.....113
Figura 16 – Página para o pedido de documentos.....114

Tabela 1 – Abreviaturas.....5
Tabela 2 – Glossário.....8
Tabela 3 – Documentos potenciais a serem criados pela sua equipa de desenvolvimento (Fonte: Ambler, 2006f).....50
Tabela 4 – Representação da Relação caso de uso – actor.....62

ABREVIATURAS

Elemento	Descrição
AM	Agile Modeling
API	Application Programming Interface
CRC Cards	Class Responsibility Collaboration Cards
CGI	Common Gateway Interface
CVS	Concurrent Versions System
DRA	Direcção do Registo Académico da UEM
DSDM	Dynamic Systems Development Method
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JDBC	Java Database Conector
JSP	Java Server Pages
JSTL	JavaServer Pages Standard Tag Library
MVC	Model View Controler
TI	Tecnologia de Informação
XML	Extensible Markup Language
XP	Extreme Programming
OO	Object Oriented
RUP	Rational Unified Process
SQL	Structured Query Language
SSL	Secure Socket Layer
TCP/IP	Transmission Control Protocol/Internet Protocol
UEM	Universidade Eduardo Mondlane
UI	User Interface
UML	Unified Modeling Language
SISTAFE	Sistema administração Financeira do Estado
URL	Universal Resource Language

Tabela 1 – Abreviaturas

GLOSSÁRIO

Elemento	Descrição
Acegi Security	é uma solução flexível e ponderosa para "*enterprise software*", com uma particular ênfase para aplicações que usam *spring*. O uso de *Acegi Security* provê a sua aplicação com uma autenticação compreensiva, controle de acesso baseado no instante, segurança do canal. (Acegi, 2006).
Agilidade	É a habilidade de criar e responder as mudanças de modo a lucrar num ambiente de negócio agitado (Highsmith, 2002)
Container web	O J2EE define o container web como ambiente para execução de aplicações web em Java. Um *container web* executa Servlets, classes Java cujo ciclo de vida é gerido pelo *container*, obtendo assim vantagens em escalabilidade e performance. (LOZANO, 2003).
CVS	Sistema de Versões Concorrentes é um sistema de controle de versão que permite que se trabalhe com diversas versões de arquivos organizados em um directório e localizados local ou remotamente, mantendo-se suas versões antigas e os *logs* de quem e quando manipulou os arquivos. É especialmente útil para se controlar versões de um software durante seu desenvolvimento, ou para composição colaborativa de um documento. (Wikipedia, 2006b).
eXtreme Programming	Programação eXtrema, ou simplesmente *XP*, é uma metodologia ágil para equipas pequenas e médias e que irão desenvolver software com requisitos vagos e em constante mudança. Para isso, adopta a estratégia de constante acompanhamento e realização de vários pequenos ajustes durante o desenvolvimento de software. (Wikipedia, 2006b).
Framework	É uma estrutura de suporte definida em que um outro projecto do software pode ser organizado e desenvolvido. Tipicamente, um *framework* pode incluir programas de apoio, bibliotecas de código, linguagens de *script* e outros softwares para ajudar a desenvolver e juntar diferentes componentes do seu projecto. (Wikipedia, 2006b).
HTML	É um acrónimo para a expressão inglesa *HyperText Markup Language*, que significa Linguagem de Formatação de Hiper texto, é uma linguagem de marcação utilizada para produzir páginas na *web*. (Wikipedia, 2006b).
Java EE/ J2EE	É uma plataforma de programação de computadores que faz parte da plataforma *Java*, voltada para aplicações multi-camadas, baseadas em componentes que são executados em um

	servidor de aplicações. (Wikipédia, 2006a).
JavaServer Pages (JSP)	É uma tecnologia para desenvolvimento de aplicações *web* criada pela *sun*[1], que permite a criação de páginas dinâmicas. Como uma parte da tecnologia java, permite o desenvolvimento rápido de aplicações *web* que são de plataforma independente. (Sun, 2006).
JSTL	*JavaServer Pages Standard Tag Library* é uma colecção de bibliotecas de *tags* customizadas que implementam funcionalidades gerais, comuns a aplicações *web*, incluindo iteração e condição, formatação de dados, manipulação de *Extensible Markup Language* (XML) e acesso a base de dados. (Guapo, 2006).
Linguagem de marcação	É um conjunto de códigos aplicados a um texto ou a dados, com o fim de adicionar informações particulares sobre esse texto ou dado, ou sobre trechos específicos. (Wikipedia, 2006b).
Servlet	É um componente que disponibiliza ao programador da linguagem *Java* uma interface para o servidor web (ou servidor de aplicação), através de uma API. As aplicações baseadas no *Servlet* geram conteúdo dinâmico (normalmente HTML) e interagem com os clientes, utilizando o modelo *request/response*. Os *servlets* normalmente utilizam o protocolo HTTP, apesar de não serem restritos a ele. Um *Servlet* necessita de um *container web* para ser executado. (Wikipedia, 2006b).
Scriplet	É um *script* independente que é referenciado de uma página HTML e representa um objecto que aceita dados de fora e suporta várias operações no dado. (Shiran, 2000).
SCRUM	É um método ágil para gestão de projectos. Ele é documentado para melhorar a produtividade dinamicamente em equipas previamente paralisadas por metodologias mais pesadas. (Wikipedia, 2006b).
Stakeholder	É qualquer um que seja utilizador directo/indirecto, gestor de utilizadores, gestor sénior, pessoal do nível operativo, pessoal do suporte, desenvolvedores que trabalham no outro sistema que integra ou interage com o software em desenvolvimento, ou profissionais de manutenção afectados pelo desenvolvimento de um projecto de software. (Ambler, 2006a).
Struts	É um *framework* de desenvolvimento da camada controladora, numa estrutura seguindo o padrão MVC-2, de aplicações *web* (principalmente) construído em *Java* para ser utilizado em um *container web* em um servidor J2EE. (Wikipedia, 2006b).
Springframework	É um *framework open source* criado por Rod Johnson e descrito

[1] http://www.sun.com/

	em seu livro "*Expert One-on-One: J2EE Design e Development*". Ele foi criado para endereçar a complexidade de desenvolvimento de aplicativos corporativos. (Wikipedia, 2006b).
Tags	São estruturas de linguagem de marcação que consistem em breves instruções, tendo uma marca de início e outra de fim. (Wikipedia, 2006b).

Tabela 2 – Glossário

1. INTRODUÇÃO

1.1 Enquadramento do trabalho

Não existe dúvida alguma que o mundo está a tornar-se, a cada dia, mais dependente de software. De facto, muitas inovações das quais nós tomamos proveito hoje, como por exemplo o Amazon[2], simplesmente não existiriam se não existisse software (Eeles, 2006).

Para que estas inovações sobrevivam, o software do qual elas dependem deve ser capaz de responder às necessidades do utilizador, ter qualidade suficiente, estar disponível quando prometido e desenvolvido a um preço aceitável.

Em Moçambique, muitas instituições, privadas e públicas têm-se mostrado preocupadas em adoptar tecnologias baseadas no computador. A título de exemplo, temos:

- O Conselho Municipal da Cidade de Maputo que em Janeiro de 2005 optou por um sistema para a gestão de resíduos sólidos urbanos, totalmente informatizado, para a gestão do processo de licenciamento e pagamento das taxas de limpeza (MUCHANGA, 2005:12);
- O centro de saúde de Polana Caniço contratou uma empresa moçambicana para o desenvolvimento de um sistema de gestão de recursos humanos e salários, nos meados do mês de Junho de 2006 (TEMBE, 2006);
- No início desta década, o Ministério das Finanças, criou uma unidade técnica, para o desenvolvimento do modelo electrónico do Sistema de Administração Financeira do Estado, denominado *e-SISTAFE*.

Mas, de um modo geral, o processo de desenvolvimento de software tem reflectido uma qualidade muito abaixo do que seria considerado ideal. E, normalmente, os sistemas são entregues aos clientes fora do prazo estipulado no projecto preliminar, e com um custo maior do que o previsto. Além disso, os sistemas, muitas vezes, não satisfazem as reais

[2] Amazon – Site de venda de vários produtos através da internet

necessidades dos usuários (devido a fraca capacidade do utilizador, de expor os seus requisitos), necessitando ser desenvolvidos de novo, num processo conhecido por muitos profissionais como "isso terá de ficar para uma próxima versão" (Santos, 2003:1).

Por exemplo, o Sistema de Gestão de Resíduos Sólidos Urbanos, que deveria ter sido concluído e entregue até Junho de 2005, só foi concluído em Agosto de 2005. Já foi sujeito a várias correcções desde lá (mais de 10 versões), mas até hoje continua não satisfazendo na íntegra as necessidades do utilizador: Isto deve-se ao facto do Conselho Municipal não ter conseguido especificar, com exactidão, os seus requisitos e também por se tratar de um sistema novo.

Na perspectiva de resolver os problemas semelhantes aos acima descritos, surge a necessidade de procurar metodologias que visem um desenvolvimento ágil do *software,* de forma a garantir que, mesmo com o aparecimento tardio de requisitos do sistema, seja possível a sua incorporação no mesmo, e, por conseguinte, a satisfação do utilizador. Uma metodologia que aborda estes pontos, conhecida internacionalmente e com grandes resultados é a metodologia *Agile Modeling* (AM), um processo de desenvolvimento de *software* baseado em práticas que visa aumentar a eficiência da equipa dentro de um projecto de desenvolvimento de *software*.

Na Internet, já se encontram vários artigos de programadores que afirmam ter alcançado bons resultados com o uso de metodologias ágeis de desenvolvimento. É o caso de Agsilva (2006) que escreveu um artigo chamado APGR (*Agile Project - Good Result*), que não traz nada de novo, simplesmente realça as práticas já conhecidas de AM e XP.

Em Moçambique, o uso das metodologias ágeis, é já uma realidade. A título de exemplo temos a Unidade Técnica de Reforma Administrativa e Financeira do Estado (UTRAFE), que já usa uma abordagem ágil no desenvolvimento do sistema *e-Sistafe* e dos outros sistemas, usando as boas práticas da AM.

Para a criação desta metodologia, Santos (2003), no seu trabalho ***Agile Modeling – Overview***, cita duas motivações principais:

- O objectivo primário de um projecto de software é o próprio software, e não um grande conjunto de documentação sobre ele;
- Um artefacto é criado primordialmente para permitir a comunicação e a troca de informações entre a equipa e permitir a discussão e refinamento do modelo do sistema. Assim, se um artefacto não passa informação relevante ao projecto, ele não cumpre seu objectivo básico.

Baseando-se nestas constatações um grupo de dezassete pesquisadores esteve reunidos, de 11 a 13 de Fevereiro de 2001, num alojamento nas montanhas de *Wasatch* em Utá (uma cidade de Estados Unidos da América) para reflectir, trocar ideias, esquiar e relaxar e, como resultado, foi definido um manifesto e doze princípios para o desenvolvimento ágil de software guiados por quatro (4) valores a serem aplicados no processo de desenvolvimento, para que se tornem mais interactivos e menos burocrático do que os actuais (BECK, 2001a).

Para mais detalhes dos valores e princípios da Aliança Ágil, veja o anexo A.

Para a melhor compreensão e teste da aplicabilidade desta metodologia, surge a necessidade de aplica-la num caso de estudo. Neste trabalho, o caso de estudo é o Sistema de Emissão de Declarações e Certificados da UEM (SEDC).

O SEDC faz o registo das requisições das declarações, colhe os dados a nível das faculdades e elabora o documento requisitado. A escolha deste sistema vem na óptica da adopção de Tecnologias de Informação (TI), por parte da maioria das instituições nacionais e da fragilidade que este sistema possui (um erro numa declaração/certificado é muito comprometedor).

A Universidade Eduardo Mondlane (UEM), a maior instituição de ensino superior do país possui 13 faculdades as quais leccionam actualmente cerca de 42 cursos distribuídos entre as diversas faculdades (UEM, 2006)

A Direcção do Registo Académico da UEM (DRA), é o órgão responsável pela coordenação entre as diferentes faculdades desta Universidade bem como pela emissão

de certificados, diplomas, declarações, entre outros documentos. Entretanto, para a obtenção de qualquer um dos documentos acima citados, o interessado deve dirigir-se à DRA, onde preenche um formulário ou redige um requerimento dirigido ao reitor da UEM, consoante for o documento a solicitar. O solicitante deverá, então, para tal, fazer um pagamento correspondente ao pedido e aguarda durante 10 dias para a sua obtenção, podendo, em alguns casos, este período se prolongar, dependendo de vários factores, como, por exemplo, a demora no envio das notas à DRA por parte da faculdade do solicitante.

É na perspectiva de melhorar o sistema de emissão das declarações e certificados, permitindo a não demora e veracidade dos dados que constem no documento, que se pretende, com este trabalho, desenvolver um Modelo de Emissão de Declarações e Certificados usando *Agile Modeling*.

1.2 Definição do Problema

A DRA possui um sistema manual de emissão de declarações e certificados, que leva um período de 10 dias para a obtenção dos mesmos, podendo, em alguns casos, ser prolongado.

Para a emissão de qualquer um dos documentos acima mencionados, a DRA envia uma carta à faculdade do indivíduo solicitando as suas notas. Este processo leva muito tempo, visto que a DRA não possui, neste momento, mecanismos que possibilitam a rápida busca de informação nas faculdades.

A DRA tem recebido constantemente reclamações dos estudantes devido a demora que estes documentos levam para serem emitidos, bem como de erros que têm constado no/a certificado/ declaração, levando a um prolongamento do tempo de espera, podendo atingir 15 dias. Estes erros vão desde a mal escrita do nome do estudante até ao curso, sendo este último o mais grave. Em alguns casos é emitida uma declaração que indica, por exemplo, que o estudante frequenta o curso de Direito quando ele frequenta o curso de Informática.

Em 2004, eu solicitei, à DRA, *uma declaração de nível para efeitos de emprego e depois de 10 dias recebi uma declaração que indicava que eu tinha concluído em Junho de 2003 o terceiro ano de Direito enquanto eu sou estudante de informática "Penicela Pedro Vasco"*

1.3 Objectivos

1.3.1 Gerais

Este trabalho apresenta como objectivos gerais:

- Estudar a metodologia *Agile Modeling* e apurar como esta pode ser aplicada no desenvolvimento de software, tendo em conta as realidades de Moçambique;
- Aplicar os princípios fundamentais da AM no desenvolvimento do Modelo de Emissão de Certificados e Declarações, usando tecnologias de análise e modelação orientada a objectos em ambiente WEB.

1.3.2 Específicos

- Avaliar a aplicabilidade da metodologia *Agile Modeling* e identificar as suas vantagens e desvantagens;
- Analisar os problemas do sistema actual que rege o departamento de Eissão de Dclarações e Certificados da DRA;
- Conceber e testar o Modelo de Emissão de Declarações e Certificados da UEM usando *Agile Modeling.*

1.4 Metodologia

Para avaliar a aplicabilidade da metodologia Agile Modeling e identificar as suas vantagens e desvantagens foi necessário:

- Pesquisa bibliográfica (consulta de livros, e Internet);

- Troca de emails com *Scott W. Ambler*[3] (um dos primeiros a escrever sobre esta metodologia);
- Entrevista informal ao Eng. Cameron Smith.

Para Analisar os problemas do sistema actual que rege o Departamento de Emissão de Declarações e Certificados da DRA, recorreu-se a:

- Análise do sistema actual na base da documentação disponível na DRA;
- Entrevista ao pessoal da reitoria, nomeadamente *Frederico Guirugo* (nos dias 10/10/2005 e 27/10/2006), *Sérgio Bata* (nos dias 12/10/2006 e 03/11/2006), *Ivan Conninfon* (no dia 03/11/2006) e Sr Armando (no dia 26 de Fevereiro de 2006) com objectivo de envolvê-los no processo de forma a obter informação que possa garantir que o modelo a implementar siga os padrões de funcionamento da instituição.

Para conceber o modelo de emissão de certificados e declarações usando Agile Modeling:

- Procurou-se usar, à medida do possível, os princípios e práticas da AM durante o processo de captação de requisitos.

Para desenhar e implementar o modelo de emissão de certificados e declarações:

- Fez-se a análise e desenhos do modelo de Emissão de Declarações e Certificados, usando uma linguagem de Análise Orientada à Objectos, tendo como base a *Unified Modeling Language* (UML) e *Agile Modeling* (AM) e usando como ferramenta de desenho o *Enterprise Architect 6.0*;
- A base de dados foi feita usando o Sistema de Gestão de Base de Dados (SGBD) - *My Structured Query Language* (MySQL 5.0), um produto *open source* e pôde ser adquirido gratuitamente na Internet, e para a visualização, em ambiente gráfico usou-se o *dbvisualizer*[4];

[3] www.ambysoft.com
[4] ***Dbvisualizer*** – uma feramenta open source que transforma os scripts de criação de tabelas em diagrama.

- Para o desenho da interface WEB usou-se o *JavaServer Pages* (JSP), struts[5] e *JavaServer Pages Standard Tag Library* (JSTL), que são feramentas *open source*, usadas para produzir páginas WEB eficazes, dinâmicas e iterativas;
- A conexão entre a base de dados e o *front-end* foi feita através de código java 1.5, com o suporte do *springframework*[6];
- Para fazer integração código java, *springframework*, *struts*, *JSP, JSTL* foi usado o editor eclipse[7] (versão 3.1);
- Foi produzida a documentação do sistema, para facilitar a manutenção do sistema e possíveis correcções, visto que um sistema nunca está isento de erros.

Para testar o modelo desenhado:

- Organizou-se a plataforma e recursos necessários para o sustento do sistema;
- Os módulos foram separadamente testados durante o desenvolvimento usando a ferramenta JUnit[8] para a realização de testes unitários, cujos resultados se encontram no CD da aplicação, em anexo ao trabalho;
- Criaram-se alguns *web-testes* usando a linguagem de programação *ruby*[9] para desenvolvimento de um *framework* para a realização de testes de simulação de uso da aplicação.
- Foram elaborados ainda, diversos tipos de testes, entre os quais, teste de funcionalidade, teste de *user interface*, teste de performance e teste de stress (usando a ferramenta Jmeter[10]), de forma a acompanhar o funcionamento do sistema e o envolvimento dos interessados, detectando falhas e/ou erros do mesmo.

[5] ***Struts*** – um *framework open-souce* para o desenvolvimento de aplicações web com Java EE (Wikipédia, 2006a)
[6] ***Springframework*** – um framework *open-source* que facilita a conexão com base de dados e garante maior segurança nas transações.
[7] http://www.eclipse.org/
[8] http://www.junit.org/index.htm
[9] http://www.ruby-lang.org/
[10] http://jakarta.apache.org/jmeter/

1.5 Estrutura do trabalho

Este trabalho está dividido em seis (6) capítulos apresentando-se no primeiro, o enquadramento do trabalho, os problemas identificados no mundo do software e no sistema de emissão de declarações e certificados, bem como os objectivos do trabalho e a metodologia usada para atingir tais objectivos. O capítulo seguinte faz uma abordagem descritiva da metodologia *Agile Modeling*. O terceiro capítulo refere-se a descrição do sistema actual e o proposto. Os dois capítulos seguintes referem-se a modelação do Sistema (o quarto capítulo) e os critérios de segurança previstos, e, por fim, as conclusões e recomendações no último capítulo.

2. A METODOLOGIA AGILE MODELING

> ***"Uma maneira produtiva de pensar acerca de desenvolvimento de software é considerá-lo como um jogo cooperativo de invenções e comunicação"*** (Cockburn, 2002).

Numa troca de emails, entre 29 de Abril e 04 de Julho de 2006, com *Scott W. Ambler*[11] (que se intitula como sendo o primeiro a escrever sobre esta metodologia), ele disse que tinha notado que a maioria dos artigos acerca de modelação de software não tinham sentido algum, soavam muito bem na teoria, mas na prática provaram ser incrivelmente ineficientes, e, portanto, começou a escrever algo que funcione na prática. Isto levou Ambler, nos finais de 2000, a escrever um artigo chamado *Extreme Modeling* para uma revista de desenvolvimento de software, que mais tarde, em 2001, eles (Ambler e a revista) renomearam o artigo para *Agile Modeling (AM).*

Agile Modeling é uma metodologia baseada na prática para modelação e documentação eficaz de sistemas baseados em *software.* AM é uma colecção de práticas geridas por princípios e valores, para modelação de *software,* que podem ser aplicados por profissionais de *software* no dia a dia (Ambler, 2005).

O seu ponto forte não é o conjunto de técnicas de modelação em si – tais como modelos de caso de uso, modelos de classes ou modelos de *user-interface* – mas sim, como as tais técnicas são aplicadas. Um desenvolvedor pode levar uma abordagem ágil de modelação para os requisitos, análise arquitectura e desenho.

Não sendo um processo prescritivo, não define procedimentos detalhados de como criar um dado tipo de modelo, em vez disso, provê conselhos de como ser efectivo como modelador. Em resumo, AM vai ao encontro, através dos seus princípios e práticas, do ponto de melhor custo/benefício entre os artefactos[12] que devem ser criados para o sistema e o custo de manutenção e actualização desses artefactos.

[11] www.ambysoft.com

[12] **Artefacto** é o produto de uma ou mais actividades dentro do contexto do desenvolvimento de um software ou sistema (Wikipédia, 2006a).

Ao contrário dos processos "tradicionais" como o sugerido pelo *Unified Process* (UP), por exemplo, que requer basicamente os mesmos artefactos para todos os tipos de projectos, AM vai ao encontro da construção e manutenção eficiente de artefactos, criando-os apenas quando agregarem valor real ao projecto, e focando principalmente os esforços no desenvolvimento do *software* que, em última análise, é o objectivo principal do processo (Santos, 2003).

Deve-se notar, entretanto, que AM não é uma metodologia de desenvolvimento ágil como *eXtreme Programming*[13] (XP), SCRUM[14], *Dynamic Systems Development Method* (DSDM), etc., mas uma metodologia de modelagem ágil, isto é, AM visa construir e manter modelos de sistemas de maneira eficaz e eficiente e, portanto, pode ser utilizada dentro de metodologias ágeis como as citadas há pouco, como também em metodologias prescritas como o *Unified Process (UP)* (Ambler, 2005).

Este apresenta a definição de modelos ágeis e a descrição das motivações para a criação desta metodologia, recorrendo a descrição dos valores e princípios da aliança ágil que deram origem ao estudo nesta área, valores estes que sustentam os valores, princípios e práticas da AM, descritas neste capítulo. Este capítulo descreve ainda, a forma como a AM trata a documentação incluindo estratégias para garantir a agilidade[15] na criação de documentos.

2.1 AM realça outros processos de software

Ambler (2002) cita três (3) objectivos a atingir com esta metodologia:

1. Definir e mostrar como pôr em prática uma colecção de valores, princípios e práticas pertinentes para uma modelação efectiva e leve.

[13] **eXtreme Programming** é uma metodologia ágil para equipas pequenas e médias e que irão desenvolver software com requisitos vagos e em constante mudança. Para isso, adopta a estratégia de constante acompanhamento e realização de vários pequenos ajustes durante o desenvolvimento de software (Wikipedia, 2006b).
[14] **SCRUM** é um método ágil para gestão de projectos. Ele é documentado para melhorar a produtividade dinamicamente em equipas previamente paralisadas por metodologias mais pesadas (Wikipedia, 2006b).
[15] **Agilidade** é a habilidade de criar e responder as mudanças de modo a lucrar num ambiente de negócio agitado (Highsmith, 2002).

2. Explorar como aplicar as técnicas de modelação em projectos de *software* levando uma abordagem ágil.

3. Explorar como se podem melhorar as actividades de modelação seguindo uma abordagem ágil para o desenvolvimento de *software*, e em particular, para equipas de projecto que tenham adoptado uma instanciação de *Unified Process* como *Rational Unified Process* (RUP) ou *Enterprise Unified Process* (EUP). Estes processos são todos flexíveis tal que num extremo são muito prescritivos e no outro extremo ágeis de modo que AM possa trabalhar com elas.

Um conceito importante a compreender acerca de AM é que AM não é um processo completo de *software,* ela tem o seu foco numa modelação efectiva e documentação. Ela não inclui actividades de programação, de teste, de gestão de projectos mais ela vai dizer-lhe para provar os seus modelos com código e considerar a testabilidade durante a modelação. Ela não cobre a gestão de projectos, desenvolvimento, operação e suporte de sistemas. Uma vês que o seu foco está em todo processo de *software*, é preciso usá-la como um processo desenvolvido, tal como XP, DSDM, SCRUM ou UP.

Segundo Ambler (2005), embora AM seja independente de outros processos tais como XP e RUP, ela é usada para realçar esses processos, como mostra a figura que se segue. Temos em volta dos princípios e práticas da AM, as outras técnicas de desenvolvimento como XP, RUP, entre outros; e dentro delas se podem aplicar os princípios e práticas da AM.

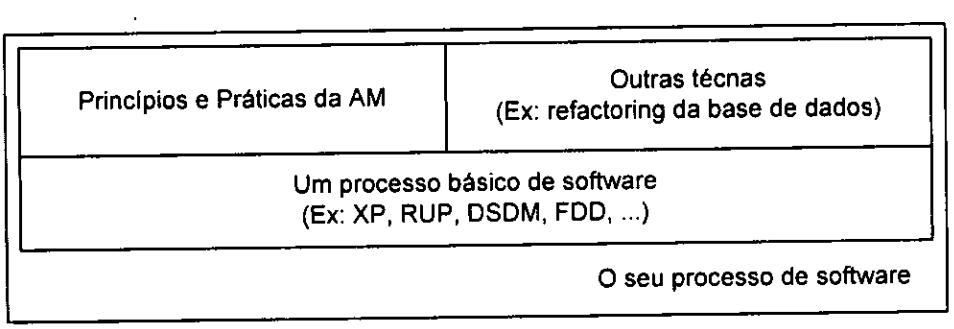

Figura 1 – AM realça outros processos de *software* (Fonte: Ambler, 2005)

2.2 Motivações para a criação da metodologia AM

Santos (2003), cita duas motivações principais para a criação desta metodologia:

- O objectivo primário de um projecto de software é o próprio software, e não um grande conjunto de documentação sobre ele;
- Um artefacto é criado primordialmente para permitir a comunicação e a troca de informações entre a equipa e permitir a discussão e refinamento do modelo do sistema. Assim, se um artefacto não passa informação relevante ao projecto, ele não cumpre seu objectivo básico.

Foi a partir deste tipo de constatações que um grupo de dezassete pesquisadores esteve reunidos, de 11 a 13 de Fevereiro de 2001, num alojamento nas montanhas de *Wasatch* em Utá (um estado dos Estados Unidos da América) para esquiar, relaxar e trocar impressões sobre o processo de desenvolvimento de software. Como resultado, foi definido um manifesto para o desenvolvimento ágil de software, que define um conjunto de ***quatro valores*** e ***doze princípios*** (Anexo A) a serem praticados e seguidos para a obtenção de um processo de desenvolvimento mais interactivo e menos burocrático do que os actuais (Agile Alliance, 2001c).

Estes princípios definem requisitos de alto nível para uma metodologia efectiva de *software*, requisitos estes usados para formular os valores, princípios e práticas da *Agile Modeling*

2.3 Definição de modelos ágeis

Segundo Ambler (2002), um modelo ágil é um modelo que é apenas bom o suficiente. É um modelo que atinge o seu propósito: é inteligível para a sua audiência planejada, é simples, suficientemente exacto, consistente e detalhado; e o investimento na sua criação e manutenção provê valor positivo ao projecto.

Um modelo é considerado suficientemente bom quando apresenta as seguintes características:

Atende o seu propósito

Um modelo tem por propósito comunicar algo para a equipa. Se o modelo não traz informação adicional sobre o projecto, então, ele não está cumprindo o seu papel e não vale a pena pagar o custo necessário para mantê-lo.

É inteligível

Um modelo deve ser inteligível, e não bonito. Isto significa que um modelo desenhado à mão em uma folha de papel ou uma foto digital de um modelo desenhado em um quadro branco, pode comunicar a informação pretendida tão bem como o mesmo modelo desenhado em uma ferramenta CASE da moda, com a vantagem de ser muito mais fácil de ser construído.

É suficientemente exacto

Modelos não precisam ser 100% exactos; eles apenas precisam ser exactos o suficiente. Quando se encontra um problema no modelo de requisitos, ou no modelo de dados, pode-se decidir corrigi-lo ou aceita-lo como está – bom o suficiente mas não perfeito. Algumas equipas podem tolerar inexactidão enquanto que outras não. A natureza do projecto, a natureza dos membros da equipa, e a natureza da organização é que decide isto.

É suficientemente consistente

Um modelo ágil não precisa ser perfeitamente consistente com ele mesmo ou com outros artefactos para ser útil. Num mundo ideal todos seus artefactos deveriam ser perfeitamente consistentes, mas não é necessário, frequentemente não funciona desta maneira. Quando se constrói uma simples aplicação de negócios, pode-se tolerar algumas inconsistências.

É suficientemente detalhado

Todo o modelo deve ser actualizado de acordo com a evolução do sistema. Se o modelo for muito detalhado, o custo de manter esse modelo actualizado é maior devido à sua

maior complexidade. O modelo deve ser suficientemente detalhado para conseguir transmitir a informação desejada e manter o custo de manutenção em um nível mínimo. Muito detalhe num ponto pode inserir informação demais no modelo, o que gera uma complexidade desnecessária. Esse detalhe adicional normalmente pode ser colocado em um outro modelo ou, preferencialmente, definido pelo próprio código do sistema.

Provê um valor positivo

Um aspecto fundamental de um artefacto de um projecto é que ele deve adicionar um valor positivo. O benefício que um modelo de arquitectura trás ao projecto não deve exceder os valores do custo do desenvolvimento e manutenção do mesmo. Se os custos do modelo excedem o valor dos benefícios, então ele não provê mais valor positivo.

É o mais simples possível

É preciso esforço para manter os modelos o mais simples possível, mas sobretudo ter o trabalho feito. A Simplicidade é claramente afectada pelo nível de detalhe dos modelos, mas também pode ser afectado pela magnitude da notação que se aplica.

2.4 Valores da AM

Quando Scott W. Ambler leu uma obra de Kent Beck sobre XP na primeira edição de *Extreme Programming Explained*[16] ele concordou na totalidade com os seus quatro valores (*Comunicação, Simplicidade, Feedback, Coragem*) e decidiu adoptá-los uma vez que não faz sentido nenhum refazer algo quando se pode reusar. Mas sentiu que algo estava em falta. Até que um dia vendo uma discussão entre um desenvolvedor e um cliente acerca dos requisitos, ele descobriu o valor o qual ele sentia em falta, a *Humildade* (Ambler, 2002, 2006d).

[16] http://www.amazon.com/exec/obidos/ASIN/0321278658/ambysoftinc/103-2623302-5194249

2.4.1 Comunicação

Uma comunicação efectiva entre todos os envolvidos no seu projecto incluindo tanto desenvolvedores como *stakeholders*[17] é um requisito para o sucesso do desenvolvimento do *software*.

Num projecto de *software* vários problemas podem advir de uma falha da comunicação. Por exemplo, o facto de um desenvolvedor não dizer aos seus colegas que um determinado código não funciona em perfeitas condições, pode resultar num trabalho extra na parte do outro desenvolvedor para rastear o problema.

2.4.2 Simplicidade

É importante que desenvolvedores entendam que os modelos são críticos na simplificação tanto do *software* como do processo do *software* – é muito mais fácil explorar uma ideia e melhorá-la enquanto o seu entendimento aumente, desenhando um ou dois diagramas em vez de escrever centenas de linhas de código.

Ambler (2002) no seu trabalho *Agile Modeling (Effective Practices for eXtreme Programming and the Unified Process),* aconselha a não usar *patterns* complexos sem que haja necessidade, a não sobrecarregar a arquitectura do seu projecto para suportar requisitos futuros; aconselha ainda a manter os modelos o mais simples possível, *modelar hoje para atingir as necessidades de hoje*, e se preocupar amanhã com as necessidades de amanha.

2.4.3 Feedback

A única maneira de determinar que o seu trabalho está correcto é obtendo feedback, e que o feedback abranja os seus modelos.

Existem várias formas de obtenção de retorno nos seus modelos, dentre elas:

[17] ***stakeholder*** é qualquer um que seja utilizador directo/indirecto, gestor de utilizadores, gestor sénior, pessoal do nível operativo, pessoal do suporte, desenvolvedores que trabalham no outro sistema que integra ou interage com o software em desenvolvimento, ou profissionais de manutenção afectados pelo desenvolvimento de um projecto de software (Ambler, 2006a).

- Comunicando as suas ideias através de diagramas. Para além de rapidamente obter o retorno, faz com que haja no conselho dado;
- Desenvolvendo os modelos em equipa;
- Fazendo a revisão dos modelos com a audiência alvo;
- Implementando o modelo no *software* e ter o *stakeholder* do seu projecto a trabalhar com o *software*.

2.4.4 Coragem

AM e desenvolvimento ágil de software em geral, são conceitos novos para a maioria das pessoas, tal como para as organizações nas quais eles trabalham.

Coragem é muito importante para a tomada de vários níveis de decisão, como, por exemplo a escolha do uso de uma abordagem ágil e permanecer com essa abordagem durante todo o processo, a escolha de uma arquitectura em vez da outra, a escolha de uma linguagem de desenvolvimento. Durante o desenvolvimento será preciso ter coragem para mudar de direcção, quando algumas das decisões tomadas se provarem inadequadas, descartando ou reescrevendo o código.

Coragem é igualmente importante para confiar em superar amanhã os problemas de amanhã.

2.4.5 Humildade

Os melhores desenvolvedores têm a humildade de reconhecer que eles não sabem tudo, que os desenvolvedores da mesma categoria, seus clientes e de facto todos os *stakeholders* do projecto também têm suas próprias áreas de perícia e algum valor para adicionar ao projecto. Uma abordagem efectiva é assumir que todos os envolvidos no seu projecto tenham igual valor, e, portanto, devem ser tratados com consideração.

Uma abordagem positiva é de estimar os outros em vez de estimarmos a nós, onde se trata a opinião dos outros como se elas tivessem mais valor do que as suas. Com esta

abordagem a sua primeira reacção em relação à ideia dos outros será mais positiva (Ambler, 2002).

2.5 Princípios da AM

Os valores da AM em combinação com os valores e princípios da aliança ágil, são usados para definir os princípios da AM. Quando aplicados num projecto de desenvolvimento de *software*, estes princípios mudam a fase para uma colecção de práticas de modelação (Ambler, 2002).

2.5.1 Princípios fundamentais ou centrais

Ambler (2002) aconselha o uso na sua totalidade dos princípios fundamentais da AM para que a modelação possa ser considerada ágil. Os princípios fundamentais da AM são:

- Modele com um propósito;
- Maximize o investimento do *stakeholder*;
- Viaje com pouca bagagem (*Travel Ligth*);
- Construa múltiplos modelos;
- Obtenha rápido feedback;
- Assuma simplicidade;
- Aceite a mudança;
- Aplique mudanças incrementais;
- Trabalho de qualidade;
- *Software* é o seu objectivo primário;
- Habilitar o seu próximo esforço é um objectivo secundário.

Modele com um propósito

Durante a fase da modelação, é muito importante ter um propósito e uma audiência para a criação dos modelos. Pode ser que esteja a tentar compreender um aspecto no *software*, pode ser para apresentar ao gestor do projecto ou mesmo um documento para o pessoal da manutenção. Mas, se não consegue identificar porquê e para quem está a criar o modelo, então, não precisa preocupar-se e gastar tempo criando e mantendo o modelo.

O primeiro passo é identificar um motivo válido para a criação do modelo, e a audiência do modelo. Uma vez identificados os motivos e a audiência do modelo, devemos desenvolve-lo até um ponto que seja o bastante acurado e suficientemente detalhado. Uma vez que o modelo tenha satisfeito os seus objectivos, já se pode considerar que o modelo tenha terminado por enquanto e fazer outras tarefas relacionadas, como escrever algumas linhas de código para testar se o modelo funciona.

Reconheça que você precisa apenas de criar modelos que são bons o suficiente para a sua tarefa – modelos não precisam ser perfeitos (eles nunca são), eles nem precisam estar completos, eles precisam de satisfazer os requisitos da sua audiência.

Este princípio também se aplica durante a alteração de um modelo.

Maximize o investimento do stakeholder

Os *stakeholders* dos seus projectos investem recursos – tempo, dinheiro, facilidades, etc. – para terem um *software* que satisfaça as suas necessidades. Os *stakeholders* merecem investir seus recursos da melhor maneira possível, e não verem seus recursos serem ignorados pelos elementos da sua equipa. Ainda mais, eles merecem ter uma palavra final em como estes recursos serão ou não investidos.

Viaje com pouca bagagem (Travel Ligth)

Todo o artefacto criado e guardado precisará ser mantido ao longo do tempo. Se você decide guardar sete modelos, qualquer alteração (um novo requisito, uma tecnologia que

sua equipa decidiu adoptar) que ocorra você precisará considerar o seu impacto nos sete modelos e agir de acordo com esse impacto. Se você decide guardar apenas três modelos, está claramente visível que você terá menos trabalho para suportar a mudança, fazendo com que você se torne mais ágil porque você estará a "*viajar com pouca bagagem*".

Similarmente, quanto mais complexo/detalhado forem os seus modelos, está bem claro que qualquer alteração se tornará mais difícil de acoplar.

Cada vez que você decide guardar um modelo você estará perdendo agilidade a favor da conveniência de ter a informação do modelo disponível para a sua equipa.

Uma equipa de desenvolvimento que decide desenvolver e manter um detalhado documento de requisitos, uma detalhada colecção de modelos de analise, uma detalhada colecção de modelos da arquitectura e uma detalhada colecção de modelos de desenho vai rapidamente descobrir que gasta a maior parte do seu tempo actualizando documentos do que escrevendo algumas linhas de código (Ambler, 2006b).

Construa Múltiplos Modelos

Qualquer desenvolvedor precisa de usar potencialmente múltiplos modelos para desenvolver *software* porque cada modelo descreve um aspecto do *software*. Para suportar a complexidade de um *software* moderno, é preciso ter um campo vasto de técnicas no seu conjunto de ferramentas de modelação.

Um ponto importante é que não é preciso desenvolver todos estes modelos para qualquer sistema dado, mas isso depende da natureza exacta do *software* que se está a desenvolver. Você vai precisar, pelo menos, de um subconjunto de modelos.

A título de exemplo, qualquer trabalho caseiro não requer o uso de todas ferramentas disponíveis.

Ao longo do tempo, as variedades de trabalhos que fizer vão requerer o uso de uma dada ferramenta, tal como qualquer pessoa usa uma ferramenta mais do que a outra, você usará mais algum tipo de modelo do que os outros.

Obtenha Rápido feedback

O tempo entre uma acção e o seu retorno é crítico. Trabalhando com outras pessoas num modelo, particularmente quando se trabalha com uma tecnologia de modelação partilhada (como quadro, *Class Responsibility Collaboration cards*[18] *(CRC cards)*, etc.) você obtém retorno instantâneo nas suas ideias. Trabalhando perto do seu *stakeholder*, para entender melhor os requisitos e analisá-los, ou para desenvolver o *user-interface* que vai de acordo com as suas necessidades, traz oportunidade para retorno rápido.

Assuma simplicidade

Durante o desenvolvimento, deve-se assumir que a solução mais simples é a melhor. Não sobrecarregue o seu *software*, ou no caso de AM, não represente funcionalidades adicionais nos seus modelos que não são precisas hoje. É preciso ter em conta que não é preciso sobrecarregar o modelo hoje, deve-se modelar baseando-se nos requisitos existentes hoje e ajustar o sistema no futuro quando os requisitos evoluírem. Mantenha os seus modelos o mais simples possível.

Aceite a Mudança

Alguns requisitos do sistema a desenvolver evoluem com o tempo. O entendimento do pessoal sobre os requisitos muda com o tempo. Os *stakeholders* dos projectos podem mudar durante o decorrer do seu projecto, pessoal novo vai entrando no projecto e pessoal antigo vai saindo. Os *stakeholders* podem mudar os seus pontos de vista, alterando os objectivos e o sucesso da sua conquista. A implicação é que o ambiente do seu projecto muda no momento em que o seu esforço progride e, como resultado, o seu modelo de desenvolvimento deve reflectir esta realidade.

[18] *Class Responsibility Collaboration cards* (CRC cards) são ferramentas de quebra-cabeças usadas no desenho de software orientado a objectos. Elas foram propostas por Ward Cunningham e são tipicamente usadas quando primeiramente se determina que classes usar e como elas interagem.

Aplique Mudanças incrementais

Um conceito importante a perceber, no que diz respeito a modelação, é que ela não precisa ser correcta a primeira vez, de facto, é muito desgostoso que você não consiga mesmo que as circunstâncias não o ajudem.

Ainda mais, não é obrigatório capturar cada detalhe nos seus modelos, é preciso tê-los pelo menos suficientemente bons. Em vês de tentar desenvolver todo modelo no início, pode-se ir desenvolvendo um modelo pequeno, ou talvez um modelo de um nível mais elevado e evolui-lo com o andar do tempo de uma maneira incremental.

Trabalho de qualidade

Ninguém gosta de trabalho sem qualidade, o pessoal que o faz também não gosta porque é algo do qual não se podem orgulhar, o pessoal que vai aparecendo para dar continuidade ao trabalho também não gosta porque o trabalho torna-se difícil de perceber, e, portanto, difícil de manter, e, por último, os utilizadores também não irão gostar de trabalhar por ser frágil e não satisfazer as suas expectativas.

Software é o seu objectivo primário

O objectivo primário de um processo de desenvolvimento de *software* é o próprio *software* que satisfaça as necessidades do *stakeholder* de uma maneira efectiva. O objectivo primário não é produzir documentos e artefactos estranhos ou mesmo modelos. Qualquer actividade que não contribui directamente para este objectivo deveria ser questionada e evitada se não poder ser devidamente justificada.

Habilitar o seu próximo esforço é um objectivo secundário

O seu projecto pode ser considerado um fracasso, mesmo que a sua equipa entregue o sistema aos utilizadores – parte do alcance das necessidades do seu *stakeholder* é garantir a robustez do sistema e sua evolução ao longo do tempo.

A sua próxima tarefa pode ser o desenvolvimento duma próxima versão ou, simplesmente, as operações e o suporte da versão corrente. Para garantir isto, você não deve desenvolver apenas *software* com qualidade, mas deve também criar documentação e material de suporte suficiente que o pessoal envolvido possa usar. Factores que você precisa considerar incluem que os membros da sua equipa estejam envolvidos com a tarefa seguinte, a sua natureza e a importância da mesma. Quando se trabalha num sistema, é necessário manter um olho no futuro.

2.5.2 *Princípios suplementares*

Os princípios suplementares da AM definem conceitos para aumentar a sua efectividade como um modelador ágil. Embora estes princípios apoiem os princípios fundamentais da AM, a sua adopção não é necessária para se considerar um modelar ágil. Estes princípios têm boas ideias e deveriam ser adoptadas se elas se encaixassem bem na sua cultura organizacional. Os princípios suplementares da AM são (Ambler, 2006b):

- Conteúdo é mais importante do que representação;
- Comunicação aberta e honesta.

Conteúdo é mais importante do que representação

Um dado modelo pode ter várias formas de ser apresentado. Por exemplo, a especificação de *user-interface* pode ser criada usando "*post-it notes*" numa imensa folha de papel, com um esboço num papel ou quadro, usando uma ferramenta para o desenho de protótipos, uma linguagem de programação ou um documento formal que inclui tanto a representação visual tal como uma descrição textual do *user-interface*. Um ponto importante é que um modelo não precisa de ser um documento, mesmo um complexo conjunto de diagramas criados usando uma ferramenta case pode não fazer parte dum documento, mas eles são usados como *input* para outros artefactos como código, por exemplo, mas nunca formalizados com documentação. O ponto é que se leva vantagem dos benefícios de modelar sem considerar os custos de criar e manter documentos.

Comunicação aberta e honesta

Os membros da sua equipa precisam de ser livre e perceber que são livre de sugerir. Isto inclui ideias pertinentes para um ou mais modelos. Talvez alguém tenha um novo caminho para se aproximar a uma porção do desenho ou ter um novo discernimento que envolve um requisito; a apresentação de más notícias, tais como o atraso do projecto ou simplesmente o estado corrente do seu trabalho. Uma comunicação aberta e honesta faz com que a equipa se torne menos tímida e tome melhores decisões porque a qualidade da informação em que ela se baseia é mais precisa.

2.6 Práticas da AM

O coração de AM está nas práticas (Ambler, 2002). São as práticas o que se deve aplicar nos projectos. Estas práticas são guiadas pelos valores e princípios da AM. Tal como os princípios, elas (as práticas) estão organizadas em fundamentais e suplementares.

2.6.1 Práticas fundamentais ou centrais

Tal como os princípios fundamentais, estas práticas devem ser seguidas à risca para poder afirmar que está a modelar de forma ágil. Claro que você pode beneficiar-se de apenas algumas das práticas mas é aconselhável usar todas se elas se encaixam bem na cultura da sua organização.

As práticas fundamentais da AM estão organizadas em quatro categorias, que são listadas e detalhadas a seguir:

1. **Modelação Iterativa e Incremental**
   - Aplique o artefacto correcto;
   - Crie vários modelos em paralelo (trabalhar em vários modelos em paralelo);
   - Modele em pequenos incrementos;
   - Itere entre diferentes artefactos.

2. **Trabalho em Equipa**

- Participação activa do *stakeholder*;
- Modele com outras pessoas;
- Exiba modelos publicamente (colocar em painéis, parede, etc.);
- Propriedade colectiva.

3. **Simplicidade**

- Crie um conteúdo simples;
- Represente os modelos de forma simples;
- Use a ferramenta mais simples.
- Use único repositório para a informação.

4. **Validação**

- Prove os seus modelos com código.

2.6.1.1 Práticas para Modelação Iterativa e Incremental

Aplique o artefacto correcto

Esta prática é a equivalente a frase "*use a ferramenta certa para o trabalho*". Neste caso, você quer criar o modelo certo para ter o trabalho feito. Cada artefacto – como um diagrama de estados em UML, um caso de uso indispensável, código fonte, ou um digrama de fluxo de dados (DFD) – tem o seu ponto forte e as suas fraquezas, e, portanto, é apropriado para algumas situações mas não para outras. Muitas vezes um diagrama é melhor escolha que algumas linhas de código.

Crie vários modelos em paralelo

Uma vez que cada tipo de modelo tem o seu ponto forte e as suas fraquezas, nenhum modelo único é suficiente para as necessidades da modelação. Por exemplo, quando se explora os requisitos, pode-se precisar de desenvolver alguns casos de uso, um protótipo

de *user-interface*, e algumas regras de actividade. Em combinação com a prática de iteração entre diferentes artefactos, modeladores ágeis descobrirão que eles são mais produtivos trabalhando em vários modelos em simultâneo do que se eles estiverem focados em um único modelo num dado tempo.

Modele em pequenos incrementos

Desenvolvimento incremental, no qual você organiza uma tarefa grande em pequenas porções que você lança ao longo do tempo, esperançosamente em incrementos de várias semanas, um ou dois meses, aumenta a sua agilidade fazendo com que você entregue o *software* ao seus *stakeholders* mais rápido.

Itere entre diferentes artefactos

Quando se está a trabalhar num determinado artefacto – com um caso de uso, digrama de sequência ou mesmo código fonte – e se descobre enormes dificuldades em trabalhar nele, é um sinal de haver uma necessidade de iterar para um outro artefacto. Pois iterando para um novo artefacto você se torna aliviado porque poderá estar a progredir no novo artefacto. Além disso, mudando o seu ponto de vista, na maioria dos casos, acaba-se ultrapassando as dificuldades que lhe estavam a encravar no primeiro artefacto.

2.6.1.2 Práticas para um trabalho efectivo em equipa

Participação activa do stakeholder

Uma expansão de *XP's on-site customer* que descreve a necessidade de ter acesso *on-site* aos utilizadores que têm autoridade e habilidade para prover informação pertinente ao sistema em desenvolvimento e tornar pertinente e atempadas as decisões que tem a ver com requisitos e sua priorização. AM expande a prática de *XP's on-site customer* para ter os *stakeholders* do projecto – incluindo utilizadores directos, sua gestão, gestão sénior, nível operacional e pessoal do suporte (*helpdesk*) – activamente envolvidos no projecto. Isto inclui tomar atempadamente as decisões do uso dos recursos pela gestão sénior, suporte público e privado para o projecto. Participação activa do pessoal do nível

operacional e de suporte no desenvolvimento dos requisitos e modelos pertinentes para as suas respectivas áreas.

Modele com outras pessoas

Quando se modela com um objectivo, facilmente se descobre que se está a modelar para compreender algo, para comunicar uma ideia aos outros ou se está a procura de desenvolver uma visão comum do projecto. Facilmente se descobre também que a sua equipa de desenvolvimento precisa de trabalhar em conjunto para criar o núcleo de modelos críticos no seu projecto. Por exemplo, para desenvolver a arquitectura dum sistema, vai-se precisar constantemente de modelar com um grupo de pessoas para desenvolver uma solução que seja da concordância de todos, e que seja a mais simples possível. Na maioria dos casos, a melhor maneira de fazer isto é discutir a tarefa com uma ou mais pessoas.

Exiba os seus modelos publicamente

É aconselhável visualizar os modelos publicamente com frequência num local conhecido como parede de modelação. Isto conduz a uma comunicação honesta e aberta na sua equipa porque os todos modelos correntes são rapidamente acessíveis a equipa, tal como os *stakeholders* do seu projecto pois você não estaria a esconder nada para eles. A sua parede de modelação é onde você coloca os seus modelos. Esta parede de modelação pode ser física, um quadro branco para os seus diagramas de arquitectura ou onde você cola uma impressão dos seus modelos de dados. A parede de modelação também pode ser virtual tal, como uma página de Internet que é actualizada com imagens através de um *scan*.

Propriedade colectiva

A ideia desta prática é que todos possam trabalhar em qualquer módulo, e, de facto em, qualquer artefacto se eles precisarem de o fazer, e nunca deixar que somente uma pessoa domine todo o processo, pois se a mesma morrer, acabou o projecto. Esta prática é encorajada no uso de ferramentas partilhadas como quadros.

2.6.1.3 Práticas que habilitam simplicidade

Crie um conteúdo simples

É preciso manter o conteúdo actual dos seus modelos – os seus requisitos, sua análise, sua arquitectura, ou o seu desenho – o mais simples que puder enquanto ainda estiver a captar as necessidades dos seus *stakeholders*. O importante é que você não deve incorporar aspectos adicionais aos seus modelos a não ser que sejam justificáveis – se você não tiver um requisito que o faça adicionar uma dada funcionalidade no sistema, então, não adicione essa funcionalidade aos seus modelos. Tenha coragem de confiar que você pode, de facto, acrescentar essa funcionalidade no futuro quando e se alguém pedir.

Representa os modelos de forma simples

Quando você considera os potenciais diagramas que pode ter (Diagramas UML, modelos de dados, etc.) rapidamente você se apercebe que a maioria dos casos você precisa apenas de um subconjunto de notação de diagramas que lhe é disponível, um modelo simples que mostra as funcionalidades-chave que você está a tentar perceber, talvez um modelo de classes descrevendo as responsabilidades primárias das classes e a relação entre elas. Você podia modelar todo o código que vai precisar de escrever, todas operações de *get* e *set* que os padrões da sua codificação lhe indicam para usar. Mas que valor isso traz? Muito pouco.

Use as ferramentas mais simples

A maioria dos modelos pode ser desenhado num quadro ou num papel. Sempre que for preciso guardar um destes diagramas, pode-se tirar uma foto com uma câmara digital, ou simplesmente transcrever para uma folha de papel. Isto funciona muito bem porque muitos diagramas são deitados fora após algum tempo, o seu valor vem do desenhá-los para pensar durante uma tarefa, e uma vez que esta tarefa esteja resolvida, este diagrama já não oferece mais valor ao projecto. Como resultado, um quadro e um marcador são, muitas vezes, a melhor ferramenta de modelação alternativa. Deve-se usar uma ferramenta de desenho para criar diagramas que servirão para mostrar à *stakeholders*

importantes do projecto e, ocasionalmente, uma ferramenta de modelação se e somente se os modelos a desenvolver têm um valor para as tarefas de modelação, como, por exemplo, gerar código. Pense da seguinte maneira: se você modela para entender uma tarefa, certamente que não precisa guardar estes modelos uma vez que tenha compreendido a tarefa, então, não é preciso usar uma ferramenta de modelação complexa para o efeito.

Use único repositório para a informação

A informação deve ser guardada num único repositório. Em outras palavras, o importante não é só usar os artefactos certos, deve-se modelar um conceito uma única vez, guardando a informação no melhor lugar possível. Enquanto se modela, é preciso fazer sempre a seguinte pergunta: Preciso de reter esta informação permanentemente? Se sim, qual é o melhor local para guardar esta informação? E esta informação já foi capturada num outro local que eu podia simplificar a referência? Algumas vezes o melhor local para guardar a informação é num *"documento agile"* e também no código fonte.

Guardando a informação num único repositório reduz-se a carga de manutenção, reduz-se a carga de actualização e sincronização e aumenta-se a consistência da informação.

2.6.1.4 Práticas para validar o seu trabalho

Prove os seus modelos com código

Um modelo é uma abstracção, algo que devia, precisamente, reflectir o aspecto do que quer que seja que você esteja construindo. A questão é: O modelo vai funcionar? Para poder garantir isto, você terá que provar o seu modelo com algumas linhas de código. Por exemplo, se você desenvolveu um modelo de uma página HTML para efectuar encomendas *on-line*, deve-se codificar e mostrar que o *user-interface* resultante aos utilizadores de forma a obter um *feedback*. Se você desenha um diagrama de sequência em UML que representa a implementação de uma regra de actividade complexa, convém escrever um código de teste e correr os testes para se certificar que o desenho funciona. Portanto, deve-se modelar, codificar e testar continuamente.

2.6.2 Práticas suplementares

AM inclui práticas suplementares que suportam as suas práticas fundamentais descritas na secção anterior. Estas práticas são opcionais, a sua equipa pode decidir adoptar ou não. Similarmente às práticas fundamentais, estas estão organizadas duas em categorias:

1. **Produtividade**
   - Utilize padrões e normas de modelação;
   - Aplique padrões (*design patterns*) gentilmente.
2. **Documentação**
   - Actualize apenas quando "doer";
   - Formalize os modelos de contrato (*Contract Model*);
   - Descarte Modelos temporários.

2.6.2.1 Práticas para melhorar a sua produtividade

Utilize padrões e normas de modelação

Esta prática é renomeada da *XP's Coding Standards*[19]. A ideia básica é que desenvolvedores devem concordar e seguir um conjunto comum de padrões de modelação num projecto de *software*. Seguir convenções comuns de codificação traz um valor grande ao projecto, um código limpo que segue os seus *guidelines* é fácil de entender. Existe um valor similar em seguir convenções de modelação. Existe uma grande variedade de padrões comuns de modelação disponíveis incluindo UML (do grupo *Object Oriented*), que define a notação e semântica para modelos comuns orientados a objectos. UML fornece um bom início mas não é suficiente –como podemos ver no *Be Realistic about UML*[20] – nem todos artefactos de modelação são abrangidos por UML. Além do mais, não diz nada acerca de *guidelines* para estilos de modelação para criar diagramas limpos. Em relação a código, um padrão seria nomear os atributos num formato *atributoNome* enquanto que *guidelines* de estilo é alinhar o

[19] http://www.ambysoft.com/essays/codingGuidelines.html
[20] http://www.agilemodeling.com/essays/realisticUML.htm

código numa estrutura de controle. Em relação a *standards* de modelação, seria usar quadrados/rectângulos para modelar uma classe num diagrama de classes e em *guidelines* de estilo seria ter subclasse do diagrama colocados a sul das super classes.

Aplique padrões (design patterns) gentilmente

Modeladores efectivos aprendem e aplicam *patterns* comuns de arquitectura, desenho e análise nos seus modelos. Os desenvolvidores devem considerar a facilidade que a aplicação de *patterns* traz para o usarem delicadamente. Isto reflecte o valor da simplicidade. Em outras palavras, se você suspeita que um *pattern* se aplica ao seu caso, então, você deveria modelar de modo a implementar o mínimo que você precise hoje mas que torne fácil fazer *refactoring* mais tarde quando estiver claro que, aplicando o *pattern* desenvolvido é, de facto, a abordagem mais simples possível. Em outras palavras, não sobrecarregue o teu modelo.

2.6.2.2 Práticas para uma documentação ágil:

Actualize apenas quando "doer"

Você deve actualizar um modelo apenas quando for absolutamente necessário, quando é mais doloroso manter o modelo desactualizado do que o esforço de actualizá-lo. Com esta abordagem, você descobre que actualiza um número menor de modelos do que teria no passado, porque a realidade é que os seus modelos não precisam de ser perfeitos para fornecer valor. Muito tempo e dinheiro é gasto a tentar manter os modelos e documentos em sincronização com o código fonte, tempo e dinheiro que poderiam ser melhor usados desenvolvendo um novo *software*. Esta prática e praticamente importante para uma *documentação agile.*

Formalize modelos de contracto

Os modelos de contractos são muitas vezes necessários quando um grupo externo controla um recurso de informação que o seu sistema requer, tais como base de dados e serviço de informação. O modelo de contracto é algo que ambas as partes deviam concordar mutuamente e mutuamente alterar com o decorrer do tempo, se necessário.

Exemplos de modelos de contractos incluem a documentação detalhada do API, a descrição do *layout* dum ficheiro e um modelo físico de dados descrevendo uma base de dados partilhada.

Tal como um contracto legal, um modelo de contracto normalmente requer o investimento significante de recursos para desenvolver e manter o contracto para garantir que é exacto e suficientemente detalhado. É, contudo, muito importante minimizar o número de modelos de contracto para que o seu sistema possa ir de acordo com o princípio *viaje com pouca bagagem.*

Descarte modelos temporários

A maior parte dos modelos que se criam durante o desenvolvimento são temporários ou de trabalho, modelos estes que já atingiram os seus objectivos e não trazem valor algum de decisão para a sincronização dos modelos. Se tiver este tipo de modelos, o melhor é descartá-los porque o investimento para actualizá-los não será recuperado pelo valor de os ter actualizado. Esta prática e praticamente importante para uma documentação ágil.

2.6.2.3 Práticas complementares a AM

As práticas que se seguem são complementares para AM, mas não são incluídas explicitamente como práticas.

Refactoring

Esta é uma prática de codificação na qual você faz pequenas alterações, chamadas *refactorings*, ao seu código para suportar novos requisitos, ou manter o seu desenho o mais simples possível. Para o ponto de vista da AM, esta prática garante que o código se mantém limpo e claro enquanto se codifica.

Test-First Design

Esta é uma prática de desenvolvimento em que você primeiramente considera e depois codifica um caso de teste antes de escrever o código de negócio que satisfaz este caso de teste. Para o ponto de vista da AM, esta prática força que você pense no seu desenho

antes de escrever o código, removendo a necessidade de uma modelação de desenho detalhada.

2.7 Como é que as práticas de AM se encaixam?

Para fazer AM funcionar efectivamente, é preciso ter conhecimento de como as práticas se encaixam. A que se segue descreve a relação entre as práticas da AM, organizadas em sete categorias. As primeiras quatro categorias *validação, Iterativo e incremental, trabalho em equipa* e *simplicidade* consolidam as práticas fundamentais da AM, as práticas que qualquer um que quer ser modelador ágil deve seguir. As práticas suplementares são consolidadas por *documentação, motivação* e *produtividade.*

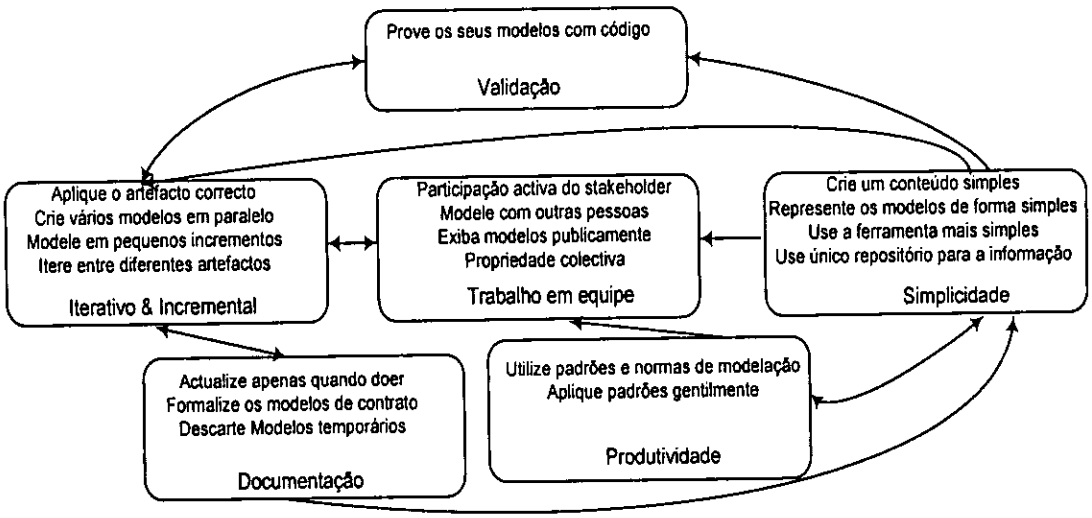

Figura 2 – Como é que as práticas de AM se encaixam (Fonte: Ambler, 2006e)

2.7.1 As práticas fundamentais

Na categoria **Trabalho em equipa**, a prática *Participação activa do stakeholder* é critica para o seu sucesso porque é para os *stakeholders* que se está a desenvolver o sistema, é este pessoal cujos requisitos são precisos entender e cumprir. Em outras palavras, é preciso trabalhar junto dos seus *stakeholders* algo que é suportado pela prática *Modele com outras pessoas*. A prática *Propriedade colectiva* reforça o trabalho em equipa porque quando uma única pessoa "é dona" dum modelo, essa pessoa se torna

uma "passagem estreita" para os esforços da sua modelação, enquanto que quando se permite que qualquer um trabalhe num modelo, pode-se facilmente trabalhar como uma equipa. A prática *Exiba modelos publicamente* torna fácil que o pessoal possa olhar para trás e adiante entre os modelos, considerando a informação que os modelos comunicam todos de uma vez, e, por isso, reforçam o esforço colaborador entre os elementos da equipa.

Na categoria **Iterativo e incremental,** a prática *Aplique o artefacto correcto* realça que cada artefacto, não interessa qual seja, tem os seus pontos fortes e as suas fraquezas e nenhum modelo único é suficiente para descrever a maioria dos aspectos do seu projecto tal como seus requisitos e arquitectura. Modeladores ágeis *criam vários modelos em paralelo* para colher uma gama maior de informação na captação de dados. Esta prática é claramente suportada pelas práticas *aplique o artefacto correcto* e *Itere entre diferentes artefactos* – você pode estar a capturar informação acerca de requisitos de uso num determinado caso de uso quando os *stakeholders* do seu projecto começam a discutir os requisitos para uma janela de edição. Iterar para trás e adiante ente os artefactos é suportado pela prática *Modele em pequenos incrementos* – você vai tipicamente trabalhar numa parte de um artefacto, depois noutro, depois noutro ainda, e assim sucessivamente.

Na categoria da **simplicidade** as práticas *Crie um conteúdo simples* e *Represente os modelos de forma simples,* frequentemente andam juntas durante a modelação – focando em como descrever algo de forma simples, modeladores frequentemente descobrem como fazer o que quer que seja que eles estejam a modelar. A prática *Use único repositório para a informação,* sugere que você tente guardar a informação apenas uma vez de preferência no local mais apropriado possível. Isto simplifica os esforços da sua documentação tal como a facilidade de "seguir" os documentos. A Simplicidade do processo é reforçada pela prática *Use a ferramenta mais simples.* A ferramenta mais simples é normalmente a mais fácil de usar, e, isto diminui as barreiras do trabalho nos seus modelos, e assim aumenta as chances que outras pessoas também o façam, incluindo os *stakeholders* do seu projecto. Com o uso de ferramentas mais simples, aumentam-se as chances de descrever os modelos de forma simples.

A categoria da **validação** consiste na prática *Prove o seu modelo com código.* Procurando provar um modelo com código o mais cedo possível, facilmente mostra-se que o sistema funciona.

2.7.2 As práticas suplementares

Na categoria da **documentação,** os requisitos do seu sistema, o seu entender dos requisitos e mesmo o seu entender da sua solução mudam durante o projecto (lembre-se do princípio **abrace mudanças**). Muitos dos artefactos do seu projecto, incluindo modelos e documentação precisarão de evoluir para reflectir as suas mudanças. Uma das melhores maneiras de garantir que você segue uma abordagem ágil para os seus modelos e documentação é *actualizando apenas quando dói.* Quando se segue esta prática e nota-se que um modelo não está sendo actualizado, é uma indicação de que o modelo não tem muito valor para a sua equipa, e se um modelo não tem valor ele deve ser descartado. Porém, lembre-se que modelos de contrato, modelos que definem uma *interface* entre o seu sistema e outro, são improváveis de mudar muito durante o tempo devido a sua importância e não são candidatos a serem descartados.

Finalmente, na categoria da **produtividade**, a agilidade segue padrões e *guidelines*, mesmo na modelação, porque ela promove consistência no seu trabalho. Certamente, você escreve os seus próprios *guidelines* e algumas vezes necessita de criá-los devido a alguns factores incomuns no seu ambiente, mas com um pouco de pesquisa na Internet você pode rapidamente encontrar *guidelines* de desenvolvimento, como por exemplo, os de java, divulgados em http://www.ambysoft.com/javaCodingStandards.html.

2.7.3 Como é que as categorias se relacionam

Considere as práticas da categoria **trabalho em equipa**. *Participação activa do stakeholder* é sustentada pelas práticas da categoria **simplicidade,** porque simplicidade reduz qualquer barreira para participação. Participação é também habilitada pelas práticas da categoria **Iterativo e Incremental**, particularmente criar vários modelos em paralelo porque isto abre mais oportunidades para os *stakeholders* se envolverem. As práticas *propriedade colectiva* e *modele com outras pessoas* são sustentadas pelas

práticas da Motivação, o facto de você precisar de entender ou comunicar um assunto frequentemente motiva as pessoas a trabalharem juntas, tal como as práticas da simplicidade, mais uma vez porque elas reduzem barreiras para participação. *Visualizar modelos publicamente* é realçada pelas práticas da produtividade, *seguir padrões e aplicar patterns* aumenta consistência e facilidade de leitura, e *reuso de recursos existentes,* tais como modelos comuns de arquitectura provê uma base familiar que os desenvolvedores podem usar nos seus modelos. *Propriedade colectiva* é sustentada pelas práticas da categoria **Iterativo e incremental**, em particular *criar vários modelos em paralelo* e *iterar entre diferentes artefactos* promove trabalho colectivo em qualquer modelo que seja apropriado num dado tempo.

As práticas da categoria **simplicidade** são sustentadas pelas práticas de várias outras categorias. A prática *represente os modelos de forma simples* é realçada por *utilize os padrões e normas de modelação* e *aplique padrões gentilmente* porque ambas as práticas sustentam modelação numa língua comum (os padrões por ti escolhidos e esperançosamente *patterns* bem percebidos). As práticas da categoria **Simplicidade** são realçadas pelas práticas da **documentação** – quando você actualiza apenas quando dói é mais provável que você *represente os modelos de forma simples* e *cria um conteúdo simples* porque você não estaria adicionando informação desnecessariamente aos seus modelos.

Agora, considere as práticas da categoria **Iterativo e Incremental**. As práticas da categoria **trabalho em equipa** claramente sustentam estas práticas, com muita gente envolvida há uma grande probabilidade de alguém saber qual é o artefacto correcto a aplicar para a sua situação tornando possível que você itere para tal quando necessário. As práticas da **validação** dar-te-ão a coragem para levar uma abordagem incremental, particularmente quando *você prova com código e mantendo testabilidade* em mente é mais provável que *trabalhe em vários modelos duma vez, e itere entre eles*, porque casos de teste vão, com certeza, precisar de ser efectuados numa variedade de casos. As práticas da **documentação** também promovem uma abordagem incremental, particularmente *actualizar apenas quando doer*. Embora *formalizar modelos de contracto* frequentemente vá contra grau incremental porque você quer garantir as interfaces com outros sistemas, o mais rápido possível. *Iterar entre diferentes artefactos*

e *descartar modelos temporários* são complementares porque você frequentemente quer trabalhar num modelo e depois mudar quando alcançar o seu propósito. As práticas da categoria **simplicidade** também são importantes para esta categoria. Quando se usa a ferramenta mais simples torna-se mais fácil de *iterar para trás e adiante entre os artefactos,* pois gasta-se pouco tempo para entender a ferramenta, e focando num *conteúdo simples* e uma *simples representação* assegura que se tenha uma curva mínima de aprendizagem lembrando o que o modelo comunica.

As práticas da **validação** são sustentadas pelas práticas da **simplicidade** – quando se *cria um conteúdo simples* e *representa os modelos de forma simples* torna-se mais simples de prová-los com código. As práticas da categoria **Iterativo e incremental** também promovem validação. Por exemplo, quando se *itera entre diferentes artefactos* é provável que se itere para a codificação, de tal forma que se possa mostrar que o modelo funciona.

As práticas da **produtividade** são realçadas pelas práticas da **simplicidade**: é mais fácil *aplicar padrões gentilmente* se estiver a trabalhar com modelos simples; é mais fácil *aplicar padrões de modelação* quando se *representa os modelos de forma simples*; e é mais fácil ainda *reusar recursos existentes,* tais como modelos de arquitecturas comuns quando esses modelos são *simples* e fáceis de perceber.

As práticas da **documentação** são sustentadas por ambas as práticas das categorias **simplicidade** e **Iterativo e incremental**. Quanto mais *simples for a sua documentação,* mais fácil será trabalhar com ela – se a sua documentação é fácil de perceber, isso dá-lhe a coragem de *actualizar apenas quando doer* porque você sabe que será capaz de fazê-lo de forma fácil. Documentação que é difícil de entender é um grande risco para o seu projecto porque você não pode ter certeza que poderá actualizá-la o necessário. As práticas: *actualize apenas quando doer* e *descarte modelos temporários,* claramente funcionam em ambientes promovidos por práticas como *itere entre os diferentes artefactos* e *modele em incrementos pequenos.*

2.8 Documentação Ágil

Alguns modelos ágeis "transformam-se" em documentação oficial do sistema, se bem que a maioria não, é, portanto relevante uma discussão de como ser ágil neste processo.

A seguir são apresentados um conjunto de pontos críticos a ter em conta na criação de documentação:

1. A questão fundamental é **comunicação**, não documentação;
2. Documentação deveria ser "magra e efectiva";
3. Viaje com pouca bagagem, o quanto possível;
4. Documentação deveria ser apenas boa o suficiente;
5. Documentação completa não garante sucesso do projecto, pelo contrário aumenta a probabilidade de insucesso;
6. Modelos não são necessariamente documentos e documentos não são necessariamente modelos;
7. Documentação é tanto parte do projecto, tanto quanto código fonte;
8. O benefício de ter documentação deve ser maior que os custos da criação e manutenção;
9. Cada sistema tem a sua própria e única necessidade de documentação;
10. Pergunte sempre se precisar da documentação e não se quer a documentação;
11. Crie documentação apenas quando precisar;
12. Actualize a documentação apenas quando doer;

2.8.1 *Relação entre modelos documentos e código fonte*

A figura que se segue descreve a relação entre os modelos documentos e código fonte.

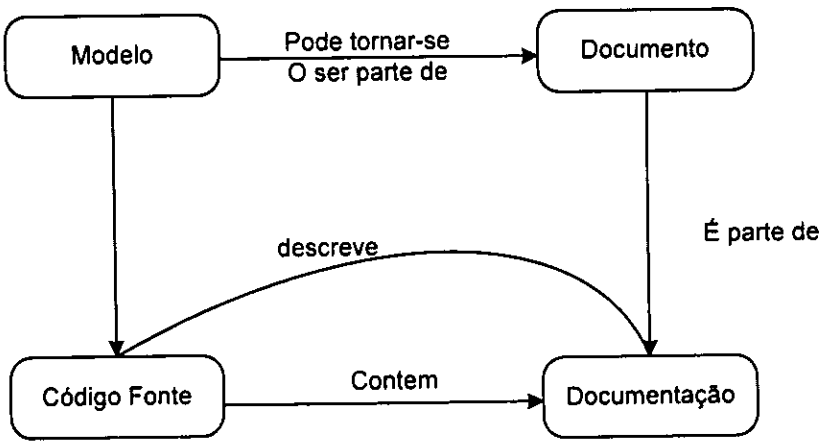

Figura 3 – A relação entre Modelo, Documento, Código Fonte e Documentação (Fonte: Ambler, 2006f)

Do ponto de vista da AM, documento é um artefacto para código fonte cujo propósito é transmitir informação, isto é diferente do conceito de modelos que é uma abstracção que descreve um ou mais aspectos de um problema ou uma solução possível de um problema, portanto, um *modelo* pode tornar-se ou ser parte de um *documento* e pode ser usado para descrever o *código fonte.* Código fonte contém várias linhas de código, e, portanto, contém documentação, e, por conseguinte a documentação descreve o código fonte.

2.8.2 *Razões para a criação de documentos*

Desenvolvedores ágeis reconhecem que documentação é uma parte real de qualquer sistema. Ambler (2006f), define quatro razões válidas para a criação de documentos:

1. **Os *stakeholders* do projecto precisam** – se os *stakeholders* do seu projecto solicitam um documento, talvez para comunicação do seu entendimento do compromisso, será necessário criar um documento;

2. **Para definir modelos de contracto** – modelos de contracto definem como o seu sistema interage com outros sistemas externos, e são frequentemente necessários quando um grupo externo controla uma informação de recurso para o seu sistema;

3. **Para suportar comunicação com um grupo externo** – quando se precisa de trabalhar com um grupo externo é necessário criar formas de comunicação com o grupo, e partilhar a documentação é frequentemente parte da solução em combinação com encontros ocasionais cara-a-cara, teleconferências, e-mail, entre outras.

4. **Para acompanhar o pensamento de algo** – O acto de escrita das suas ideias em papel, ajuda a solidificar as suas ideias e descobrir problemas no seu pensamento.

Existem muitos factores que frequentemente levam a criação de documentação excessiva e "roubam" tempo de programação podendo resultar em fracasso do projecto. É o caso de criação de documentos devido ao processo em uso, criação de documentos apenas para os chefes do projecto acompanharem o estágio do projecto.

2.8.3 Quando é que um modelo se torna permanente

Durante o desenvolvimento, os documentos surgem a partir de ideias, e uma vez, que estas ideias se tornem consistentes, estes modelos transformam-se em temporários, ou descartados quando as ideias são abandonadas. Os modelos temporários são, por sua vez, actualizados e, em muitos casos, descartados uma vez que tenham satisfeito os objectivos da sua criação. No entanto, estes modelos temporários podem evoluir e se tornarem modelos permanentes.

Segundo Ambler (2006f), os modelos evoluem para se tornar permanentes quando:

- Existe uma razão clara e válida para torná-lo permanente;
- Existe uma audiência para a qual o modelo tem valor;
- Quando os *stakeholders* estão dispostos a investir para torná-lo documentação.

Os dois primeiros pontos são guiados pelo princípio *modele com propósito*: é preciso ter uma razão válida para criar modelos e, portanto, os documentos também precisam de um propósito para a sua criação. O terceiro ponto é guiado pelo princípio *maximize o investimento do stakeholder* e pelo conceito de serem os recursos do *stakeholder* a serem investidos, e eles devem ser quem diz como estes recursos são investidos, bem ou mal.

2.8.4 Quando é que um documento é Ágil?

Ambler (2006f), define um documento como ágil quando segue os critérios que se seguem.

1. **Documentos ágeis maximizam o investimento do *stakeholder*.**

2. **Documentos ágeis são "magros e suficientes"**. Eles contêm apenas informação suficiente para alcançar o seu propósito, em outra palavras, são o mais simples possível.

3. **Modelos ágeis alcançam um propósito**.

4. **Documentos ágeis descrevem "coisas boas a saber"**. Eles capturam informação crítica, informação que não é óbvia como desenho relacional, requisitos, procedimentos de uso ou procedimentos operacionais.

5. **Documentos ágeis têm um cliente específico e facilitam esforços de trabalho para esse cliente**. Documentação de sistema é tipicamente escrita para desenvolvedores de manutenção, provendo uma visão geral da arquitectura e uma descrição resumida dos requisitos e decisões de desenho. Documentação de utilizadores frequentemente inclui secções de como usar o sistema e são escritas em linguagem que os utilizadores possam perceber. Documentação operacional descreve como correr o sistema e é escrita numa linguagem que o pessoal do nível operacional perceba. Portanto, para clientes diferentes, temos tipos de documentos diferentes e linguagem diferente.

6. **Documentos ágeis são suficientemente precisos, consistentes e detalhados.** Documentos ágeis não precisam de ser perfeitos, apenas precisam de ser suficientemente bons.

2.8.5 Tipos de documentos a criar no seu projecto

A tabela que se segue, apresenta os potenciais documentos, sugeridos por Ambler (2006f), a serem criados pela sua equipa de desenvolvimento.

Documento	Audiência	Descrição
Modelos de contracto	Outras equipas	Um documento descrevendo o interface técnico para o sistema ou parte do sistema.
Decisões de desenho	Desenvolvedores, desenvolvedores de manutenção, gestores do projecto	Resumo de decisões críticas pertencentes a desenho e arquitectura que a equipa fez durante o tempo do desenvolvimento.
Visão geral executiva	Gestão sénior, gestão de utilizadores, gestão do projecto	Uma definição da visão do sistema e um resumo do custo corrente estimado e marcos calendarizados.
Documentação operacional	Pessoal operacional	Tipicamente incorpora uma indicação das dependências do sistema, a natureza da sua interacção com outros sistemas, bases de dados e ficheiros, referências para procedimentos de cópia de segurança; uma lista de pontos do contracto para o seu sistema e como alcança-los; um resumo dos requisitos de disponibilidade/segurança; uma indicação do perfil do sistema; uma guia de solucionamento de problemas.
Visão geral do projecto	Desenvolvedores, gerentes, desenvolvedores de manutenção, e pessoal operacional	Um resumo de informação crítica como visão do sistema, primeiros contactos do utilizador, tecnologias e ferramentas usadas para fazer o sistema, e processo crítico de operação (alguns aplicáveis a desenvolvimento, como por exemplo, como fazer o sistema e outra aplicáveis para a produção, como por exemplo, como fazer cópias de segurança). Também provê informação de artefactos críticos do projecto, tais como, código fonte, onde os modelos permanentes do sistema estão e onde outros documentos estão.
Documento de	Desenvolvedores,	Este documento define o que o sistema vai fazer,

requisitos	desenvolvedores de manutenção, utilizadores, gestores de utilizadores	resumindo artefactos de requisitos, tais como definições de regras de negócio, casos de uso, ou histórias dos utilizadores, ou alguns protótipos de *user-interface*.
Documentação de suporte	O pessoal do suporte	Esta documentação inclui material de treino específico para o pessoal do suporte; toda a documentação do utilizador para usar quando estiver a solucionar problemas; uma guia de solucionamento de problemas; procedimentos escalados para lidar com problemas difíceis; e uma lista de pontos de contacto dentro da equipa de manutenção.
Documentação do sistema	Desenvolvedores de manutenção, desenvolvedores	O propósito desta documentação é de prover uma visão geral do sistema e ajudar a perceber o sistema. Informação neste documento inclui uma visão geral da arquitectura técnica, arquitectura de negócio, requisitos de alto nível. Modelos de desenho e arquitectura detalhada ou referência a eles também pode ser incluída onde apropriada.
Documentação do utilizador	Utilizadores, gestores de utilizadores	Este documento contém um manual de referência, guião de uso, guião de suporte e até mesmo material de treino. É importante distinguir bem estes documentos, porque a maneira de uso de cada um varia: uns são apenas para consultas rápidas, outro é para descobrir como trabalhar com o sistema, outro ainda é para como obter uma ajuda adicional e o último é para treino.

Tabela 3 – Documentos potenciais a serem criados pela sua equipa de desenvolvimento (Fonte: Ambler, 2006f)

2.8.6 Estratégias para aumentar a agilidade da documentação

1. **Concentrar no cliente.** Identifique o potencial cliente para o seu documento.
2. **Manter os modelos suficientemente simples, mas não muito simples**. Siga as práticas, use ferramentas simples, crie conteúdo simples e represente os modelos de forma simples, quando estiver a criar a documentação.
3. **O cliente determina a suficiência de meios.**
4. **Documente com um propósito.** Deve-se criar documentos apenas quando estes satisfazem um objectivo claro, importante, de valor imediato ao sistema.

5. **Dê preferência a comunicação em torno da documentação.** O objectivo principal da documentação é prover comunicação efectiva com a audiência do documento. O importante não é criar um manual de utilizador com funcionalidades que os utilizadores não vão usar, o importante é garantir boa comunicação.
6. **Ponha a documentação no local apropriado.** É preciso colocar a documentação num local que tanto o cliente como qualquer elemento da equipa possa aceder com facilidade. Esta estratégia é guiada pela prática *Use único repositório para a informação.*
7. **Espere.** Adie a criação de todo documento para o mais tarde possível, o documento deve ser criado imediatamente antes de o precisar.
8. **Exiba modelos publicamente.**
9. **Comece com modelos que costumam estar actualizados.**
10. **Exija a sua equipa a justificar os pedidos de certos documentos.**
11. **Crie o menor número de documentos com menor sobreposição.** É preciso evitar que dois documentos falem sobre o mesmo assunto.
12. **Coloque na sua equipa alguém com experiência de escrita.**

3. O SISTEMA DE EMISSÃO DE DECLARAÇÕES E CERTIFICADOS

3.1 Sistema Actual

No final de cada semestre lectivo, cada faculdade da UEM deve enviar, à Direção de Registo Académico (DRA), actas do aproveitamento de cada estudante. A DRA, por sua vez, deve actualizar a sua base de dados sobre o progresso do estudante.

Este Processo de envio dos dados por parte das faculdades e actualização das notas por parte da DRA não funciona devidamente. As faculdades não são regulares no envio dos dados, nem sempre os enviam a tempo e, por outro lado, a DRA não dispõe de capacidade humana suficiente para processar a informação. Portanto, cada faculdade vai enviando as actas e a DRA arquiva as *(Frederico Guirugo)*.

3.1.1 Descrição do sistema actual

Quando um estudante ou qualquer outra pessoa que tenha frequentado um curso na UEM está interessado em obter uma declaração, um certificado ou um diploma, ele dirige-se a DRA,. E caso o documento a solicitar seja declaração, preenche um formulário, caso o documento a solicitar seja certificado ou diploma, redige um requerimento dirigido ao reitor da UEM.

Tanto no requerimento assim como no formulário, o solicitante indica seu curso, nome completo, número de estudante e o motivo para o qual solicita o documento.

No acto da entrega do formulário ou requerimento, consoante seja o documento a solicitar, o solicitante deverá fazer o pagamento referente ao pedido, e recebe em troca um recibo com o seu nome, documento solicitado e data da solicitação. A partir desse momento, o estudante deverá aguardar dez dias para levantar o documento.

A DRA envia, então, uma carta à faculdade solicitando as notas do estudante, e esta, por sua vez, consulta a ficha do estudante e envia uma cópia à DRA.

Em alguns casos o solicitante pode solicitar as notas no seu departamento e levar a DRA para acelerar o processo.

Uma vez recebido o documento do aproveitamento do estudante, enviado pela faculdade, a DRA analisa os dados (cadeiras feitas) para apurar o nível do estudante. Depois de apurado o nível do estudante, o processo é enviado a área de digitação. Esta já possui um molde de documentos (declaração, certificado, diploma) e, portanto, apenas altera os dados do último documento emitido colocando os dados do novo estudante (daí, em alguns casos, aparecerem alguns erros).

Depois de concluída a digitação é feita a impressão do documento em duas cópias, que seguem à direcção para se proceder a assinatura. Após a assinatura das cópias, estas são arquivadas numa pasta de acordo com o curso. É de salientar que, em alguns casos, o digitador se esquece de mudar o curso, o que torna difícil a sua localização quando o solicitante se dirige a DRA para levantar o seu documento.

Depois de dez dias, o solicitante dirige-se, então, à DRA para levantar o seu documento. Para o efeito, ele apresenta, à recepcionista, o recibo do pagamento, e esta procura a documento de acordo com o recibo.

Uma vez localizado o documento, o solicitante confere, a pedido da recepcionista, os dados nele constantes. Caso a declaração esteja correcta ele assina a cópia, levanta a declaração e deixa o recibo com a DRA. Caso não esteja correcta, o solicitante apresenta a sua reclamação, que tem sido, na maioria dos casos de nome mal escrito, curso errado, nível errado, entre outros.

Caso o documento não seja localizado, o funcionário procura informar-se se o processo já deu andamento, e se o documento já foi emitido. Se a resposta for positiva, a recepcionista volta a procurar em todas as pastas sem atender ao curso, para verificar se não terá sido arquivada na pasta errada. Caso não encontre o documento, pede-se ao solicitante que volte depois de alguns dias (o número de dias a esperar depende do

guiché, não existe regra para tal, pode ser depois de um, dois, três ou até mesmo dez dias).

3.2 Sistema Proposto

Propõe-se um sistema novo informatizado, baseado em tecnologias *Web*, onde o estudante, a partir de qualquer ponto do mundo, possa requisitar sua declaração ou certificado, bastando, para tal, autenticar-se e preencher um formulário de pedido do documento (declaração ou certificado).

Caso o solicitante não tenha acesso a Internet, ele poderá dirigir-se a DRA e solicitar o documento, através da recepcionista.

Depois da confirmação do pedido do documento, o solicitante recebe o número do pedido que poderá usar para apresentar à DRA, caso seja solicitado na recepção do documento. A partir deste momento, o solicitante deverá dirigir-se a DRA, no dia seguinte, para proceder o pagamento e levantar o documento.

No sistema proposto, os pagamentos serão feitos na altura da recepção do documento, portanto, vários estudantes poderão solicitar documentos e nunca procederem o levantamento, o que resultaria num prejuízo para a DRA. Portanto, o sistema registará os pedidos não levantados de tal modo que, no futuro, este estudante seja penalizado quando precisar de um novo documento, ou qualquer outro serviço da reitoria.

3.2.1 Acesso ao sistema

Todo o estudante e todo chefe do registo académico de cada departamento, poderá ter acesso ao sistema. Para tal ele deverá dirigir-se a DRA para receber o seu nome de usuário e senha (que deverá alterar a primeira vez que entrar no sistema).

3.2.2 Boas razões para adopção deste sistema

1. Com este sistema, as faculdades não precisarão mais de enviar à DRA, as actas do aproveitamento do estudante, precisarão apenas de um computador conectado a Internet para registrar o aproveitamento de cada estudante, reduzindo, desta forma, a carga de trabalho para a DRA em actualizar as notas de cada estudante e poupando recursos à universidade no processo, uma vez que já não precisará de enviar actas do aproveitamento de cada estudante por semestre.

2. O sistema proposto reduz a burocracia no processo de solicitação de declarações e certificados, uma vez que o nível do estudante será calculado automaticamente e, não se precisará mais de solicitar, à faculdade, as notas do estudante, o que tornará o processo mais rápido, não precisando mais de dez dias, reduzindo-se para um dia apenas.

3. O sistema também poderá ser usado para controlar as entradas de dinheiro de pedidos de documentos, uma vez que ele regista que dos pedidos feitos, quais os que foram levantados.

4. MODELAÇÃO DO SISTEMA USANDO AM

Modele sempre com um propósito, descartando modelos temporários e tendo sempre em conta de que conteúdo é mais importante do que representação (Ambler, 2002).

A modelação do sistema de emissão de declarações e certificados da UEM (SEDC), foi feita usando uma abordagem orientada a objectos – a *Unified Modeling Language(UML*[21]*)* e seguindo os princípios e práticas da metodologia *Agile Modeling*.

Um dos valores da *Agile Aliance* (ANEXO A), é *"um software funcional em vez de extensa documentação"*, portanto não devemos nos preocupar em criar muitos documentos, muitos modelos, o importante é modelar o suficiente para garantir comunicação efectiva e produção de um software funcional e que alcança as necessidades do utilizador.

Neste capítulo se apresenta a modelação do sistema de Emissão de declarações e certificados usando, usando diagramas UML com base em alguns conceitos do *Rational Unified Process* - RUP.

4.1 RationalUnified Process

O Rational Unified Process (RUP) é uma metodologia completa criada pela *Rational Software Corporation* para garantir que grandes projectos de software sejam bem sucedidos (Boering, 2003).

Esta metodologia oferece uma abordagem baseada em disciplinas para atribuir tarefas e responsabilidades dentro de uma organização de desenvolvimento. Sua meta é garantir a produção de software de alta qualidade que atenda às necessidades dos utilizadores dentro de um cronograma e de um orçamento previsível.

[21] ***Unified Modeling Language*** (UML) é uma linguagem padrão para especificação, visualização, construção, e documentação de artefactos de um sistema de software, tanto para modelação de negócios como para outros sistemas não baseados em software (BRAUN, 2001a).

A Rational (2003), conforme mostra a figura que se segue, divide o RUP em duas dimensões:

- O eixo horizontal representa o tempo e mostra os aspectos do ciclo de vida do processo à medida que se desenvolve. Representa ainda o aspecto dinâmico do processo, sendo expresso em termos de fases, disciplinas e marcos.
- O eixo vertical representa as disciplinas, que agrupam as actividades de maneira lógica, por natureza. Representa ainda o aspecto estático do processo sendo descrito em termos de componentes, disciplinas, actividades, fluxos de trabalho, artefactos e papéis do processo.

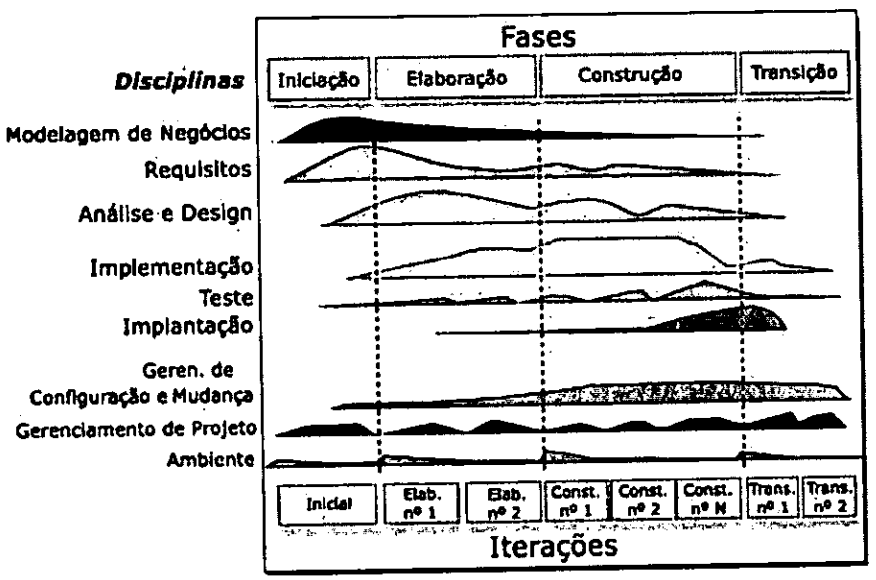

Figura 4 – Arquitectura geral da RUP (Fonte: Luiz, 2004)

4.1.1 Fases do RUP

A Rational (2003), no trabalho ***Rational Unified Process*** divide um projecto em quatro fases (Anexo B) diferentes, distinguidas na figura acima.

A primeira fase, a ***iniciação,*** enfatiza o escopo do sistema; a segunda, a ***elaboração,*** foca-se na arquitectura do sistema; a fase seguinte, denominada ***construção,*** tem o seu foco no desenvolvimento, e, por fim, a quarta fase, a ***transição,*** é a fase de ênfase na implantação.

4.1.2 Princípios do RUP

Luíz (2004), no seu trabalho ***Obtendo Qualidade de Software com o RUP,*** diz não existir uma maneira exacta de aplicar o RUP, pois ele pode ser aplicado de várias formas e será diferente em cada projecto e organização. Porém, existem alguns princípios que podem caracterizar e diferenciar o RUP de outros métodos iterativos:

- Atacar os riscos cedo e continuamente;
- Certificar-se de entregar algo de valor ao cliente;
- Focar no software executável;
- Acomodar mudanças cedo;
- Liberar um executável da arquitectura cedo;
- Construir o sistema com componentes;
- Trabalhar junto como uma equipa;
- Fazer da qualidade um estilo de vida, não algo para depois.

4.2 Âmbito do SEDC

Pretende-se desenvolver um sistema de informação para a gestão do processo de emissão de declarações e certificados na DRA, que permite que um estudante possa efectuar o pedido de documento via Internet e que o pedido deste documento seja emitido por um funcionário da reitoria.

4.3 Requisitos do SEDC

Segundo Sommerville (1995), requisito dum sistema é uma descrição abstracta da funcionalidade do sistema e do regulamento sobre o qual o mesmo deve funcionar. Estes podem ser funcionais e não funcionais.

4.3.1 Requisitos Funcionais

Requisitos funcionais descrevem o que o sistema faz ou é esperado que faça. Estes são os requisitos que inicialmente serão levantados, abrangendo a descrição dos processamentos a efectuar pelo sistema (Nunes, 2001).

O sistema proposto deverá:

- Possuir uma tela de autenticação onde o utilizador introduz o seu nome de usuário e a sua senha, para ter acesso ao sistema. E, após a autenticação, o utilizador só deve ter acesso à informação de acordo com o seu nível de acesso;
- Permitir que um estudante, a partir de qualquer parte do mundo, possa efectuar pedido de um documento;
- Calcular o nível do estudante automaticamente, tendo em conta o regulamento pedagógico em vigor na UEM;
- Permitir a emissão dos documentos através de opções simples de menu;
- Registar e marcar estudantes que fizerem pedidos de documentos e não procederem o seu levantamento;
- Ter uma tela para criação de turmas;
- Permitir o registo das notas no final de cada semestre a partir da faculdade;
- Gerar relatórios de pedidos por tipo de documentos (declarações e certificados);
- Gerar relatórios de documentos emitidos levantados e não levantados;

4.3.2 Requisitos Não funcionais

Os requisitos não funcionais estão relacionados com características qualitativas do sistema, descrevendo a qualidade que o sistema deverá fornecer os requisitos funcionais, abrange mediadas de desempenho, como, por exemplo, tempo de resposta, volume de dados e considerações de segurança (Nunes, 2001).

Contexto de Implantação

O sistema deve ser baseado numa aplicação *Web*, que deve ser acedida a partir de qualquer ponto do mundo, e as tecnologias a serem usadas são as seguintes:

- Servidor Web – Tomcat[22];
- Páginas de Internet – *Java Server Pages*[23] (JSP);
- Sistema de gestão de base de dados – MySQL[24];
- Todos os clientes deverão ter sistema de operativo *Windows* ou Linux/Unix, e um *browser* de Internet.

Segurança

O sistema deve ser seguro, de modo a garantir que só indivíduos autorizados possam aceder ao sistema, e garantir ainda, a transmissão segura da informação.

Usabilidade

A usabilidade, refere do ponto de vista de utilizador, como o sistema é utilizável (Cato, 2001). A definição de usabilidade está centrada nos atributos do produto.

O sistema deve ser de:

- Fácil Memorização – O sistema deve ser fácil de usar, de modo a que um utilizador possa voltar a usar sem ter que voltar a aprender tudo de novo.
- Fácil Aprendizagem – O sistema deve ser fácil de se aprender, para que o utilizador possa usar sem grandes problemas.

[22] ***Tomcat*** é um servidor de aplicações Java para web. É distribuído como software livre e desenvolvido como código aberto dentro do conceituado projecto Apache Jakarta e oficialmente endossado pela Sun como a Implementação de Referência (RI) para as tecnologias *Java Servlet e Java Server Pages* (WIKIPÉDIA, 2006a).
[23] ***Java Server Pages*** (JSP) é uma tecnologia para desenvolvimento de aplicações WEB, que permite acesso à base de dados, ficheiros e funciona em qualquer sistema operativo (Telmaco, 2004).
[24] ***MySQL*** é um sistema de gestão de base de dados, que suporta qualquer sistema operativo e utiliza, como interface, a linguagem *Structured Query Language* (SQL) - Linguagem de Consulta Estruturada). É actualmente um dos bancos de dados mais populares, com mais de 4 milhões de instalações pelo mundo (WIKIPÉDIA, 2006a).

- Redução de Erros – O sistema deve ter uma baixa taxa de erros, para que o utilizar se sinta a fazer progressos positivos.
- Satisfação Subjectiva – O sistema deve ser agradável de se usar, para que os utilizadores se sintam subjectivamente satisfeitos quando estiverem a utilizar.
- Utilização Eficaz – Uma vez que os utilizadores aprenderam a usar o sistema, eles deveriam utilizar o mesmo de forma eficaz.

4.4 Diagrama de casos de uso

Os *diagramas de casos de uso* constituem a técnica em UML que serve para representar o levantamento de requisitos e assegurar que tanto o utilizador final como o perito numa determinada área ou o especialista informático possui um entendimento comum dos requisitos (Nunes, 2001).

O seu objectivo é mostrar o que o sistema deve efectuar e não como vai fazer. Para tal, eles usam as seguintes abstracções: *actores, casos de uso e relações.*

Actores que representam utilizadores do sistema ou outros sistemas que interagem com o sistema em modelação (BRAUN, 2001b);

Caso de uso que é uma técnica de modelação usada para descrever um conjunto de cenários que descrevem a interacção entre o utilizador e o sistema;

Relações que representam a informação de quais os actores estão associados a que casos de uso (Nunes, 2001).

Os actores identificados para este sistema são: estudante, funcionário da reitoria, funcionário da faculdade e o Administrador do sistema.

4.4.1 Casos de uso e actores

A tabela que se segue representa a relação entre casos de uso e actores.

Casos de uso	Actores
Efectuar o Login	Todos
Efectuar pedido de declaração/certificado	Estudante, funcionário da reitoria
Consultar dados de um estuante	Todos
Cancelar pedido	Estudante, funcionário da reitoria
Analisar pedido	Funcionário da reitoria
Devolver pedido	Funcionário da reitoria
Imprimir documento	Funcionário da reitoria
Assinar documento	Funcionário da reitoria
Arquivar documento	Funcionário da reitoria
Tomar conhecimento da emissão do documento	Estudante
Entregar documento	Funcionário da reitoria
Pesquisar pedidos	Funcionário da reitoria
Manter director (a) da DRA	Funcionário da reitoria
Mater usuários	Administrador
Criar turma	Funcionário da faculdade
Registar notas dos estudantes	Funcionário da faculdade
Alterar sua senha	Todos
Criar utilizadores	Administrador

Tabela 4 – Representação da Relação caso de uso – actor.

A figura que se segue ilustra a comunicação entre os casos de uso e actores através do diagrama de *Use Cases* do SEDC. Os detalhes dos casos de uso podem ser observados no Anexo C.

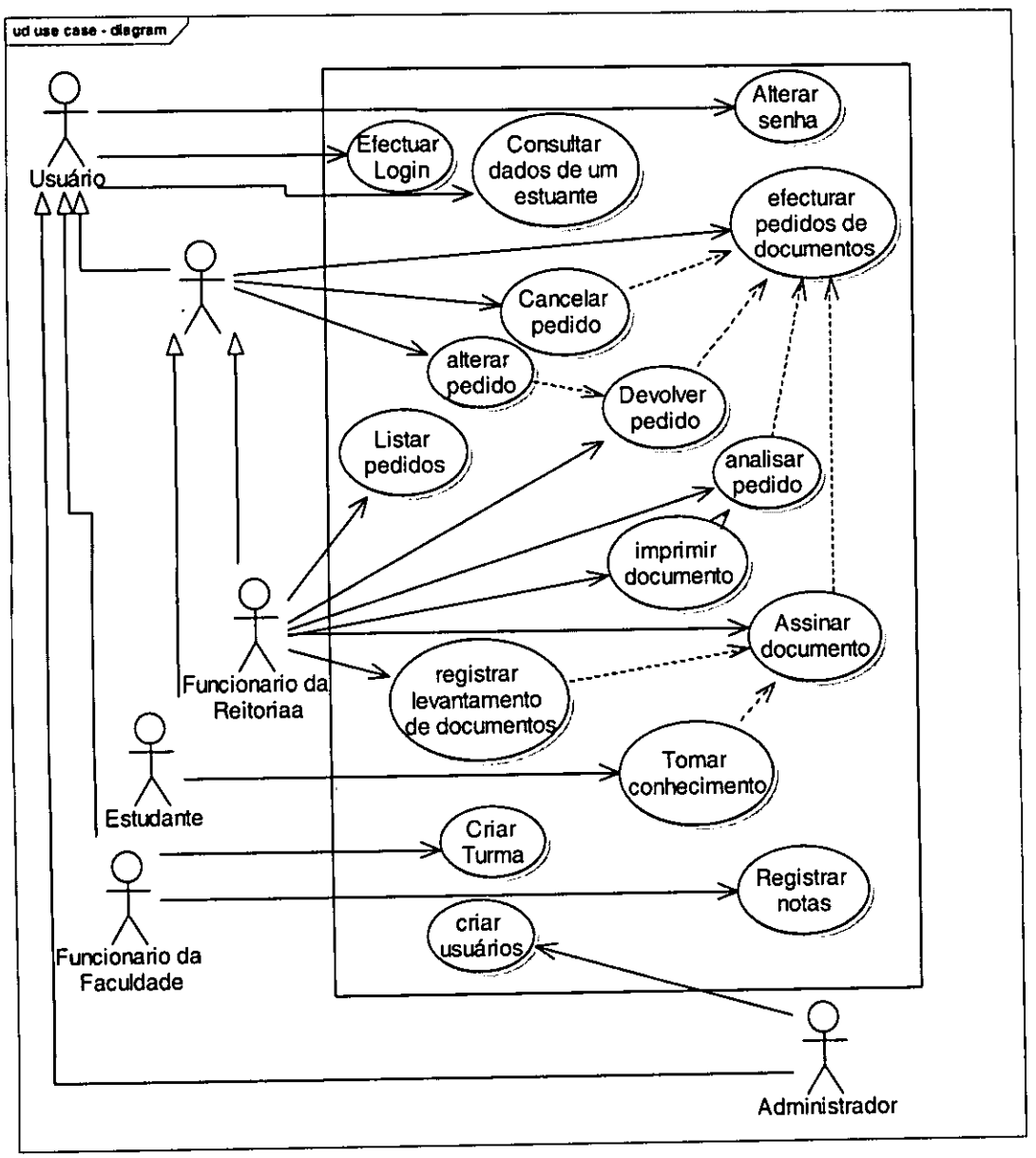

Figura 5 – Diagrama de casos de uso do SEDC

4.5 Diagrama de classes

Diagrama de classes é a espinha dorsal de quase todos os métodos orientados a objectos, incluindo UML. Eles descrevem a estrutura estática de um sistema (Smartdraw, 2006).

Diagramas de classes são largamente usados para descrever os tipos de objectos num sistema e seu relacionamento.

A criação de um modelo de classes resulta de um processo de abstracção através do qual se identificam os ***objectos*** (entidades e conceitos) relevantes no contexto que se pretende modelar e a descrição das suas características comuns em termos de *propriedades* (atributos) e *comportamento* (operações) e identidade.

Propriedades representam informação específica relacionada a uma classe de objecto. São as características dos objectos que as classes representam. Ex: nome do estudante, número de estudante, etc. (Macoratti, 2006)

comportamentos são acções que os objectos de uma classe podem realizar. Ex: corrigir o nome, inserir notas do estudante

Identidade permite identificar um objecto em particular como único num conjunto de objectos semelhantes

Assim, as **classes** representam uma abstracção sobre um conjunto de objectos que tem a mesma estrutura e comportamento. Na prática um objecto é um caso particular de uma classe, também referido como uma instância de uma classe (Nunes, 2001).

A figura que se segue representa o diagrama de classes do SEDC:

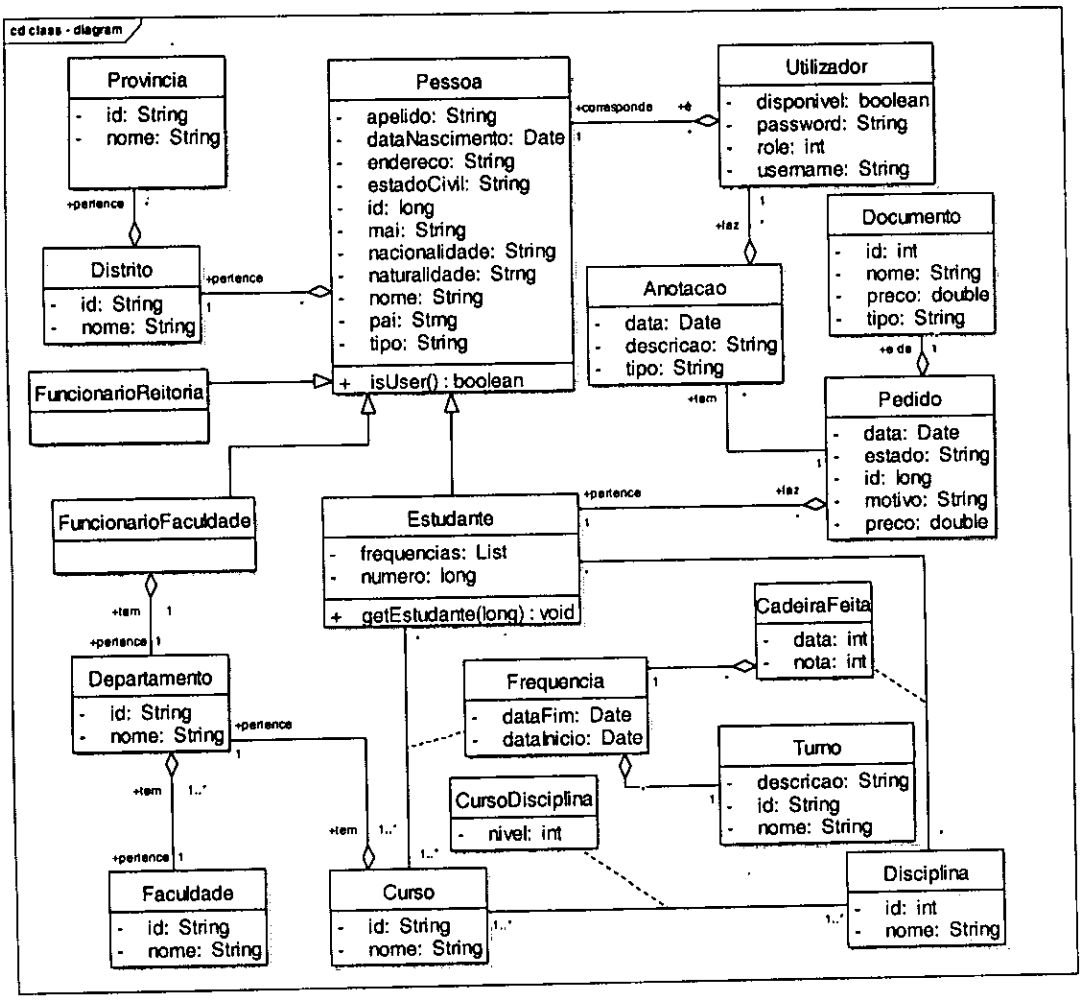

Figura 6 – Diagrama de classes do SEDC.

4.6 Diagrama de Actividades

O diagrama de actividades constitui um elemento de modelação simples mas eficaz para descrever fluxos de trabalho numa organização, a sequência de actividades de um sistema ou para detalhar as operações de uma classe, incluindo comportamentos que possuam processamento paralelo.

O diagrama de actividades do processo de requisição e emissão de documentos está representado no Anexo D.

4.7 Diagrama de Interacção

Diagrama de interacção é uma designação genérica que, em UML, se aplica a diagramas de sequência e de colaboração: os diagramas de sequência mostram a troca de mensagens entre os vários objectos, numa situação específica delimitada no tempo, enquanto os diagramas de colaboração descrevem as mesmas interacções mas centradas nos objectos intervenientes.

Os diagramas de interacção[25] são utilizados na UML para modelar os aspectos dinâmicos do sistema em termos dos objectos e suas relações, com base nas mensagens trocadas entre objectos (Nunes, 2001).

Para mais detalhes dos diagramas de Interacção, observe o Anexo E, que apresenta os diagramas de sequência dos processos de pedido e emissão de um documento no SEDC.

4.8 Diagrama de estados

Os diagramas de estados mostram os diferentes estados de um objecto durante a sua vida, bem como os estímulos que fazem com que o objecto mude o seu estado (Hensgen, 2001).

Na modelação de um sistema deve-se criar um diagrama de estado, somente para cada classe de objecto que tenha um comportamento dinâmico relevante.

Para mais detalhes, veja o diagrama de estados do objecto Pedido no Anexo F.

4.9 Desenho de Base de Dados

A figura que se segue representa o desenho da base de dados do SEDC. No CD em anexo, pode ser observado com mais detalhes.

[25] **Interacção** – segundo Booch et al (1999) é o comportamento que consiste na troca de um conjunto de mensagens entre objectos, dentro de um contexto, para atingir um objectivo.

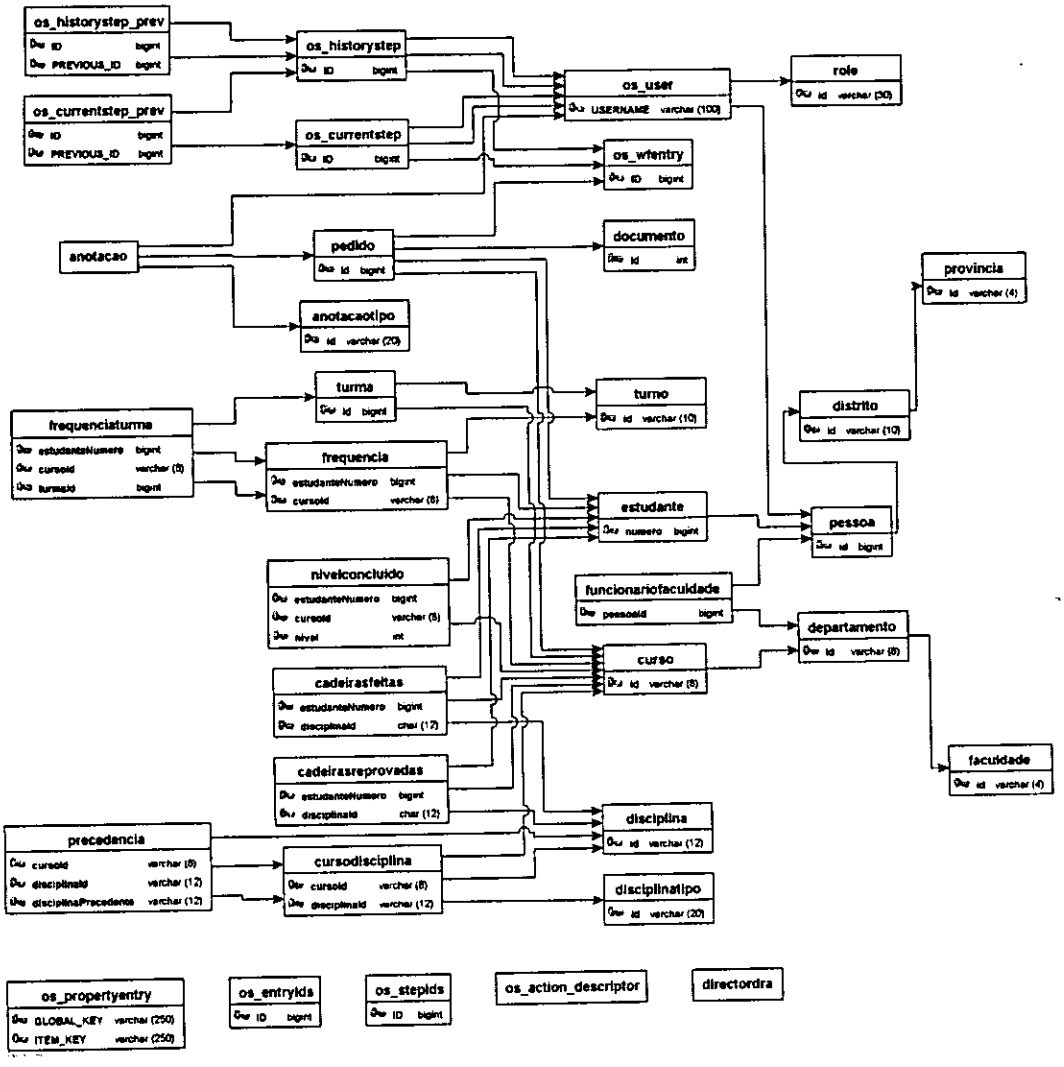

Figura 7 – Modelo Relacional de Base de Dados com Mapeamento de Tabelas.

4.10 Arquitectura de Implementação Usada

A qualidade de um software não se deve apenas ao processo de desenvolvimento usado, mas depende bastante da arquitectura do software (Eeles, 2006).

Whitehead (2001) defende que uma boa arquitectura também ajuda a reduzir stress e tensão na equipa.

Eeles (2006) define Arquitectura como sendo a organização fundamental do sistema que concerne nos seus componentes[26], a relação entre eles e o ambiente e os princípios que guiam seu desenho e evolução.

Segundo Morisseau-Leroy (2001), a arquitectura dum modelo de programação JSP depende das necessidades da aplicação a ser desenvolvida. Porém, como os requisitos da maioria das aplicações evolui com o tempo, aconselha-se o uso duma arquitectura que permita o aprimoramento modular e flexível da funcionalidade no futuro, embora ela possa parecer demasiadamente complexa para as suas necessidades do momento.

Para a concepção do protótipo do Sistema de Emissão de declarações e certificados (SEDC) usou-se a tecnologia *Java Server Pages* (JSP) para produzir conteúdo dinâmico das páginas *Web*.

O uso desta tecnologia é possível usando o código Java embutido nas páginas JSP, os chamados *Scriptlets*[27], ou, usando *frameworks* que facilitam a divisão da aplicação em camadas, evitando a mistura do código java e html[28], e facilitando, desta forma, a integração das diferentes camadas na aplicação.

Os *scriplets* são inconvenientes por misturar lógica com apresentação e quebrar a separação de papéis desenvolvedor/*Web designer*, o que torna as páginas de difícil percepção e manutenção.

Para contornar estas situações indesejadas, o protótipo desenvolvido usa uma estrutura *Model View Controler* 2 (MVC-2), uma variante de MVC para web.

A arquitectura *Model View Controller* ou Modelo-Visão-Controlador é um padrão de arquitectura de aplicações que visa separar a lógica da aplicação (*Model*), da interface

[26] Um **componente** de software é um módulo de programa com interface bem definida e semânticas de activação (Morisseau-Leroy, 2001).
[27] **Scriplet** é um script independente que é referenciado de uma página HTM e representa um objecto que aceita dados de fora e suporta várias operações no dado. (Shiran, 2000).
[28] **HTML** é um acrônimo para a expressão inglesa *HyperText Markup Language*, que significa Linguagem de Formatação de Hipertexto, é uma linguagem de marcação utilizada para produzir páginas na Web (Wikipedia, 2006b).

do usuário (*View*) e do fluxo da aplicação (*Controller*). Permite que a mesma lógica de negócios possa ser acedida e visualizada por várias interfaces. (Wikipédia, 2006d)

A implementação da arquitectura MVC foi feita com ajuda de alguns *frameworks* aliados a arquitectura j2ee[29]:

- O *struts*, um *framework* de desenvolvimento, na arquitectura MVC, da camada controladora, numa estrutura seguindo o padrão MCV-2 (uma variante do MVC oficializada pela Sun), de aplicações *web* (principalmente) construído em *Java* para ser utilizado em um *container web* em um servidor J2EE (Wikipedia, 2006b).

  O *struts* contém ainda, uma colecção de *tags*[30] que facilitam a apresentação da informação em página *JSP*, com vista a reduzir o número de *Scriptlets* no JSP;

- O *springframework*, um *framework* constituído por um conjunto de classes que facilitam a conexão com base de dados e execução de *querys* com poucas linhas de código. Neste trabalho, este *framework* é usado na camada de modelo (junto com os javabeans[31]), embora ele também possa ser usado na camada de controle;

- *JavaServer Pages* (JSP) é uma tecnologia para desenvolvimento de aplicações web, que na arquitectura MVC representa a *view*, criada pela sun[32], que permite a criação de páginas dinâmicas. Como uma parte da tecnologia java, permite o desenvolvimento rápido de aplicações *web* que são de plataforma independente (Sun, 2006).

- *JasperReports* é uma ferramenta *open-source*, bastante poderosa, de desenvolvimento de relatórios, totalmente construída em java, que tem a habilidade de devolver conteúdo ao *screen*, a impressora, ou ainda PDF, HTML, XLS, CSV e ficheirso XML (Jasperforge, 2006);

[29] **J2EE** é uma plataforma de programação de computadores que faz parte da plataforma Java,.voltada para aplicações multi-camadas, baseadas em componentes que são executados em um servidor de aplicações (WIKIPÉDIA, 2006a);
[30] **Tags** são estruturas de linguagem de marcação que consistem em breves instruções, tendo uma marca de início e outra de fim (Wikipedia, 2006b).
[31] Javabeans – classes em java que têm métdodos get e set para todos atributos e um construtor default.
[32] http://www.sun.com/

- *JavaServer Pages Standard Tag Library* (JSTL), uma colecção de bibliotecas de tags customizadas que implementam funcionalidades gerais, comuns a aplicações Web, incluindo iteração e condição, formatação de dados, manipulação de *Extensible Markup Language* (*XML*) e acesso a base de dados (Guapo, 2006);

  O uso de *JSTL* reduz bastante o uso de *scriplets* nas páginas JSP;
- *OSWorkflow*[33]*(opensymphony*[34] *workflow)*, um *framework* que permite a implementação de um fluxo de trabalho através de uma definição dos caminhos que um documento deve seguir, durante um processo, em um ficheiro xml, garantindo, desta forma, que seja mais fácil codificar o resto da aplicação e deixar que o framework do OSWorkflow realize a sua parte do trabalho.

  Neste trabalho, o *OSWorkflow* foi usado para gerir o processo que vai desde o pedido, emissão até à recepção do documento.

A figura que se segue mostra como é que o *struts* e *spring* são usados no SEDC, na arquitectura MVC.

Desta figura, podemos notar que a qualquer requisição do cliente, o *struts* actua como controlador (*controler*), lê a *URL* enviada, interpreta-a e comunica com a camada de modelo (*model*), caso seja necessário, a fim de obter a resposta da solicitação do cliente, e, por fim, o *framework* do *struts* envia os dados a camada de visualização (*view)* como resposta da solicitação do cliente.

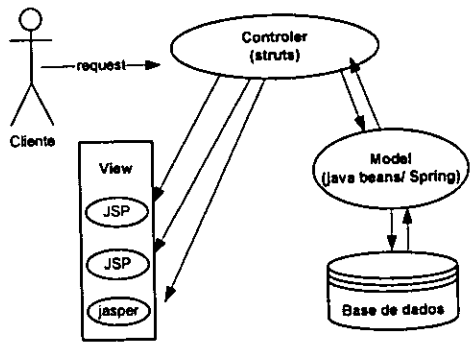

Figura 8 – O uso de struts e spring na arquitectura MVC

[33] http://www.opensymphony.com/osworkflow/

[34] **OpenSymphony** é um projecto *open-source* dedicado a prover classes a componentes j2ee. Os seus componentes apontam a simplicidade, integração, plugabilidadee e especificação compatível (Hermanns, 2006).

A figura que se segue mostra o funcionamento da aplicação, na sua totalidade, os componentes que a integram incluindo ficheiros de configuração mais importantes.

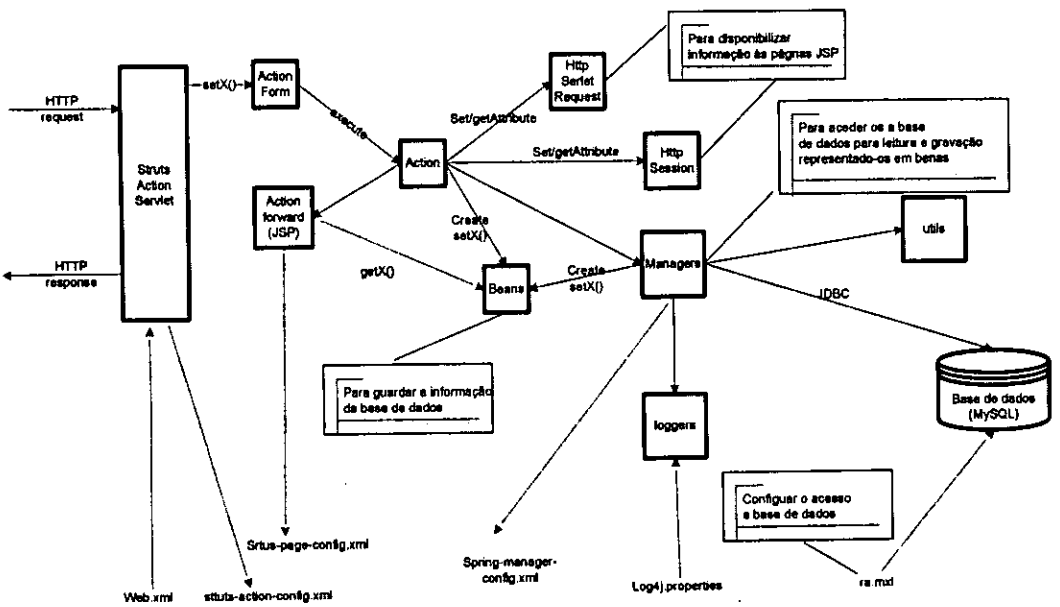

Figura 9 – Componentes arquitecturais e funcionamento do SEDC

Desta figura, podemos observar que toda requisição HTTP é interpretada pelo *framework* do *struts* (***controler***), que usa um *ActionForm* para enviar os dados do utilizador, e este, por sua vez, executa o *action* mapeado no ficheiro (*struts-action-config.xml*). Este action usa o *HttpServletRequest* e o *HttpSession* para guardar e recuperar ou pegar informação enviada na requisição. Dependendo da necessidade da solicitação, o *Action* pode criar os *javabeans* (***model***) e usa os métodos definidos nos *managers*(***model***) para executarem uma determinada operação na base de dados, caso seja necessário. Os *managers,* uma vez chamados, podem usar os *utils* (utilitários), e a ponte JDBC[35] para efectuar as transacções na base de dados, escrevem os resultado de sucesso e erro nos *logs*. Uma vez executadas as operações com os *managers*, o *action* direcciona a aplicação a uma página *JSP (view),* ou o *pdf(view)* definido no mapeamento (*struts-action-config.xml*) para visualização do resultado da requisição.

Para mais detalhes sobre a implementação do sistema, veja o CD em anexo, para ter acesso ao código fonte e ao protótipo do sistema.

[35] JDBC – Java Database Conector

5. CRITÉRIOS DE SEGURANÇA

Diariamente, no mundo inteiro, redes de computadores e servidores são invadidos. O nível de sofisticação destes ataques é muito diversificado; enquanto se acredita que a maioria das invasões tem sucesso devido a fracas senhas, há ainda um grande número de acessos ilegítimos que usam técnicas mais avançadas (Oliveira, 2006).

A segurança da informação é um assunto complexo e pode abranger várias situações: erro, displicência, ignorância do valor da informação, acesso indevido, roubo, fraude, sabotagem, causas da natureza, entre outros.

Pouco se sabe sobre a maioria das técnicas de invasão, por poderem apresentar naturezas distintas e serem de difícil detecção.

Ao longo da descrição que se segue, são abordados os mecanismos de segurança lógica e física que se sugerem para a implementação no sistema de emissão de declarações e certificados. Alguns dos mecanismos já implementados (segurança lógia), no diz respeito a transmissão segura de informação, autenticidade e integridade e outros tantos critérios de segurança que não são apresentados neste capítulo, uma vez que já existem muitos trabalhos que falam sobre os mesmos.

5.1 Critérios de segurança implementados

- **Confidencialidade.** O Acesso a informação deve ser limitado somente às entidades legítimas, ou seja, àquelas autorizadas pelo proprietário da informação.

- **Integridade**. A informação manipulada, a nível do sistema, mantém todas as características originais estabelecidas pela DRA.

- **Disponibilidade.** A Informação está sempre disponível para o uso legítimo, ou seja, por aqueles usuários autorizados pelo DRA.

- Foi incorporado um mecanismo de criptografia das senhas para garantir maior segurança no acesso ao sistema.

No modelo proposto, foram implementados alguns mecanismos de segurança de modo a garantir a transmissão segura dos dados na rede, como é o caso do protocolo HTTP em conjunto com SSL; e

- Foi usado também, o *Acegi Security*[36], uma solução flexível e ponderosa para "*enterprise software*", com uma particular ênfase para aplicações que usam Spring[37]. O uso de *Acegi Security* provê a sua aplicação com uma autenticação compreensiva, controle de acesso baseado no instante, segurança do canal (Acegi, 2006);
- *Backups* automáticos diários.

5.2 Critérios de segurança por implementar

- Sugere-se a implantação de um *Firewall*, que pode ser definido como uma barreira de protecção e filtragem que controla o tráfego de dados entre seu computador e a Internet, e tem como objectivo permitir somente a transmissão e a recepção de dados autorizados de um domínio de rede para a outra (Valente, 2004).
- Deve existir um sistema de alimentação eléctrica sem interrupção para os servidores e equipamentos activos;
- Devem-se fazer cópias para CDs dos *backups* gerados pelo sistema;
- O acesso às áreas dos servidores deve ser restrito.

[36] http://acegisecurity.org/
[37] http://springframework.org/

6. CONCLUSÕES E RECOMENDAÇÕES

6.1 Conclusões

O presente trabalho foi realizado com o objectivo de investigar a metodologia *Agile Modeling*(AM), que é uma das metodologia ágeis de desenvolvimento de software, e, aplica-la no desenvolvimento do protótipo do Sistema de Emissão de Declarações e Certificados (SEDC).

As Metodologias ágeis visam garantir a qualidade e entrega atempada, ao cliente, do software produzido e sem aumento dos custos iniciais.

As metodologias ágeis são muito recentes no mundo do desenvolvimento de software, e são bastante caracterizadas pelo seu sucesso.

As diversas pesquisas bibliográficas, feitas sobre as AM, levaram as seguintes conclusões:

- AM não é nada mais do que uma colecção de valores, princípios e práticas que focam na modelação, em primeiro plano, e a documentação, em segundo. Não sendo uma metodologia completa, ela pode ser usada junto das metodologias existentes, para aumentar a eficiência da modelação e documentação, garantido desta forma, o alcance da qualidade desejada e satisfação dos requisitos do utilizador, provendo técnicas que permitam envolver o *stakeholder*, e aceitar as suas mudanças de requisitos.
- AM não visa a eliminação da documentação, muito menos ferramentas case: Ela apenas aconselha a criação de documentos que têm valor, maximizando o investimento do cliente no processo de desenvolvimento e no uso da ferramenta que permita criar um modelo da maneira mais simples. Se um modelo for mais fácil de ser desenhado em uma ferramenta CASE do que em um papel, então, a ferramenta CASE deve ser utilizada para a criação desse modelo.

A aplicação destas metodologias no SEDC usando um desenvolvimento modular levou a concluir que:

- A aplicação da AM no desenvolvimento de software, implementa uma grande dinâmica no processo, e é uma garantia de que o software a produzir atingirá a qualidade desejada pelo cliente, uma vez que este é envolvido em todas as fases do sistema, através de *releases* constantes;
- O uso de arquitecturas e *frameworks* existentes reduz o tempo de produção de software e garante maior manuntenabilidade do software resultante, uma vez que será mais fácil encontrar soluções aos problemas que surgem durante o desenvolvimento, que de certeza vão surgir;
- *JSP* é uma ferramenta *open-source* e robusta de desenvolvimento de aplicações web, e existem na Internet, muitos *frameworks open-source* que podem ser baixados, e que, aliados a *JSP*, garantem desenvolvimento rápido do software e com qualidade;
- Páginas *JSP* não são interpretadas a cada requisição do navegador; em vez disso, são compiladas em *Servlets* no primeiro acesso, e este *Servlet* é executado nos acessos subsequentes, até que a página seja modificada, e isto faz com que as aplicações em *JSP* sejam mais rápidas que as em PHP, por exemplo;
- Antigamente, na era do *Common Gateway Interface* (CGI), cada vez que se fizesse uma requisição ao servidor, o sistema operativo criava um processo, executava a requisição e, por fim, matava o processo. E, se o cliente votasse a fazer a mesma requisição, o processo repetia-se, e isto tornava as aplicações muito lentas. Com a introdução dos *Servlets*, a primeira vez que um cliente faz uma determinada requisição, o servidor *web* cria um *Servlet* para atender a requisição (e este permanece vivo até que o servidor *web* seja desligado), isto é, se o mesmo cliente ou um outro fizer a mesma requisição, o *Servlet* já não é criado de novo. Isto faz com que as aplicações em *JSP* sejam bastante rápidas.
- O *Java Virtual Machine* (JVM), possui, dentro de si, um compilador conhecido como JIT Compiler que, durante a execução do código java, se ele se apercebe que o código pode ser optimizado, ele recompila o código escrevendo-o de

forma que este seja mais eficiente, mais um factor que torna as aplicações em *JSP* bastante rápidas, comparativamente com as outras;

- O *struts* e *springframework* são *frameworks* robustos e muito usados a nível mundial pela comunidade de desenvolvidores *web* em *JavaServer Pages*.

6.2 Recomendações

Após ter estudado e aplicado alguns dos princípios e práticas da AM recomendo que durante a modelação:

- AM é uma metodologia recente que permite simplificar o processo de desenvolvimento, e já tem dado bons resultados, tanto pelo mundo fora como em Moçambique, portanto, recomenda-se que ela seja introduzida como uma disciplina de ensino na universidade, em todas instituições de ensino de engenharia de software, pois ela não é uma metodologia rígida e pode facilmente ser usada em paralelo com as já em uso na universidade.
- Durante o desenvolvimento, é sempre importante manter o modelo o mais simples possível, mantendo sempre o fim na mente: *desenvolver software para os clientes*;

De modo a resolver os problemas que regem a DRA no processo de emissão de declarações e certificados recomendo que:

- Se crie e se integrem os demais módulos complementares deste sistema, nomeadamente, cadastro de estudantes, disciplinas, cursos, departamentos e faculdades;
- Se integre o sistema proposto tento em conta a realidade da DRA, de forma a que os dados da base de dados existente, desaguem no presente sistema, e, por conseguinte, os dados do sistema proposto sejam úteis para as restantes actividades da DRA;

- Se carregue a base de dados, com dados reais que envolvem todos os cursos, de modo a garantir que se possa emitir o certificado ou declaração de nível de qualquer estudante;
- Se implementem as medidas de segurança propostas no capítulo 5, de modo a garantir a fiabilidade dos resultados do sistema.

7. BIBLIOGRAFIA

ACEGI (2006), ***Acegi Security System for Spring***, disponível em <http://acegisecurity.org/>, consultado em 04 de Novembro de 2006

ALECRIM, E. (2004), ***Firewall – conceitos e tipos***, disponível em <http://www.infowester.com/firewall.php>, consultado em 23 de Maio de 2006

AGILE ALLIANCE (2001a), ***Manifesto for Agile Software Development,*** disponível em <http://www.agilemanifesto.org/ >, consultado em15 de Março de 2006

AGILE ALLIANCE (2001b), ***Principles behind the Agile Manifesto,*** disponível em <http://www.agilemanifesto.org/principles.html>, consultado em 15 de Março de 2006

AGILE ALLIANCE (2001c), ***History: The Agile Manifesto,*** disponível em <http://www.agilemanifesto.org/history.html>, consultado em15 de Março de 2006

AGSILVA (2006), ***Práticas de APGR***, disponível em <http://agsilva.wordpress.com/2006/05/08/praticas-de-apgr/>, consultado em 04 de Novembro de 2006

AMBLER, S. W, JEFFRIES, R.. (2002), Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process, John Wiley & Sons, Inc., New York

AMBLER, SCOT A. W. (2005), ***Agile Modeling (Effective Practices for Modeling and Documentation),*** disponível em <http://www.agilemodeling.com/>, consultado em17 de Dezembro de 2005

AMBLER, SCOT A. W. (2006a), ***Active stakeholder participation,*** disponível em <http://www.agilemodeling.com/essays/activeStakeholderParticipation.htm>, consultado em 05 de Fevereiro de 2006

AMBLER, SCOT A. W. (2006b), ***Agile Modeling (AM) Principles v2),*** disponível em <http://www.agilemodeling.com/principles.htm>, consultado em 05 de Fevereiro de 2006

AMBLER, SCOT A. W. (2006c), ***Agile Modeling (AM) Pratices v2),*** disponível em <http://www.agilemodeling.com/practices.htm>, consultado em 05 de Fevereiro de 2006

AMBLER, SCOT A. W. (2006d), ***Agile Modeling (AM) Values v2),*** disponível em <http://www.agilemodeling.com/values.htm>, consultado em 05 de Fevereiro de 2006

AMBLER, SCOT A. W. (2006e), ***Agile Modeling (AM) Pratices v2),*** disponível em <http://www.agilemodeling.com/essays/practicesFitTogether.htm>, consultado em 17 de Março de 2006

AMBLER, SCOT A. W. (2006f), ***Agile Documentation: strategies for agille software documentation,*** disponível em <http://www.agilemodeling.com/essays/agileDocuemtation.htm>, consultado em 23 de Junho de 2006

BOERING, E. (2003), ***Portal Java:: RUP-Rational Unified Process***, disponível em http://www.portaljava.com.br/home/modules.php?name=Content&pa=showpage&pid=15&page=2, consultado em 07 de Setembro de 2006

BRAUN, D. e tal, (2001a), ***UML Tuitorial – What is UML?***, disponível em <http://pigseye.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/what_is_uml.htm>, consultado em 07 de Julho de 2006

BRAUN, D. e tal, (2001b), ***UML Tuitorial – Use Case Diagrams***, disponível em <http://pigseye.kennesaw.edu/~dbraun/csis4650/A&D/UML_tutorial/use_case.htm>, consultado em 07 de Julho de 2006

CATO J. (2001), ***User-Centered Web Design,*** Londres: Addison-Wesley

COCKBURN, Alistair. (2002), ***Agile Software Development***, Canada , Addison Wesley

DO AMARAL, Wanda (1991), ***Guia para apresentação de teses e dissertações de trabalhos de graduação***, Maputo, 2ª edição, Livraria universitária

EELES, Peter (2006), ***what is a software architecture?***

Guapo (2006), ***Mundo OO – JSP Standard Tag Library (JSTL)***, disponível em <http://www.mundooo.com.br/php/modules.php?name=News&file=article&sid=560>, consultado em 23 de Novembro de 2006

HENSGEN, P., et Al (2001), ***Elementos de UML***, disponível em <http://docs.kde.org/development/pt/kdesdk/umbrello/uml-elements.html>, consultado em 11 de Setembro de 2006

HERMANNS, R. (2006), ***OpenSymphony - Welcome To OpenSymphony***, disponível em <http://www.opensymphony.com/>, consultado em 06 de Dezembro de 2006

HIGHSMITH, Jim (2002), ***Agile Software Development Ecosystems***, Canada, Addison Wesley

JACOBSON, I., et. Al (1999), Unified ***Software Development Process,*** Addison Wesley;

JASPERFORGE (2006), ***JasperReports Project Home***, disponível em <http://jasperforge.org/sf/projects/jasperreports>, consultada em 03 de Janeiro de 2007

MACORATTI, J. C. (2006), ***UML - Diagrama de Classes e objectos,*** disponível em <http://www.macoratti.net/net_uml1.htm>, consultado em 07 de Julho de 2006

LOZANO (2003), Introdução ao J2EE, disponível em <http://www.lozano.eti.br/palestras/intro-j2ee.pdf>, consultado em 05 de Dezembro de 2006

LUIZ, R. R. V (2004), ***JavaFree:: Obtendo Qualidade de Software com o RUP,*** disponível em <http://www.javafree.org/content/view.jf?idContent=7>, consultado em 07 de Setembro de 2006

MACORATTI, J. C. (2004), Modelando sistemas em UML - Casos de uso, disponível em <http://www.imasters.com.br/artigo/2753/uml/modelando_sistemas_em_uml_-_casos_de_uso>, consultado em 07 de Julho de 2006

MARTINS, D. F. (2006), ***Apresentando Model-View-Presenter, o MVC focado na visualização***, disponível em <http://www.javafree.org/content/view.jf?idContent=91>, consultado em 15 de Setembro de 2006

MORISSEAU-LEROY, N. et al (2001), ***Oracle 8i – Programação de componentes Java com EJB, CORBA e JSP***, Brazil, Editora Campus limitada

MUCHANGA, Salane (2005) ,***Informática para gestão de lixo***, Jornal Savana, 23/09/2005, p.12

NGUERRA (2006), ***SSL – O que é, disponível em,*** disponível em <http://www.nguerra.com.br/ecommerce/mensagens/pg/ssl.htm>, consultado em 20 de Julho de 2006

OLIVEIRA, J. W. (2006), ***Segurança da Informação – Técnicas e Soluções,*** disponível em <http://www.centroatl.pt/titulos/si/seguranca_da_informacao.php3>, consultado em 10 de Julho de 2006

RATIONAL (2001), ***Rational Unified Process***, disponível em <http://www.wthreex.com/rup/>, consutaldo em 07 de Setembro de 2006

SANTOS, A, et. al, (2003), ***Agile Modeling – Overview,*** disponível em <http://www.pr.gov.br/batebyte/edicoes/2003/bb131/agile.shtml>, consultado em 01 de Dezembro de 2005

SMARTDRAW, 2006, ***Types of UML Diagramas***, disponível em <http://www.smartdraw.com/tutorials/software-uml/uml.htm>, consultado em 07 de Julho de 2006

SHIRAN, Y. (2000), ***Scriptlets***, disponível em <http://www.webreference.com/js/tips/000429.html>, consultado em 05 de Novembro de 2006

SOMMERVILLE, I. (1995), ***Software Engineering***, 5ª Edição. Londres: Addison-Wesley

SUN (2006), ***JavaServer Pages Overview***, disponível em <http://java.sun.com/products/jsp/overview.html>, consultado em 06 de Dezembro de 2006

TELEMACO, U. (2004), ***O site da comunidade J2EE no Brasil***, disponível em <http://www.jspbrasil.com.br/jsp/tutoriais/tutorial.jsp?idTutorial=001_001>, consultado em 04 de Julho de 2006

TEMBE, Joaquim (2006), ***responsável do departamento de estatística do Centro de Saúde e Polana Caniço***, sobre "desenvolvimento de novo sistema informático para gestão de recursos humanos e salários", entrevista em 16 de Agosto de 2006

UEM (2006), ***Edital de Admissão à UEM/ Ano Lectivo de 2007***, disponível em <http://www.admissao.uem.mz/>, consultado em 04 de Novembro de 2006

UEM, B. (2001), ***Regulamento Pedagógico – Universidade Eduardo Mondlane***, Maputo

VALENTE, Carlos (2004), ***Firewall***, disponível em http://novainter.net/cgi-bin/wiki.pl?Firewall, consultado em 23 de Maio de 2006

Velloso, F. C, (2002), ***Informática: Conceitos Básicos***, 6ª Edição, Lisboa: Editora . Campus

WIKIPÉDIA (2006a), ***Wikipédia, a enciclopédia livre***, disponível em <http://pt.wikipedia.org/wiki/>, consultado em 22 de Maio de 2006

WIKIPEDIA (2006b), ***Wikipedia, the free encyclopedia***, disponível em <http://en.wikipedia.org/wiki/>, consultado em 05 de Dezembro de 2006

WIKIPÉDIA (2006c), ***Wikipédia, a enciclopédia livre – Segurança da informação***, disponível em <http://pt.wikipedia.org/wiki/Seguran%C3%A7a_da_informa%C3%A7%C3%A3o>, consultado em 12 de Julho de 2006

WIKIPÉDIA (2006d), ***Wikipédia, a enciclopédia livre – MVC,*** disponível em <http://pt.wikipedia.org/wiki/MVC>, consultado em 15 de Setembro de 2006

WHITEHEAD, Richard (2001), ***Leading a software development team***, Londres, Addison Wesley

ANEXO A — O Manifesto para o Desenvolvimento Ágil de Software

De 11 à 13 de Fevereiro de 2001, um grupo de dezassete defensores de processos leves de desenvolvimento (*ligthweight development process*), esteve reunido em Utá (um estado dos Estados Unidos da América) para trocar impressões sobre o que eles poderiam ter em comum. Nesse encontro, acordou-se que a palavra *ligthweight* era mais uma reacção a alguma coisa e não uma convicção em algo. E concordando na importância de estar a altura de responder à mudança de requisitos durante o projecto, eles escolheram a palavra *agile*, e decidiram, então, dar ao grupo, o nome, *Agile Alliance* (Cockburn, 2002).

Este grupo tem trabalhado no sentido de descobrir melhores caminhos de desenvolvimento fazendo-o e ajudando os outros a fazerem. E através desse trabalho, eles chegaram a um conjunto de ***quatro valores*** (os quais intitularam de "*O Manifesto*") e uma dúzia de princípios para suportar tais valores (Cockburn, 2002).

O Manifesto

Indivíduos e interacção *em vez de processos e ferramentas*

Infelizmente, em muitas empresas de desenvolvimento de *software,* os analistas e gestores de projectos acabam limitando-se em documentações e ferramentas de integração de seus modelos e, com isso, acabam deixando de lado algo que é muito importante em uma equipa, que é a cooperação de todos e o *feedback* dos colaboradores envolvidos com o projecto.

Software funcional *em vez de extensa documentação*

Em várias ocasiões é mais útil um protótipo simples que mostre o funcionamento de uma arquitectura do que um grande e complexo diagrama de classes; também é mais

simples para o *stakeholder*[38] analisar o funcionamento do sistema através de um protótipo, mesmo que não seja real, do que através de um conjunto de diagramas de casos de usos e projectos de interface. Não se trata de abandonar o processo de documentação, mas apenas de utilizar a ferramenta certa para transmitir a informação desejada no momento.

***Colaboração com o stakeholder** em vez de renegociação de contrato*

Como todo o desenvolvedor deve saber, quem deve definir o que o sistema deve ou não fazer é o *stakeholder*. Pode dizer-se que o *stakeholder* não tem conhecimento suficiente para especificar o sistema, que o *stakeholder* vive mudando de ideia sobre o que o sistema deve fazer, que o *stakeholder* vive adicionando requisitos, etc., porém essa é a realidade do processo de desenvolvimento; em vez de tratar o *stakeholder* como "aquele pessoal que só atrapalha", deve realizar-se um trabalho de descoberta das necessidades do *stakeholder* e educação do mesmo, para o processo durante a duração do projecto. É um caminho bastante árduo, porém tem se mostrado muito gratificante para aqueles que o trilham.

***Aceitação das mudanças** em vez de obediência cega a um plano*

As mudanças fazem parte da vida, especialmente na área de desenvolvimento de *software*; é simplesmente contraproducente reclamar desse facto. Em vez disso, acostume-se às mudanças; é importante ter um plano flexível o bastante para aceitar as mudanças do ambiente onde esse *software* deverá ser utilizado.

Princípios para um desenvolvimento ágil de software

Para ajudar aos desenvolvedores a perceber melhor o que é um desenvolvimento ágil de um *software*, os membros da *Agile Aliance* refinaram as filosofias capturadas no seu

[38] ***Stakeholder** - É qualquer um que seja utilizador directo ou indirecto, gestor de utilizador ou sénior, pessoal do nível operativo, pessoal do suporte, desenvolvedores dos sistemas que integram ou interagem com o software em desenvolvimento, ou profissionais de manutenção afectados pelo desenvolvimento de um projecto de software (Ambler, 2006a)*

manifesto numa colecção de doze princípios, que todas as metodologias ágeis de desenvolvimento de *software* devem concordar (Cockburn, 2002).

1. A nossa prioridade superior é satisfazer o cliente através de entrega atempada e contínua do valioso *software*;
2. Sejam bem-vindas as mudanças dos requisitos, mesmo tarde no desenvolvimento. O processo ágil aproveita mudanças para vantagens competitivas do cliente;
3. Entregar partes do *software* frequentemente, semanalmente ou mensalmente, com preferência a menor escala de tempo possível;
4. Os desenvolvedores e os stakeholders devem trabalhar juntos durante todo o projecto;
5. Construa projectos em torno das motivações individuais. Dê ambiente e suporte que os desenvolvedores precisam para que tenham o trabalho feito;
6. O método mais eficiente e efectivo de transmissão de informação para e com a equipa de desenvolvimento é conversa frente a frente;
7. Um *software* funcional, é a primeira medida de progresso;
8. Processo ágil promove o desenvolvimento sustentável. Os clientes, desenvolvedores e utilizadores deveriam ser capazes de manter um ritmo constante indefinidamente;
9. Atenção contínua para técnica excelente e bom desenho realça agilidade;
10. Simplicidade – a arte de maximizar a quantidade de trabalho não feito – é essencial;
11. As melhores arquitecturas, requisitos e desenhos surgem de equipas auto-organizadas;
12. Nos intervalos regulares, a equipa reflecte em como se tornar mais efectiva e depois sintonizam e ajustam o comportamento adequadamente;

ANEXO B — Fases do RUP

A Rational (2003), no trabalho ***Rational Unified Process*** divide um projecto em quatro fases diferentes, distinguidas na figura 4.

Iniciação: *ênfase no escopo do sistema*

Esta fase é a fase inicial. Ela enfatiza no escopo do sistema e tem como objectivos: estabelecer o escopo do software, discriminar os casos de uso críticos, esboçar a arquitectura, estimar custos e riscos e preparar o ambiente de suporte.

Elaboração: *ênfase na arquitectura*

A fase da elaboração enfatiza a arquitectura e tem como objectivos: assegurar que na arquitectura, os requisitos sejam estáveis o suficiente, produzir um protótipo evolutivo dos componentes de qualidade de produção, demonstrar que a arquitectura suportará os requisitos do sistema a um custo e em tempo justo e estabelecer um ambiente de suporte.

Construção: *ênfase no desenvolvimento*

Esta fase enfatiza o desenvolvimento e tem como objectos: atingir a qualidade adequada e versões úteis com rapidez e eficiência, minimizando os custos de desenvolvimento, optimizando recursos e decidir se o software, os locais e os utilizadores estão prontos para que o aplicativo seja implantado.

Transição: *ênfase na implantação*

A transição é a fase de ênfase na implantação, os seus objectivos são: a execução de testes para validar o sistema, treinamento dos utilizadores, correcção de erros, melhorias no desempenho e na usabilidade e obtenção de consentimento dos envolvidos de que o produto implantado é consistente com os critérios de avaliação da visão.

ANEXO C — Detalhamento de casos de uso

Efectuar pedido de documento (UC 001)

Breve descrição

Este caso de uso permite que um estudante ou um funcionário da reitoria registre o pedido de um documento.

Pre-requisitos

O utilizador efectuou o login.

Actores

Estudante e Funcionário da reitoria.

Fluxos

1. O caso de uso começa quando o utilizador selecciona o menu *Pedido de documentos* e em seguida, clica no *link Registar pedido de declaração/certificado*;
2. O sistema apresenta um formulário com os campos: número de estudante, motivo e documento a solicitar e dois botões *Gravar* e *Cancelar*:
    a. Se o utilizador logado for estudante, o campo número de estudante aparece preenchido sem opção de alterar;
3. O utilizador preenche os campos e clica no botão *Gravar*:
    a. Caso ele tenha entrado no formulário por engano, ou tenha desistido, ele pode clicar no botão *Cancelar*;
4. O sistema faz a validação da solicitação verificando se os campos estão todos preenchidos e se existe um estudante com o número introduzido e, finalmente se o estudante tem alguma dívida não saldada com a reitoria:

a. Caso os campos não estejam todos preenchidos ou o número de estudante seja inválido, uma mensagem de erro é apresentada e o sistema volta ao passo 3;

b. Caso o estudante tenha uma dívida, o caso de uso termina e uma mensagem de insucesso é apresentada, indicando o motivo, e como proceder para solucionar o problema, caso contrário vai para o passo seguinte;

c. Caso contrário, o sistema passa para o passo seguinte;

5. O sistema apresenta ao utilizador os detalhes do estuante, e do documento seleccionado incluindo o preço e três botões, confirmar pedido, corrigir detalhes e cancelar:

   a. Se o utilizador clicar no botão *confirmar pedido* o sistema grava o pedido do documento e o caso de uso termina e o sistema volta ao passo 1;

   b. Se o utilizador clicar no botão *corrigir detalhes*, o sistema volta ao passo 4, com os campos preenchidos;

   c. Se o utilizador clicar no botão *cancelar* o caso de uso termina e o sistema volta ao passo 1;

Casos de uso superiores

Não tem nenhum caso de uso superior.

Casos de uso subordinados

Devolver pedido, cancelar pedido e emitir documento.

Devolver pedido (UC 002)

Breve descrição

Este caso de uso permite que um funcionário da reitoria permita devolver um pedido a um estudante caso, colocando o motivo da devolução. Será normalmente usado quando o estudante não especificar claramente o motivo do pedido.

Pre-requisitos

O estudante em causa efectuou um pedido de documento.

Actores

Funcionário da reitoria.

Fluxos

1. Após efectuar o Login, o sistema apresenta uma lista de todos os pedidos, e para cada pedido um *link* com a lista das acções possíveis para o pedido em causa;
2. O utilizador selecciona, da lista, o pedido que deseja devolver;
3. O sistema apresenta uma tela com os detalhes do estudante e do documento seleccionado, com várias opções dentre, as quais *Devolver pedido* e *Voltar:*
   a. Se o utilizador clicar no botão *voltar,* o sistema volta ao passo 1;
   b. Se o utilizador clicar no botão *Devolver pedido,* o sistema passa para o passo seguinte;
4. O sistema apresenta uma tela com um campo para o utilizador indicar o motivo da devolução do pedido e dois botões *Devolver* e *voltar*;
5. O utilizador preenche o motivo da devolução e clica no botão *Devolver*:
   a. Caso o utilizador tenha entrado nesta opção por engano ou desista a operação, ele pode clicar no botão *Voltar,* e o sistema volta ao passo 3;
6. O sistema apresenta uma tela de confirmação da devolução, com três botões, *confirmar devolução, corrigir observação* e *cancelar a devolução:*
   a. Se o utilizador clicar no botão *Corrigir observação,* o sistema volta ao passo 4;
   b. Se o utilizador clicar no botão *Confirmar devolução,* o caso de uso termina e o documento será devolvido para rectificação, e o sistema volta ao passo 1;
   c. Se o utilizador clicar no botcão Cancelar, o caso de uso termina e sistema volta ao paso 1;

Casos de uso superiores

Efectuar pedido de documento.

Casos de uso subordinados

Cancelar pedido e alterar detalhes do pedido.

Alterar detalhes dum pedido (UC 003)

Breve descrição

Este caso de uso permite a um estudante ou um funcionário da reitoria altear os detalhes do pedido de documento caso este tenha sido devolvido.

Pre-requisitos

O estudante em causa efectuou um pedido de documento que tenha sido devolvido.

Actores

Estudante, Funcionário da reitoria.

Fluxos

1. Após efectuar o Login, o sistema apresenta uma lista de todos os pedidos, e para cada pedido um *link* com a lista das acções possíveis para o pedido em causa:
   a. Caso o utilizador seja estudante, o sistema apresenta apenas os seus pedidos;
2. O utilizador selecciona, da lista, o pedido que deseja alterar;
3. O sistema apresenta uma tela com os detalhes do estudante e do documento seleccionado, com várias opções, dentre as quais *Alterar detalhes* e *Voltar;*
   a. Se o utilizador clicar no botão *voltar,* o sistema volta ao passo 1.

b. Se o utilizador clicar no botão *Alterar detalhes,* o sistema passa para o passo seguinte;

4. O sistema apresenta uma tela com um campo para o utilizador indicar o novo motivo do pedido e dois botões *Alterar* e *voltar*;
5. O utilizador preenche o novo motivo e clica no botão *Alterar;*
   a. Caso o utilizador tenha desistido ou entrado na opção por engano, ele pode clicar no botão *Voltar*, e o sistema volta ao passo 1;
6. O sistema apresenta uma tela de confirmação da alteração, com três botões, *Confirmar*, *Corrigir* e *Cancelar a alteração:*
   a. Se o utilizador clicar no botão *Corrigir,* o sistema volta ao passo 4;
   b. Se o utilizador clicar no botão *Confirmar,* o caso de uso termina e o motivo de pedido de documento será alterado, e o sistema volta ao passo 1;
   c. Se o utilizador clicar no botão *Cancelar a alteração*, o caso de uso termina, nenhuma alteração é feita ao documento, e o sistema volta ao paso 1;

Casos de uso superiores

Devolver pedido

Casos de uso subordinados

Cancelar pedido, emitir pedido e devolver pedido.

Cancelar pedido (UC 004)

Breve descrição

Este caso de uso permite que um estudante ou um funcionário da reitoria cancele um determinado pedido, antes que o documento seja emitido, o cancelamento pode advir do

facto de, após efectuar o pedido, um documento, o estudante desistir do mesmo, por algum motivo qualquer.

Pre-requisitos

O estudante em causa efectuou um pedido de documento, ou o pedido do estudante em causa foi devolvido.

Actores

Estudante, Funcionário da reitoria.

Fluxos

1. Após efectuar o Login, o sistema apresenta uma lista de todos os pedidos, e para cada pedido um *link* com a lista das acções possíveis para o pedido em causa:
   a. Caso o utilizador seja estudante, o sistema apresenta apenas os seus pedidos
2. O utilizador selecciona, da lista, o pedido que deseja cancelar;
3. O sistema apresenta uma tela com os detalhes do estudante e do documento seleccionado, com várias opções, dentre as quais *Cancelar pedido* e *Voltar:*
   a. Se o utilizador clicar no botão *voltar,* o sistema volta ao passo 1;
   b. Se o utilizador clicar no botão *Cancelar pedido,* o sistema passa para o passo seguinte;
4. O sistema apresenta uma tela de confirmação do cancelamento com dois botões *Confirmar cancelamento de pedido* e *Voltar*;
   a. Se o utilizador clicar no botão *Confirmar cancelamento de pedido,* o pedido é cancelado e nunca mais aparece na lista das documentos por processar (fim do processo de emissão de documentos).
   b. Se o utilizador clicar no botão *voltar,* o sistema volta ao passo 3.

Casos de uso superiores

Efectuar pedido de documento, devolver pedido e alterar detalhes dum pedido.

Casos de uso subordinados

Não tem casos de uso subordinados.

Imprimir documento (UC 005)

Breve descrição

Este caso de uso permite que funcionário da reitoria imprima o documento de um determinado pedido.

Pre-requisitos

O estudante em causa efectuou um pedido de documento, ou o pedido do estudante em causa alteado.

Actores

Funcionário da reitoria.

Fluxos

1. Após efectuar o Login, o sistema apresenta uma lista de todos os pedidos, e para cada pedido um *link* com a lista das acções possíveis para o pedido em causa;
2. O utilizador selecciona da lista o pedido do documento a imprimir;
3. O sistema apresenta uma tela com os detalhes do estudante e do documento seleccionado, com várias opções dentre as quais *Imprimir documento* e *Voltar:*
   a. Se o utilizador clicar no botão *voltar,* o sistema volta ao passo 1;

b. Se o utilizador clicar no botão *Imprimir documento,* o sistema passa para o passo seguinte;

4. O sistema apresenta uma tela de confirmação da emissão de documento com dois botões *Confirmar Impressão de documento* e *voltar*:
   a. Se o utilizador clicar no botão *voltar,* o sistema volta ao passo 3;
   b. Se o utilizador clicar no botão *Confirmar Impressão de documento,* o documento é marcado como emitido e o sistema passa para o passo seguinte;
5. O sistema apresenta um *link* para visualizar o documento emitido, em formato pdf.
6. O ultizador clica no *link* e o documento é apresentado no formato pdf, disponível para impressão.

Casos de uso superiores

Efectuar pedido de documento, alterar detalhes dum pedido.

Casos de uso subordinados

Arquivar documento.

Tomar conhecimento da emissão do documento (UC 007)

Breve descrição

Este caso de uso serve para que um estudante tome conhecimento da emissão do documento por ele solicitado.

Pre-requisitos

O estudante em causa efectuou um pedido de documento, e este documento já foi arquivado.

Actores

Estudante.

Fluxos

1. Após efectuar o Login, o sistema apresenta uma lista de todos os pedidos do estudante logado, e para cada pedido um *link* com a lista das acções possíveis para o pedido em causa;
2. O utilizador selecciona da lista o pedido do documento a tomar conhecimento;
3. O sistema apresenta uma tela com os detalhes do estudante e do documento seleccionado, acompanhado de uma mensagem dos problemas que o estudante poderá ter se não levantar o documento nos dias seguintes;
4. O utilizador clica no botão ok e o sistema volta ao passo 1, e este documento só desaparece da lista dos pendentes quando o documento for levantado.

Casos de uso superiores

Assinar documento.

Casos de uso subordinados

Entrega do documento.

Entrega do documento (UC 008)

Breve descrição

Este caso de uso permite que um funcionário da reitoria registre a entrega do documento solicitado por um estudante.

Pre-requisitos

O estudante em causa efectuou um pedido de documento, e este documento já foi emitido.

Actores

Funcionário da reitoria.

Fluxos

1. Após o efectuar o Login, o sistema apresenta uma lista de todos os pedidos, e para cada pedido um *link* com a lista das acções possíveis para o pedido em causa;
2. O utilizador selecciona da lista o pedido do documento a entregar;
3. O sistema apresenta uma tela com os detalhes do estudante e do documento seleccionado, com duas opções dentre as *Confirmar a entrega* e voltar:
    a. Se o utilizador clicar no botão *voltar,* o sistema volta ao passo 1;
    b. Se o utilizador clicar no botão *Confirmar a entrega,* o caso de uso termina, o documento é entregue ao estudante; o sistema volta ao passo 1 e o documento nunca mais aparece na lista de pendentes (fim do processo de emissão de documentos).

Casos de uso superiores

Arquivar documento.

Casos de uso subordinados

Não tem.

Criar utilizador (UC 009)

Breve descrição

Este caso de uso permite ao administrador do sistema criar um novo utilizador do sistema.

Pre-requisitos

O utilizador efectuou o login.

Actores

Administrador do sistema

Fluxos

1. O caso de uso começa quando o utilizador selecciona o menu *Criar utilizador*;
2. O sistema apresenta uma página, para indicar o tipo de utilizador numa lista drop-down;
3. O utilizador selecciona o tipo de utilizador desejado e clica no botão Prosseguir;
4. Se não existir nenhum indivíduo no sistema, o sistema apresenta uma mensagem de erro, caso contrário, o sistema apresenta:
    a. Uma página para indicar o nome do funcionário, caso o utilizador a criar seja funcionário (*esta página só apresenta os funcionários que ainda não são utilizadores do sistema*); ou
    b. Uma página para introduzir o número de estudante, caso o utilizador a criar seja estudante;
5. O utilizador preenche o número de estudante ou selecciona o funcionário, consoante o caso, e clica no botão *Prosseguir:*
    a. Caso o utilizador seja estudante, o sistema deve validar se o número de estudante é válido e se ele ainda não é utilizador do sistema;

6. O sistema apresenta uma página de confirmação, com os detalhes do indivíduo em causa:

   a. Se o utlizador clicar no botão *confirmar*, o sistema gera automaticamente um nome de usuário (prmeironome.apelido) e uma senha automática que o utilizador *será obrigado a trocar* quando fizer o primeiro login e passa para o passo seguinte. Caso já exista alguém com mesmo primeiro nome e apelido, que já tenha sido registrado como utilizador, o sistema acrescenta automaticamente um número sequencial no fim do nome do usuário;

   b. Se o utilizador clicar em *corrigir,* o sistema volta ao passo 5 com os campos preenchidos;

   c. Se o utilizador clicar no botão *cancelar*, o caso de uso termina, e o indivíduo não será registado como utilizador do sistema;

7. O Sistema mostra o nome do utilizador e a senha.

Casos de uso superiores

Efectuar login.

Casos de uso subordinados

Não tem.

Alterar Senha (UC 010)

Breve descrição

Este caso de uso que um usuário possa alterar a sua senha.

Pre-requisitos

O utilizador efectuou o login.

Actores

Todos.

Fluxos

1. O caso de uso começa quando o utilizador selecciona o menu *Alterar senha*;
2. O sistema apresenta uma página, para indicar a senha antiga(actual), a nova senha e a confirmação da nova senha.
3. O utilizador preenche todos os campos e clica no botão *alterar senha*.
4. O sistema valida as senhas introduzidas:
    a. Caso a senha antiga/actual não confira, o sistema apresenta uma mensagem de erro e volta ao passo 2 com os dados preenchidos pelo utilizador;
    b. Caso a nova senha não confira com a nova, o sistema apresenta uma mensagem de erro e volta ao passo 2 com os dados preenchidos pelo utilizador;
    c. Caso os dados passem a validação o sistema passa para o passo seguinte;
5. O sistema apresenta uma página de confirmação com três botões *Confirmar Alteração da senha*, *Corrigir a senha* e *cancelar*:
    a. Se o utilizador clicar em *Confirmar Alteração da senha*, a senha do usuário é altearada. O caso de uso termina e o sistema volta ao passo 1;
    b. Se o utilizador clicar em *Corrigir a senha*, o sistema volta ao passo 2;
    c. Se o utilizador clicar em *Cancelar*, o processo de alteração da senha termina e a senha actual do utilizador mantém-se intacta.

Casos de uso superiores

Não tem.

Casos de uso subordinados

Não tem.

Listar pedidos de documentos (UC 011)

Breve descrição

Este caso de uso permite que um funcionário da reitoria possa listar os pedidos de um documento, e imprimir a segunda via dos mesmos.

Pre-requisitos

O utilizador efectuou o login.

Actores

Funcionário da reitoria

Fluxos

1. O caso de uso começa quando o utilizador selecciona o menu *Listar pedidos de documentos;*
2. O sistema apresenta uma página, onde o utilizador deve indicar os critérios de pesquisa (tipo de documento, número de estudante e motivo);
3. O utilizador preenche os campos que deseja usar para pesquisa e clica o botão *Procurar*; caso ele tenha desistido ele pode clicar no botão *Cancelar*;
4. O sistema lista todos documentos com as características indicadas; caso o utilizador não tenha preenchido nada, o sistema mostra todos pedidos;
5. O utilizador clica, da lista, o pedido que deseja verificar/imprimir e o sistema passa para o passo seguinte:

a. Caso o pedido que o utilizador esteja a procurar não apareça na lista, o utilizador pode clicar em no botão *mudar critérios de pesquisa,* e o sistema vota ao passo 2, com os valores previamente preenchidos;

6. O sistema apresenta todos os detalhes do pedido de documento e um botão *seleccionar outro*; o sistema apresenta ainda um botão *imprimir*, se o documento já tenha sido emitido (ver UC 005 – emitir documento):

   a. Se o utilizador clicar no botão *Imprimir*, o sistema visualiza o documento em pdf e o caso de uso termina;

   b. Se o utilizador clicar no botão *Seleccionar outro*, o sistema volta ao passo 4.

Casos de uso superiores

Não tem.

Casos de uso subordinados

Não tem.

Criar turma (UC 012)

Breve descrição

Este caso de uso permite que um funcionário da faculdade registre as turmas no sistema.

Pre-requisitos

O utilizador efectuou o login.

Actores

Funcionário da faculdade.

Fluxos

1. O caso de uso começa quando o utilizador selecciona o menu *Criar turma;*
2. O sistema apresenta um formulário com os campos: nome da turma, curso (apenas os cursos leccionados no departamento do funcionário logado), semestre, e turno e um botão *prosseguir*;
3. O utilizador preenche os campos e clica no botão corrigir, e o sistema valida se os campos estão todos preechidos:
   a. Caso os campos não estejam todos preenchidos, o sistema apresenta uma mensagem de erro, indicando os campos com problema e volta ao passo 2, com os dados introduzidos pelo utilizador;
   b. Se os campos estiverem todo preenchidos, o sistema passa para o passo seguinte;
4. O sistema apresenta uma página com os detalhes da turma a criar e a lista de estudantes que pertencem ao turno e curso escolhido, e que não estejam numa turma do mesmo semestre e mesmo ano e dois botões, *Seleccionar* e *Corrigir;*
5. O utilizador selecciona, da lista, os estudantes que pertencem a esta turma e clica no botão *Seleccionar*:
   a. Caso o utilizador se aperceba de ter preenchido mal um campo ele pode voltar ao passo anterior, bastando para tal clicar no botão *Corrigir*
6. O sistema verifica se foi seleccionado pelo menos um estudante (Caso não tenha seleccionado nenhum estudante, o sistema volta ao passo 4), e apresenta uma tela de confirmação de criação da turma. Com três botões: *Confirmar*, *Corrigir* e *Cancelar:*
   a. Se o utilizador clicar em *Confirmar*, a turma é criada, e o sistema volta ao passo 2;
   b. Se o utilizador clicar em *Corrigir*, o sistema volta ao passo 4, com os estudantes seleccionados;
   c. Se o utilizador clicar em *Cancelar*, o sistema volta ao passo 1 sem criar a turma;

Casos de uso superiores

Não tem.

Casos de uso subordinados

Não tem.

Registar notas (UC 013)

Breve descrição

Este caso de uso permite que um funcionário da faculdade registre as notas de um estudante, bem como de uma turma.

Pre-requisitos

O utilizador efectuou o login.

Actores

Funcionário da faculdade.

Fluxos

1. O caso de uso começa quando o utilizador selecciona, do menu, a opção *Registrar notas;*
2. O sistema apresenta um formulário com um campo para introduzir o número de estudante, (caso a introdução das notas seja por estudante), um *link* para a introdução das notas por turma, e um botão *prosseguir*:
   a. Se o utilizador clicar no *link* o sistema passa para o passo 7;
3. O utilizador preenche o número de estudante e clica no botão *Prosseguir*, e o sistema valida o número de estuante e:

a. Se o número de estudante não for reconhecido, o sistema apresenta uma mensagem de erro e volta ao passo 2;

b. Se os campos estiverem correctamente preenchidos, o sistema passa para o passo seguinte;

4. O sistema apresenta uma página com os detalhes do estudante, para questões de confirmação e uma lista das disciplinas que este estudante pode fazer, de acordo com as regras de precedências; e para cada disciplina, uma lista drop-down de notas de 0 à 20.

5. O utilizador, preenche as notas das cadeiras em questão e clica no botão *Registrar* (o sistema passa para o passo seguinte), caso ele tenha introduzido um número de estudante por engano, ele pode clicar em *Voltar*, e alterar o número de estudante;

   a. Se o utilizador não seleccionar nenhum estudante, uma mensagem de erro é apresentada, e o sistema vota ao passo 4;

6. O sistema apresenta uma página de confirmação, para que o utilizador possa conferir as notas introduzidas e corrigir caso tenha havido algum engano. Esta página de confirmação tem três botões: *Confirmar*, *Corrigir* e *Cancelar;*

   a. Se o utilizador clicar em *Confirmar*, o sistema grava as notas na base de dados e o caso de uso termina, e o sistema volta ao passo 2;

   b. Se o utilizador clicar em *Corrigir*, o sistema volta ao passo 5 com as notas já introduzidas;

   c. Se o utilizador clicar em *Cancel*ar, o sistema volta ao passo 1;

7. O sistema apresenta uma página com uma lista, em forma de *drop-down*, de turmas do departamento do funcionário logado, e uma lista (vazia) de disciplinas e um botão *Prosseguir*;

8. O utilizador selecciona a turma, e o sistema traz as disciplinas do curso e semestre da turma; o utilizador selecciona a turma e clica no botão *Prosseguir;*

9. O sistema apresenta uma página com os detalhes da turma, da disciplina, uma lista dos estudantes que pertencem a turma seleccionada e que tenham feito todas cadeiras precedentes a cadeira em causa e dois botões *Registar* e *Corrigir;*
10. O utilizador preenche as notas dos estudantes que vem na pauta e clica no botão *Registrar* e o sistema passa para o passo seguinte, caso tenha preenchido ao menos um estudante, caso não tenha preenchido, o sistema dá uma mensagem de erro e volta ao passo 9;
    a. Caso não tenha seleccionado a turma ou disciplina desejada, ele pode clicar no botão *Corrigir* e o sistema volta ao passo 9, com os dados já preenchidos;
11. O sistema apresenta uma página de confirmação com três botões: *Confirmar*, *Corrigir* e *Cancelar*:
    a. Se o utilizador clicar em *Confirmar*, o sistema grava os resultados dos estudantes seleccionados, o caso de uso termina e o sistema volta ao passo 7;
    b. Se o utilizador clicar em *Corrigir*, o sistema volta ao passo 10, apresentando os dados dos estudantes previamente preenchidos;
    c. Se o utilizador clicar em *Cancelar*, o sistema cancela a inserção das notas e o sistema volta ao passo 1;

Casos de uso superiores

Criar turma

Casos de uso subordinados

Não tem.

ANEXO D — Diagrama de actividades do pedido e emissão de documentos

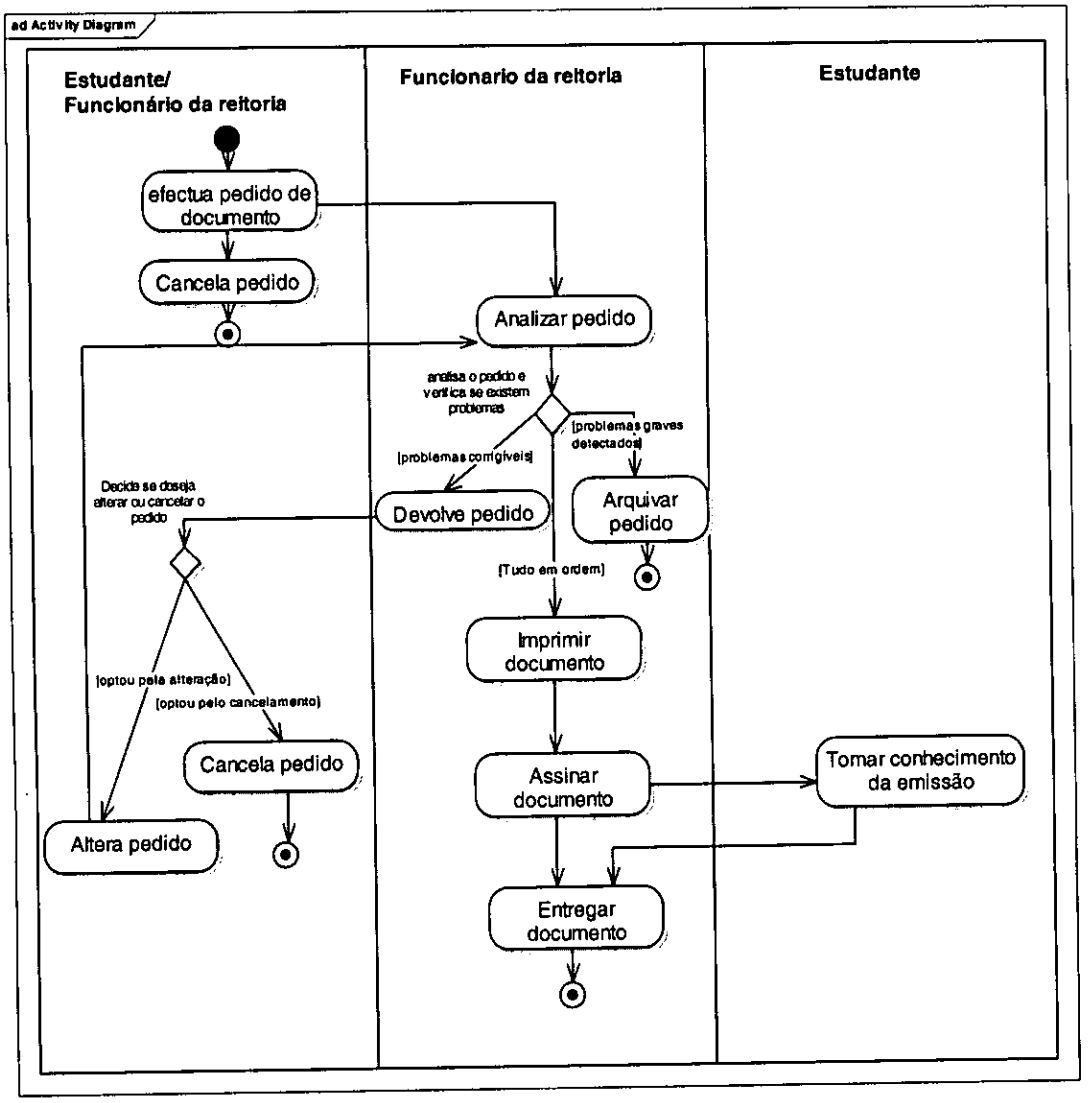

Figura 10 – Processo de pedido e emissão de um documento

ANEXO E — Diagrama de sequência do SEDC

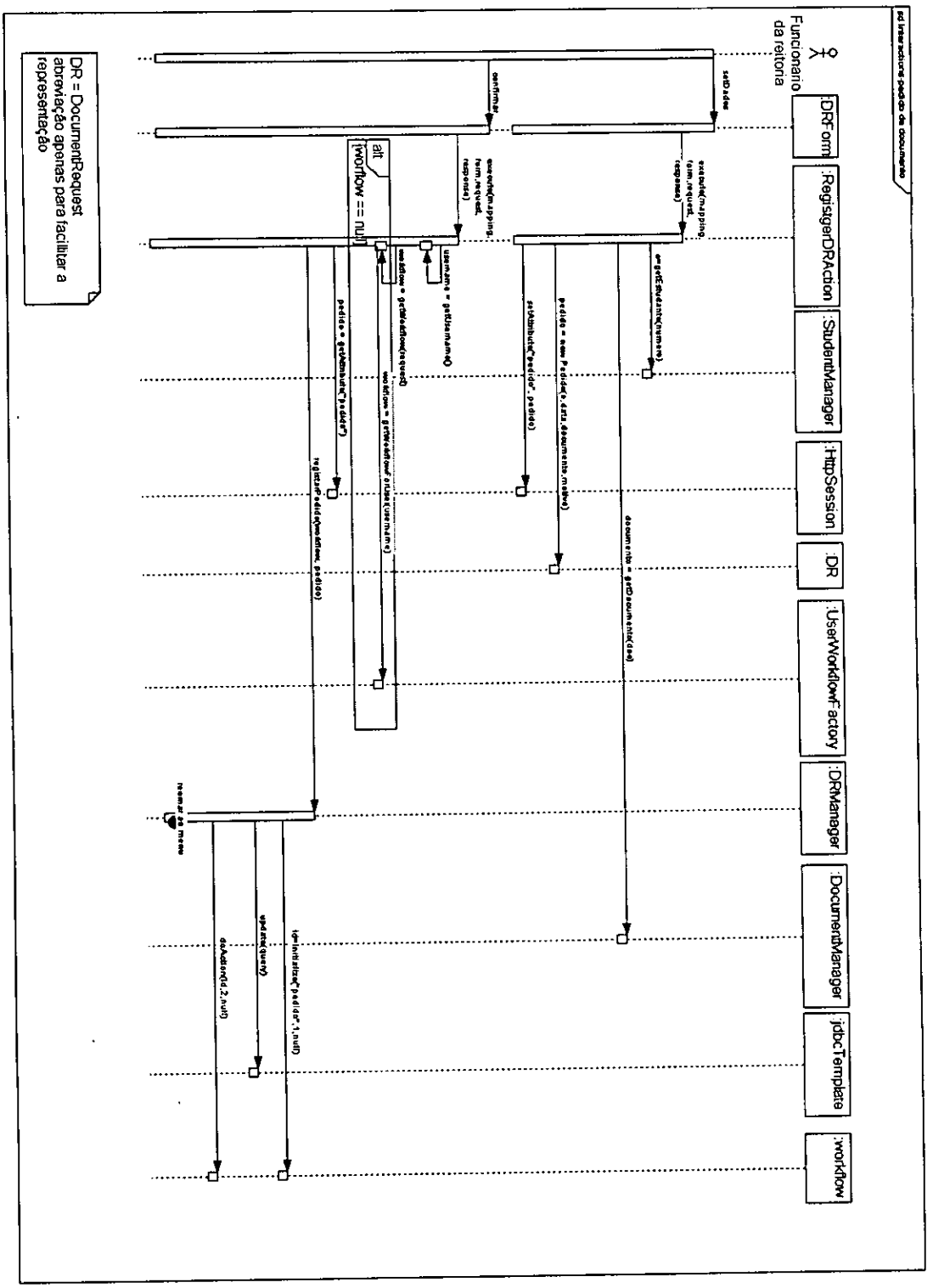

Figura 11 – Diagrama e sequência do processo de pedido de um documento

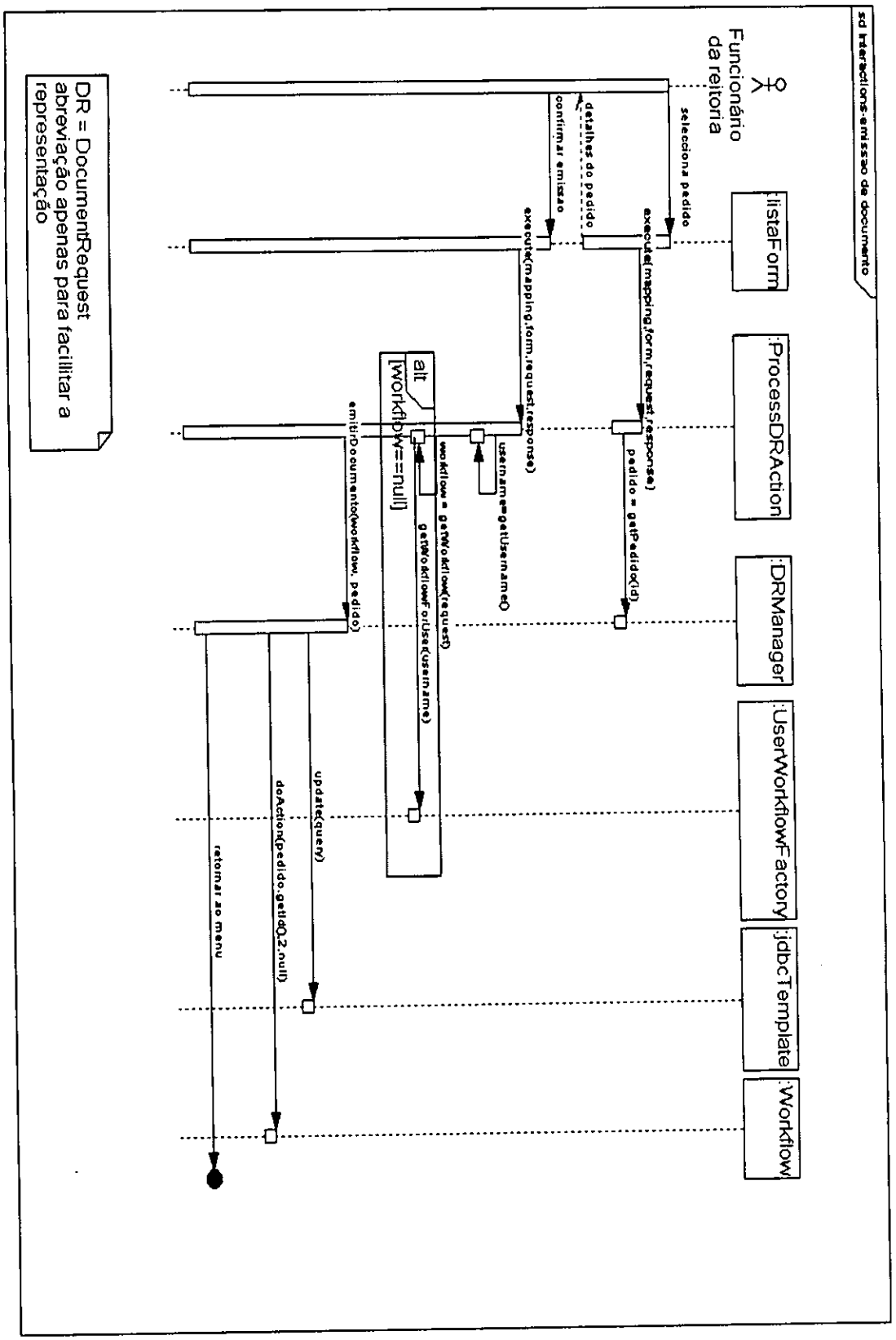

Figura 12 – Diagrama de sequência do processo de emissão de um documento

ANEXO F — Diagrama de estados do objecto Pedido

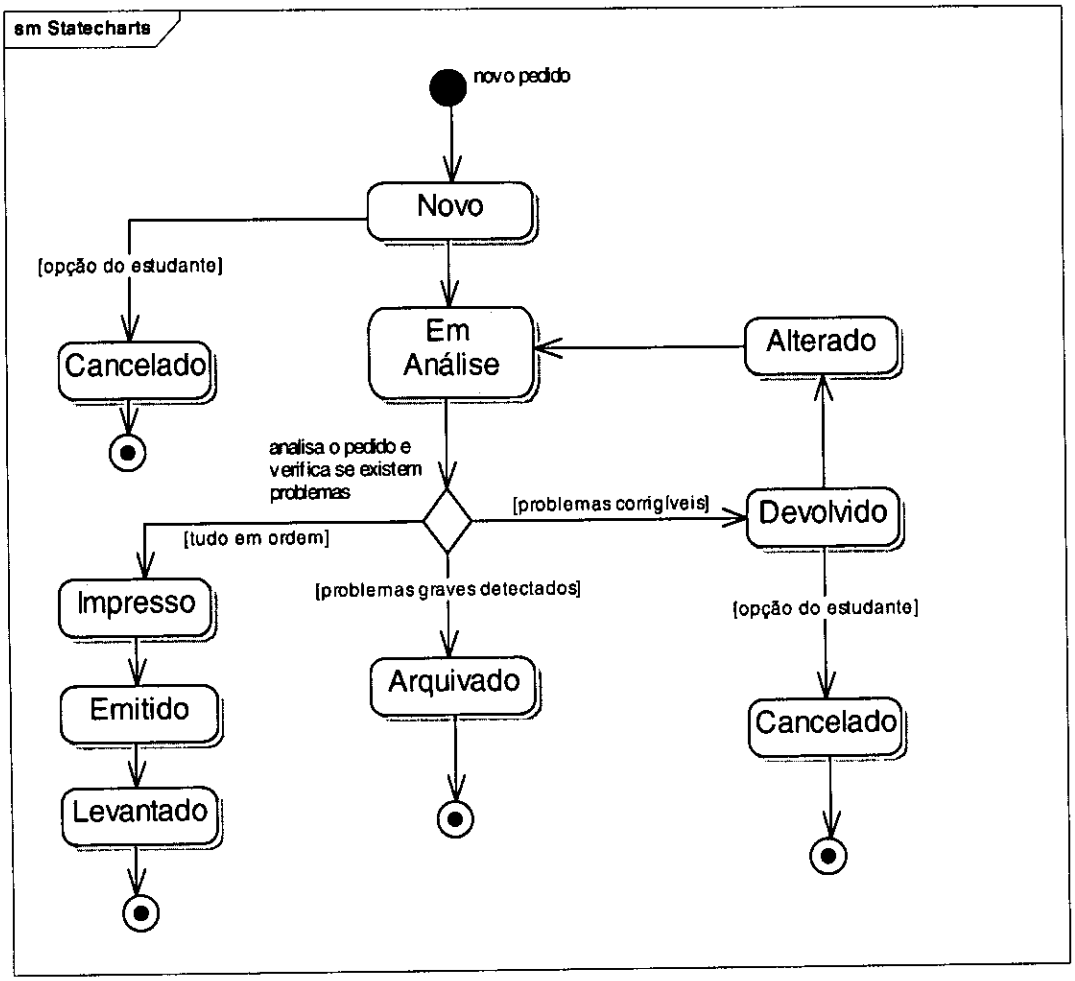

Figura 13 – Diagrama de estados do objecto pedido

ANEXO G — Algumas páginas do SEDC

Nesta secção do documento, são apresentadas algumas páginas do Sistema de Emissão de Declarações e Certificados (SEDC), incluindo a descrição das principais funcionalidades do sistema.

O SEDC, foi concebido para facilitar os processos de requisição e emissão de documentos via *web*. Com a este sistema, a DRA não precisará mais calcular o nível dos estudantes manualmente, e não terá que redigir cartas às faculdades solicitando as notas dos estudantes para a emissão de documentos académicos: Ela (a DRA), reduzirá bastante o trabalho no diz respeito à elaboração de declarações e certificados, preocupando-se apenas com a impressão e assinatura.

O sistema desenvolvido é composto por dois módulos, sendo o primeiro, para o registro das notas dos estudantes e o segundo para o processo de pedido e emissão dos documentos.

Os dois módulos foram implementados com sucesso, faltando, portanto, a introdução de módulos para manutenção de estudantes. Atendendo que a aplicação foi desenvolvida com *Java Server Pages* (JSP), e usando uma arquitectura modular, usando, para o efeito, padrões bem conhecidos mundialmente, falamos do MVC-2 (MVC para web), *frameworks* bem conhecidos como *springframework* e *struts*, e a jstl *tag-libraries*. Desta forma, o sistema pode facilmente ser continuado por outros programadores.

Estrutura do Sistema

O Sistema de Emissão de declarações e certificados da UEM foi concebido para ser usado pelos seguintes utilizadores: *Administrador do Sistema*, *Estudante*, *Funcionário da reitoria* e *Funcionário da faculdade*.

O administrador do sistema é o responsável pela manutenção da aplicação, ele não possui acesso a todas telas, por razões de segurança. O estudante é um usuário que

apenas pode fazer pedido de um documento e cancelar e consultar o estado de apenas seus pedidos. Os restantes usuários, como é o caso dos funcionários da reitoria e das faculdades possuem privilégios definidos segundo as suas actividades dentro do sistema.

Como aceder o sistema?

Para aceder ao sistema, é preciso abrir um navegador/*browser* (Windows explorer, Firefox, ou qualquer outro) e digitar, na barra de endereço, o seguinte "https://localhost:8080/ra/" e clicar na tecla *enter*. A partir deste momento aparecerá uma página de *login*, conforme a figura que se segue.

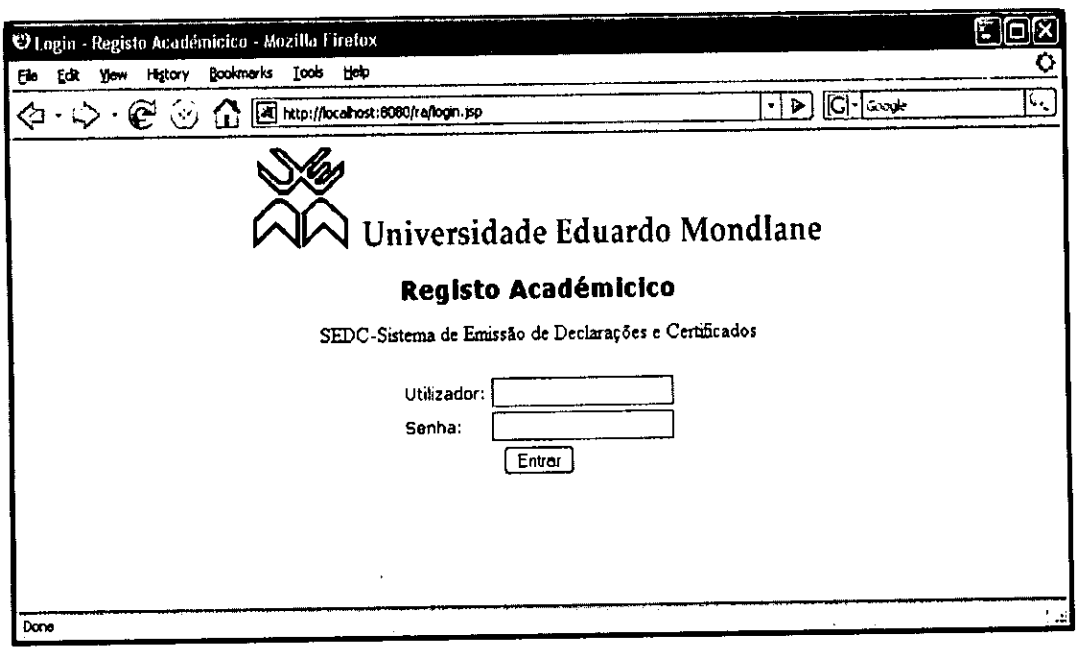

Figura 14 – Página de Login do SEDC

Para aceder ao sistema, o utilizador deverá ter sido previamente cadastrado. Após introduzir os dados da sua autenticação (nome do usuário e senha), o sistema validará os dados, e, se o usuário não for reconhecido, o sistema apresenta a mesma página, acompanhada de uma mensagem de insucesso; caso o usuário seja reconhecido pelo sistema, o sistema autorizará o usuário, apresentando apenas as funcionalidades que lhe dizem respeito.

Permissões e Autorização do usuário

O sistema foi desenhado e implementado tendo em conta as diferentes actividades disponíveis a cada tipo de usuário.

A seguir estão descritas as actividades referentes aos *funcionários da reitoria* (por este ser o actor com maior intervenção no sistema).

O funcionário da reitoria é responsável pelo registro de pedidos de documentos, sua emissão, arquivo bem como a entrega ao estudante.

A figura que se segue, mostra as opções relativas ao funcionário da reitoria. Desta figura, na parte central, podemos ver todos os pedidos, que esperam uma acção do funcionário *logado*, entre elas, devolver, cancelar um pedido, emitir um documento, arquivá-lo e por fim registrar a entrega do documento ao estudante.

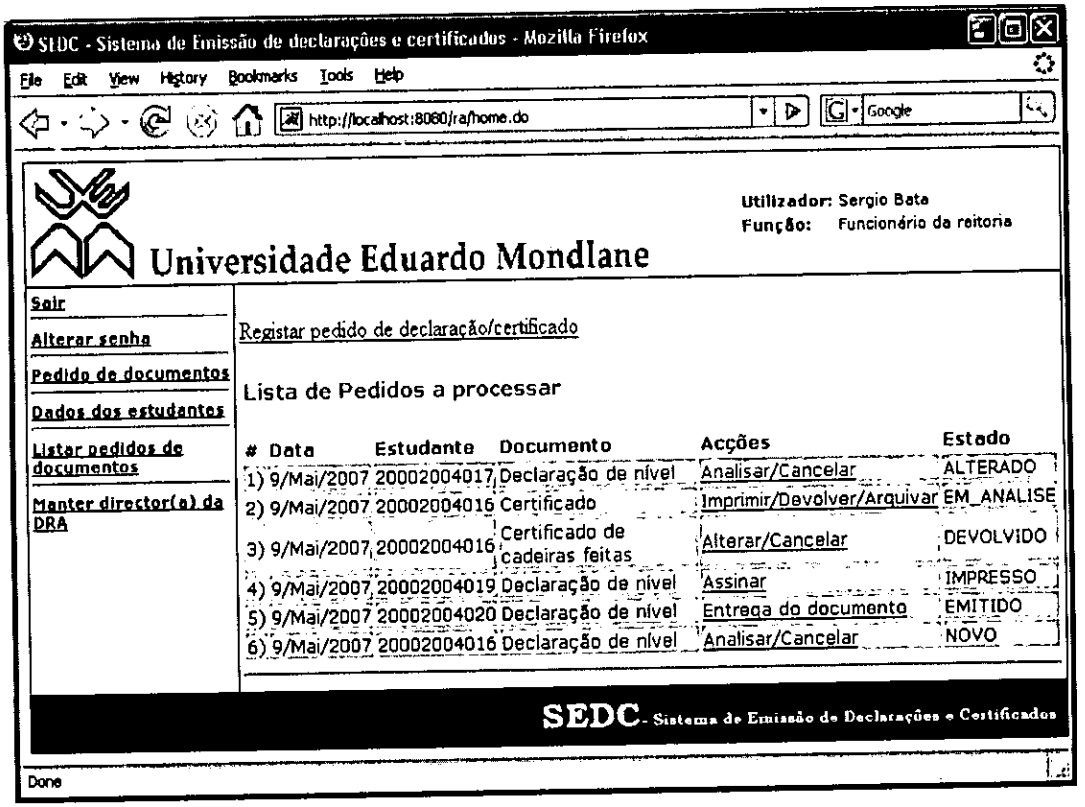

Figura 15 – Opções relativas aos funcionários da reitoria

Os documentos, uma vez entregues ao destinatário (estudante), já não aparecerão mais na parte central da página, isto é, a página central apenas apresenta os documentos que esperam por uma acção do usuário logado, tem acções no fluxo de trabalho.

Se o actor clicar no *link* "*registrar pedido de declarações e certificados*", o sistema abre uma página para o registro do pedido como a que se segue:

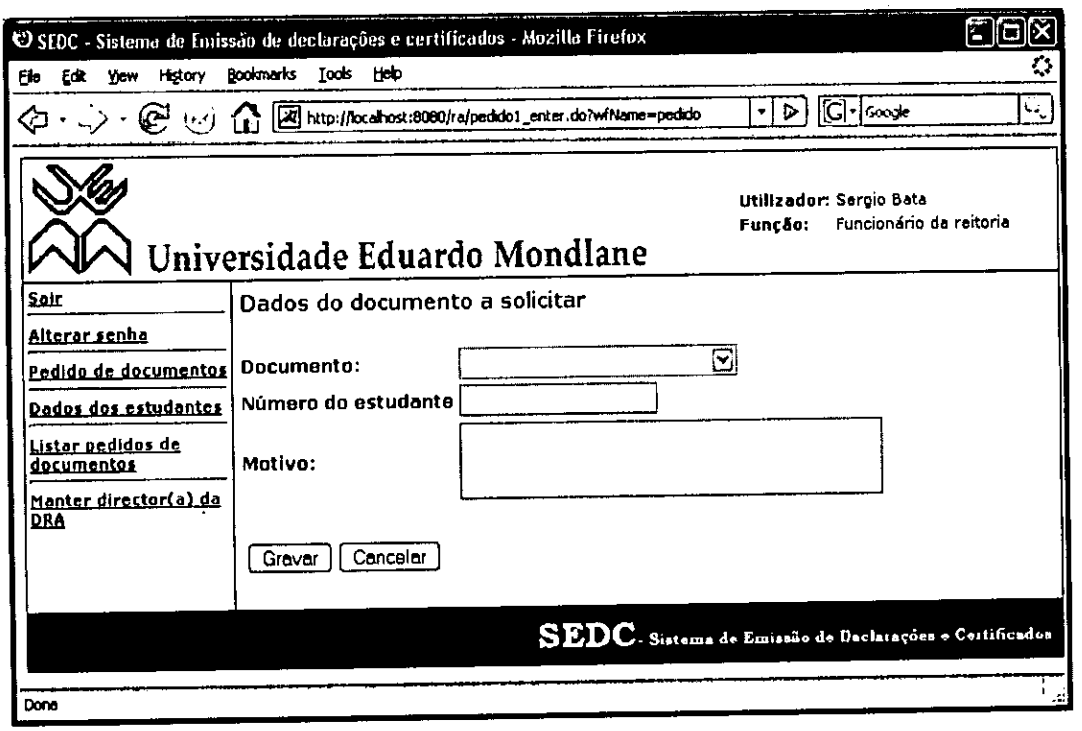

Figura 16 – Página para o pedido de documentos

Nesta página, o usuário deverá preencher os campos e clicar no botão *Gravar*, onde receberá uma tela de confirmação dos dados por ele submetidos. A partir do momento em que se confirma o pedido, este registro passa a constar da tela principal do funcionário da reitoria para o seguimento, com as opções (devolver, cancelar e emitir).

A partir deste ponto, o funcionário da reitoria pode devolver o pedido, com a opção de indicar o motivo da devolução, poderá ainda cancelar o pedido caso o estudante solicite, e, por fim poderá emitir o documento solicitado e dar seguimento ao processo de obtenção do documento.