NATHALIA SAUTCHUK PATRÍCIO

ENGENHARIA DE REQUISITOS EM SOFTWARE PARA E-LEARNING

SÃO PAULO

2013

NATHALIA SAUTCHUK PATRÍCIO

ENGENHARIA DE REQUISITOS EM SOFTWARE PARA E-LEARNING

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Engenharia.

SÃO PAULO
2013

NATHALIA SAUTCHUK PATRÍCIO

ENGENHARIA DE REQUISITOS EM SOFTWARE PARA E-LEARNING

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Engenharia.

Área de concentração:
Sistemas Digitais

Orientador:
Professor Doutor
Edison Spina

SÃO PAULO
2013

FICHA CATALOGRÁFICA

Patrício, Nathalia Sautchuk
Engenharia de requisitos em software para e-learning / N.S. Patrício. -- São Paulo, 2013.
145 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Engenharia de software 2.Engenharia de requisitos 3.Banco de dados 4. E-learning I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

DEDICATÓRIA

A Deus. Aos meus pais Solange e Vanderlei e à minha irmã Carol pelo apoio, incentivo e compreensão durante todo o meu trajeto.

AGRADECIMENTOS

Primeiramente, agradeço a minha família que me apoiou durante todo meu trajeto acadêmico, ajudando-me a concluir mais essa etapa. Aos meus amigos, que me apoiaram nos momentos difíceis, me proporcionaram bons momentos de confraternização e descontração e pelas discussões que ajudaram a inspirar esse trabalho.

Agradeço ao Prof. Dr. Edison Spina, por ter assumido o desafio da orientação dessa pesquisa e aos meus colegas do laboratório KNOMA pelas trocas de ideias que abriram minha mente para a execução desse trabalho.

Agradeço ainda a direção, aos professores e aos alunos da ETEC Raposo Tavares pela participação na pesquisa, o que deu um direcionamento interessante a seguir no estudo de caso apresentado.

Finalmente, a todos os professores e funcionários da Universidade de São Paulo, que direta ou indiretamente contribuíram na minha formação acadêmica.

RESUMO

Na engenharia de software há os modelos tradicionais e os modelos ágeis. Porém, não há um consenso sobre quais são as práticas necessárias para se obter sucesso em um projeto de engenharia de software. O presente trabalho propõe a análise do modelo SEMAT através de um estudo de caso usando esse modelo para a concepção e o levantamento de requisitos de um software para *e-learning*, na área de ensino de banco de dados. A partir desse estudo de caso foi possível verificar a aplicabilidade do modelo para a aplicação, além de demonstrar algumas restrições e adaptações necessárias para o uso nessa área específica.

PALAVRAS CHAVE: Software para *e-learning*, Modelos de Engenharia de software, Engenharia de Requisitos, Banco de dados.

ABSTRACT

In software engineering there are traditional and agile models. However, there is no consensus on what practices are required for a successful software engineering project. This work proposes to analyze the SEMAT model through a study case for the conception and requirements elicitation of an e-learning database education software. From this study case it was possible to verify the model applicability for the application, as well as to understand some restrictions and adjustments required for this specific area.

KEY WORDS: E-learning software, Software Engineering Models, Requirements Engineering, Databases.

LISTA DE FIGURAS

Figura 1: Espiral evolucionária (retirado de Simonette, 2010) 9

Figura 2: Modelo cascata (adaptado de Sommerville, 2007) 12

Figura 3: Modelo em espiral (adaptado de Sommerville, 2007) 15

Figura 4: Fases no processo unificado (adaptado de Sommerville, 2007) 17

Figura 5: Classificação de projetos pela criticalidade e número de envolvidos (retirado de Bassi, 2008) 21

Figura 6: Aplicação de modelos Crystal conforme o número de envolvidos (retirado de Bassi, 2008) 21

Figura 7: Ciclo de desenvolvimento do scrum (retirado de Bassi, 2008) 24

Figura 8: Princípios e práticas do XP (retirado de Locaweb, 2011) 27

Figura 9: As três áreas de interesse (traduzido de SEMAT, 2012a) 30

Figura 10: Alfas do kernel (traduzido de SEMAT, 2012a) 31

Figura 11: Estados dos alfas do kernel (traduzido de Jacobson et al., 2012) 33

Figura 12: Anatomia do cartão (traduzido de Jacobson et al., 2012) 33

Figura 13: Backlog de iteração no SEMAT (traduzido de Jacobson et al., 2012) 35

Figura 14: Espaços de atividade do kernel (traduzido de SEMAT, 2012a) 37

Figura 15: Cartões de estado do alfa requisitos (traduzido de Jacobson et al., 2012) 50

Figura 16: Layout do cartão XP (retirado de Sharp e Robinson, 2007) 94

Figura 17: Camada de controle em projetos de engenharia de software 115

LISTA DE GRÁFICOS

Gráfico 1: Idade dos alunos pesquisados 79

Gráfico 2: Gênero dos alunos pesquisados 79

Gráfico 3: Alunos por turma 80

Gráfico 4: Locais de uso da internet 81

Gráfico 5: Acesso a internet por locais 81

Gráfico 6: Uso de dispositivos para acesso a internet 82

Gráfico 7: Serviços usados na internet 83

Gráfico 8: Dificuldade de aprendizado por tópico 84

Gráfico 9: Interesse em uma ferramenta para ensino de banco de dados 85

Gráfico 10: Utilidade de uma ferramenta para um tópico 85

Gráfico 11: Alunos que fizeram cursos a distância 86

LISTA DE TABELAS

Tabela 1: Checklist do alfa requisitos 51

Tabela 2: Comparação entre sistemas de software para ensino de banco de dados 70

Tabela 3: Estados alvo da iteração 1 72

Tabela 4: Backlog da iteração 1 74

Tabela 5: Número de alunos e questionários por turma 78

Tabela 6: Backlog da iteração 2 92

Tabela 7: Backlog da iteração 3 100

Tabela 8: Backlog da iteração 4 107

SUMÁRIO

FICHA CATALOGRÁFICA ..... *iv*

DEDICATÓRIA ..... *v*

AGRADECIMENTOS ..... *vi*

RESUMO ..... *vii*

ABSTRACT ..... *viii*

LISTA DE FIGURAS ..... *ix*

LISTA DE GRÁFICOS ..... *x*

LISTA DE TABELAS ..... *xi*

SUMÁRIO ..... *xii*

1 INTRODUÇÃO *1*

1.1 Objetivos **3**

1.2 Organização da dissertação **3**

2 ENGENHARIA DE SISTEMAS E DE REQUISITOS *4*

2.1 Engenharia de sistemas **4**

2.2 Engenharia de software **10**

2.2.1 Modelos Tradicionais 11

2.2.1.1 Cascata 11

2.2.1.2 Espiral 13

2.2.1.3 Processo unificado 16

2.2.2 Modelos ágeis 19

2.2.2.1 Crystal 20

2.2.2.2 Scrum 23

2.2.2.3 Programação extrema 25

2.2.3 SEMAT 28

2.2.3.1 Alfas 30

2.2.3.2 Espaços de atividades 37

2.3 Engenharia de requisitos **39**

2.3.1 Estágios da engenharia de requisitos 42

2.3.2 Elicitação de requisitos em modelos tradicionais 44

2.3.3 Elicitação de requisitos em modelos ágeis 45

2.3.3.1 Crystal 46

2.3.3.2 Scrum 47

2.3.3.3 Programação extrema 48

2.3.4 Elicitação de requisitos no SEMAT 49

3 SOFTWARE PARA E-LEARNING E ENSINO DE BANCO DE DADOS 53

3.1 Software para e-learning 53

3.2 Learning Management System 55

3.2.1 Sakai 56

3.2.2 Moodle 57

3.3 Ensino de banco de dados 57

3.4 Software para ensino de banco de dados 60

3.4.1 Exploratorium 60

3.4.2 SQLT-Web 61

3.4.3 Software *The Open University* 62

3.4.4 IDLE 63

3.4.5 Software *Georgia Southern University e Troy University* 65

3.4.6 Software *Dublin City University* 66

3.4.7 Software *Silesian University of Technology* 67

3.4.8 Comparação entre software para ensino de banco de dados 69

4 USO DO SEMAT PARA LEVANTAMENTO DE REQUISITOS EM SOFTWARE PARA E-LEARNING 71

4.1 Primeira iteração 72

4.1.1 Tarefas um e dois 75

4.1.2 Tarefas três e quatro 75

4.1.3 Tarefa cinco 76

4.1.4 Tarefa seis 77

4.1.5 Tarefa sete 77

4.1.6 Tarefa oito 86

4.1.7 Tarefa nove 88

4.1.8 Resultado da primeira iteração 88

4.1.8.1 *Stakeholders* 89

4.1.8.2 Oportunidade 89

4.1.8.3 Requisitos 90

4.1.8.4 Sistema de software 90

4.2 Segunda iteração 91

4.2.1 Tarefa um 92

4.2.2 Tarefa dois 93

4.2.3 Tarefa três 94

4.2.4 Tarefa quatro e cinco 95

4.2.5 Tarefas seis, sete e oito 95

4.2.6 Resultado da segunda iteração 97

4.2.6.1 *Stakeholders* 97

4.2.6.2 Oportunidade 97

4.2.6.3 Requisitos 98

4.2.6.4 Sistema de software 98

4.3 Terceira iteração 99
4.3.1 Tarefa um 100
4.3.2 Tarefa dois 101
4.3.3 Tarefa três 104
4.3.4 Resultados da terceira iteração 104
4.3.4.1 *Stakeholders* 104
4.3.4.2 Oportunidade 105
4.3.4.3 Requisitos 105
4.3.4.4 Sistema de software 106
4.4 Quarta iteração 107
4.4.1 Tarefa um 108
4.4.2 Tarefa dois 108
4.4.3 Tarefa três 109
4.4.4 Tarefa quatro 110
4.4.5 Resultado da quarta iteração 110
4.4.5.1 *Stakeholders* 111
4.4.5.2 Oportunidade 111
4.4.5.3 Requisitos 112
4.4.5.4 Sistema de software 112
5 AVALIAÇÃO E ANÁLISE DO USO DO SEMAT 113
6 CONSIDERAÇÕES FINAIS 116
REFERÊNCIAS 118
REFERÊNCIAS CONSULTADAS 124
APÊNDICES 125
APÊNDICE A - Questionário de pesquisa de perfil de usuário 125
APÊNDICE B - Carta de informação e termo de consentimento à instituição 128
APÊNDICE C - Carta de informação e termo de consentimento ao sujeito de pesquisa 130

1 INTRODUÇÃO

O sistema educacional brasileiro é organizado em etapas. A primeira delas é a chamada educação básica, que é composta pela educação infantil, pelos ensinos fundamental e médio, pela educação profissional e pela educação de jovens e adultos. Após essa etapa, há ainda a educação superior.

Segundo INEP (2012), o Brasil possui 194.932 estabelecimentos de educação básica, nos quais estão matriculados um pouco mais de 50 milhões de alunos, sendo que cerca de 85% dessas matrículas são em escolas públicas.

Nos últimos anos houve um avanço significativo nas políticas públicas de acesso e inclusão digital dos alunos de escolas públicas, através de programas como “Um Computador por Aluno - UCA”, “Programa Nacional de Informática na Educação - ProInfo”, “Computador para Todos”, “Banda Larga nas Escolas”, “Computadores para Inclusão”, entre outros.

Apesar disso, os estudantes brasileiros ainda possuem pouco acesso à Internet e aos computadores em relação aos dos outros países. De acordo com PISA (2010), no quesito número de computadores por aluno nas escolas, dos 57 países participantes, o Brasil está em antepenúltimo lugar, atrás de outros países em desenvolvimento como a Indonésia e a Rússia. A proporção média de computadores por aluno é oito vezes maior entre os países participantes do que a indicada no Brasil.

No ensino fundamental, apenas 44% das escolas públicas possuem laboratório de informática e o acesso à internet está disponível em 42,6% delas. No caso do ensino médio, há laboratórios de informática em 91,8% e acesso à internet em 92,2% das escolas públicas (INEP, 2012).

Com a expansão da inclusão digital das escolas públicas e dos alunos ocorrida nos últimos anos no país, há um aumento na demanda por software para *e-learning*. Porém, se por um lado, essa ampliação é essencial para permitir o acesso a uma grande fonte de informação e conhecimento a uma parcela da população hoje

excluída digitalmente, por outro, o crescimento nessa demanda por software para *e-learning* aumenta a preocupação com a qualidade do que é desenvolvido na área.

Para se desenvolver um software para *e-learning* de qualidade e que atenda aos objetivos educacionais é importante que seus requisitos sejam bem entendidos e corretamente implementados. E, para isso, é necessário usar um modelo de engenharia de software adequado.

Há um grande número de modelos existentes, porém, poucos deles podem ser considerados maduros ou suficientemente testados e validados em situações reais. Muitas vezes, as diferenças entre os modelos não são bem compreendidas e ampliadas artificialmente. Há ainda a prevalência de modas ao invés de uma disciplina de engenharia, a falta de bases teóricas amplamente aceitas e uma divisão entre as práticas da indústria e a pesquisa acadêmica (SEMAT, 2012).

Como uma resposta aos problemas expostos, foi proposto o modelo SEMAT (*Software Engineering Method and Theory*). Apesar desse modelo apresentar características interessantes, ele ainda não foi testado de modo a verificar sua aplicabilidade para a concepção e o levantamento de requisitos de um software para *e-learning*, o que mostra a relevância desse trabalho.

Para o estudo de caso da pesquisa foi escolhido o público do ensino profissional, especificamente a área de ensino de banco de dados. Um dos motivos é a deficiência verificada em software para *e-learning* nessa área. Outro fator para a escolha é o grande crescimento da educação profissional nos últimos anos devido a políticas e ações de fomento do Ministério da Educação. Houve uma expansão no número de matrículas, sendo que, em 2007, era de cerca de 780 mil e atingiu, em 2011, 1.250.900 matrículas; um crescimento de 60% no período. No caso da educação profissional, o censo escolar revela que a participação da rede pública representa 53,5% das matrículas. Na rede pública, o curso mais procurado é o de informática, com 13% do total de matrículas, enquanto na rede privada esse curso ocupa a terceira posição, com 6,5% (INEP, 2012). Logo, o desenvolvimento de software para *e-learning* para atender essa demanda crescente nos cursos técnicos de informática é de suma importância.

1.1 Objetivos

O objetivo principal dessa pesquisa é analisar a aplicabilidade do modelo SEMAT para a concepção e o levantamento de requisitos de um software para *e-learning*, demonstrando assim algumas restrições e adaptações necessárias para o uso nessa área específica. Foi escolhida a área de ensino de banco de dados para o estudo de caso da pesquisa.

O objetivo secundário da pesquisa é a obtenção da especificação de um software para *e-learning* para ensino de banco de dados, podendo ser usada para a aquisição ou o desenvolvimento de um novo software.

1.2 Organização da dissertação

Esta dissertação está organizada em seis capítulos. O capítulo 2 apresenta os conceitos da área de engenharia de sistemas, os principais modelos de processos de engenharia de software e uma discussão de como é feita a elicitação de requisitos em cada um deles. Nesse capítulo é apresentado o modelo SEMAT que será usado no estudo de caso. No capítulo 3 são apresentados conceitos da área de software como ferramenta para *e-learning* e alguns exemplos. Também se apresenta a área de ensino de banco de dados. O capítulo 4 apresenta um estudo de caso usando o modelo SEMAT para a concepção e o levantamento de requisitos de um software para *e-learning*, na área de ensino de banco de dados. No capítulo 5 é feita a análise do estudo de caso e no capítulo 6 são feitas as considerações finais.

.

2 ENGENHARIA DE SISTEMAS E DE REQUISITOS

Nesse capítulo serão apresentados os conceitos da área de engenharia de sistemas, posteriormente, os principais modelos de engenharia de software e os detalhes de como é feita a elicitação de requisitos em modelos tradicionais e ágeis.

2.1 Engenharia de sistemas

A palavra sistema possui uma natureza subjetiva. No senso comum, um sistema é uma combinação de partes para se alcançar um objetivo. Porém, o que não fica explicito nessa definição é que essa combinação e a interação entre componentes de um sistema são complexas (SIMONETTE, 2010).

Utiliza-se a palavra sistema para se referir a formas de organização, sendo que ela não se refere a um objeto que existe no mundo real. Tais formas de organização estão associadas à forma como o homem as reconhecem. Esses sistemas podem ser de cinco classes distintas: sistemas naturais, físicos, abstratos, de atividades humanas e transcendentais. Os sistemas físicos, abstratos e de atividades humanas são criados pelo homem, enquanto os transcendentais são aqueles que estão além da compreensão humana (SIMONETTE, 2010).

Ackoff (1981) define sistema como um conjunto de dois ou mais elementos que satisfazem a três condições:

- O comportamento de cada elemento tem um efeito no comportamento do sistema como um todo;
- O comportamento dos elementos e seus efeitos sobre o todo são interdependentes e
- Subgrupos de elementos são formados, cada um possui um efeito sobre o comportamento do todo e nenhum possui um efeito independente sobre o todo.

Ackoff ainda diz que, como um sistema é um todo que não pode ser dividido em partes independentes, ele possui duas importantes propriedades:

- Todas as partes de um sistema possuem propriedades que deixam de existir quando elas são separadas do sistema e
- Todo sistema possui propriedades que nenhuma das partes possui isoladamente, as propriedades emergentes.

Hitchins (2012) concorda com essa visão ao definir um sistema como um conjunto aberto de elementos complementares e interativos, com propriedades, capacidades e comportamentos emergentes tanto de cada uma das partes individualmente quanto das interações entre essas partes de modo a formar um todo unificado.

Sommerville (2007) diz que há dois tipos básicos de propriedades emergentes em um sistema: as propriedades emergentes funcionais e não funcionais, sendo que as primeiras ocorrem quando todos os componentes trabalham juntos para alcançar um objetivo e as últimas estão relacionadas com o comportamento do sistema em seu ambiente de operação.

Ackoff (1981) diz que as propriedades emergentes não podem ser inferidas pelo conhecimento das propriedades das partes ou das interações entre as partes do todo.

Segundo Zandi (2000), a própria definição de sistema envolve o conceito de propriedades emergentes: quando uma “coisa” é dividida em partes e, com isso perde suas funções e suas propriedades emergentes, pode-se chamá-la de sistema.

A tarefa de se projetar um novo sistema é difícil e complexa justamente pelo fato de não se poder prever as propriedades emergentes de um todo a partir da junção de diversas partes.

Segundo Bertalanffy (1969), esse problema ocorre devido à limitação do reducionismo e determinismo utilizados pela ciência, sendo que essa limitação costuma ser expressa por uma colocação quase metafísica, como evolução emergente ou “o todo é maior que a soma das partes”. O todo ser maior que a soma das partes significa que a interação das partes dá ao sistema propriedades únicas, as propriedades emergentes.

Sendo assim, a chave para resolução de problemas envolvendo sistemas é o pensamento sistêmico, que nada mais é do que o uso do modo de pensar sintético. Ele traz um conjunto de métodos que podem ser utilizados para entender as relações, a simultaneidade, o transiente e as mudanças dessas mesmas relações, buscando o tratamento de problemas para os quais o pensamento analítico não traz respostas satisfatórias (SIMONETTE, 2010). Logo, pode-se dizer que o pensamento sistêmico é um modo de abordar problemas não apenas separando-o em suas partes menores, mas vendo as partes e as relações entre elas como um problema maior.

A disciplina de engenharia de sistemas foi formalizada para lidar com a tarefa de projetar novos sistemas. Ela possui métodos para sintetizar sistemas com as propriedades desejadas, eliminando ou diminuindo aquelas não desejadas. Também guia a engenharia no tratamento de sistemas complexos, nos quais os elementos são diversos e possuem intricadas inter-relações (SIMONETTE, 2010). Segundo Hitchins (2008), a engenharia de sistemas é a arte e a ciência de criar soluções como um todo para sistemas complexos.

Segundo Ackoff (1981), há três formas de se tratar um problema:

- Pode ser resolvido, quando foi encontrada uma resposta que seja “boa o suficiente”; uma que satisfaça;
- Pode ser solucionado, quando foi encontrada uma resposta adequada ou
- Pode ser dissolvido, quando uma mudança na situação, de alguma forma, faz com que o problema desapareça.

De acordo com Hitchins (2008), há duas escolas de engenharia de sistemas para tratar um problema: Escola de Sistemas *Hard* e Escola de Sistemas *Soft*. A Escola *Hard* se foca em solucionar o problema, encontrando uma resposta adequada à situação, enquanto que a Escola *Soft* quer encontrar uma resposta que satisfaça o problema, ou seja, resolver o problema.

O pensamento sistêmico *hard* pode ser aplicado aos sistemas *hard* (sistemas naturais, abstratos ou físicos). Porém, quando aplicado a sistemas de atividades humanas, que são *soft* por natureza, não obtém sucesso devido à complexidade na

identificação precisa do objetivo de um sistema *soft* (SIMONETTE, 2010). Logo, o pensamento sistêmico *hard* é mais apropriado para problemas técnicos bem definidos e o pensamento sistêmico *soft* é mais apropriado para situações confusas e mal definidas, normalmente envolvendo princípios humanos e considerações culturais (CHECKLAND, 1999).

Checkland (2000) explica que há uma diferença de significado da palavra sistema entre as duas formas de pensamento sistêmico. Enquanto no pensamento sistêmico hard ela é usada como um rótulo para algo existente no mundo real, no pensamento sistêmico soft é aplicada ao processo de pensar ou tratar o mundo, ou seja, o processo de pesquisa se torna um sistema.

Pessoas e instituições sociais têm um propósito, e a tecnologia deve servir a esse propósito. Sendo assim, pessoas, instituições sociais e tecnologia resultam em um sistema sociotécnico, no qual há uma infraestrutura social e uma infraestrutura tecnológica (HITCHINS, 2008; SOMMERVILLE, 2007).

Logo, um sistema sociotécnico é uma inter-relação complexa de pessoas e tecnologia, incluindo hardware, software, dados, ambiente físico, pessoas, procedimentos, leis e regulamentações. (MATÉ; SILVA, 2005).

De acordo com Sommerville (2007), os sistemas sociotécnicos têm as seguintes características essenciais:

- <u>Propriedades emergentes</u>: surgem apenas quando as partes de um sistema são colocadas juntas; elas dependem dos componentes do sistema e da relação entre eles e
- <u>Não determinismo</u>: não se pode garantir que um sistema sempre produzirá a mesma saída para determinadas entradas.

Um sistema sociotécnico pode ser modelado como várias entidades humanas e tecnológicas interagindo umas com as outras para alcançar seus próprios objetivos e os coletivos (SANCHES, 2010). Esses sistemas são normalmente de larga escala e podem ser complexos e difíceis de serem modelados devido às propriedades emergentes (LEE; PRITCHETT, 2008).

Quando se pensa no sistema educacional, pode-se dizer que este é um sistema sociotécnico. Uma instituição de ensino ou um curso a distância possui entidades humanas (alunos, professores, etc.) e tecnológicas (computadores, software para *e-learning*, internet, etc.) interagindo entre si para cumprir um propósito. Além disso, podem-se identificar propriedades emergentes e o não determinismo nesse sistema.

Segundo Simonette (2010), os métodos tradicionais de engenharia tratam com sucesso os componentes tecnológicos e os fatores humanos dos sistemas sociotécnicos, mas apresentam dificuldades no tratamento da infraestrutura social deles, seja no mapeamento do humano quanto das instituições sociais. Muitas vezes, considera-se que eles fazem parte do contexto, mas sem pertencerem diretamente ao sistema.

Alguns métodos de engenharia permitem criar sistemas bem-sucedidos nos quais as pessoas sentem que o sistema faz aquilo que elas desejam e necessitam além de apresentarem surpresas por ter funcionalidades além do esperado. Esses métodos também devem respeitar e considerar, durante todo o ciclo de vida do sistema, as dimensões humanas e sociais, indo além do tratamento dos fatores humanos tal como aparecem hoje na literatura (CHAPANIS, 1996; NEMETH, 2004; SADOM, 2004).

Um dos métodos considerados adequado para tratar de sistemas sociotécnicos é a abordagem utilizada pela Escola de Sistemas *Soft* da engenharia de sistemas (SIMONETTE, 2010). Essa visão permite entender o domínio do problema desses sistemas, além de auxiliar na identificação das dimensões humanas e sociais, uma vez que:

- A compreensão do domínio do problema é basicamente uma atividade que envolve a relação entre pessoas, ou seja, lida com atividades humanas e
- Para abranger além dos fatores humanos e possibilitar a identificação das dimensões humanas e sociais, é necessário fazer o uso dos métodos do pensamento sistêmico soft com uma estratégia de abordagem evolucionária, como a proposta por Soares (1986).

Na abordagem evolucionária de Soares (1986), há a interação entre realidade e pensamento e a interação entre problema e a solução conforme mostrado na Figura 1. A partir dessas duas interações, quatro ações geram um ciclo a ser percorrido para resolver um problema:

- Compreensão: O engenheiro constrói um entendimento através de uma representação abstrata para um problema real;
- Concepção: O engenheiro cria uma resposta ao problema para satisfazê-lo no plano do pensamento;
- Realização: Constrói-se a resposta ao problema no plano da realidade e
- Utilização: Insere-se a resposta ao problema no ambiente do problema.

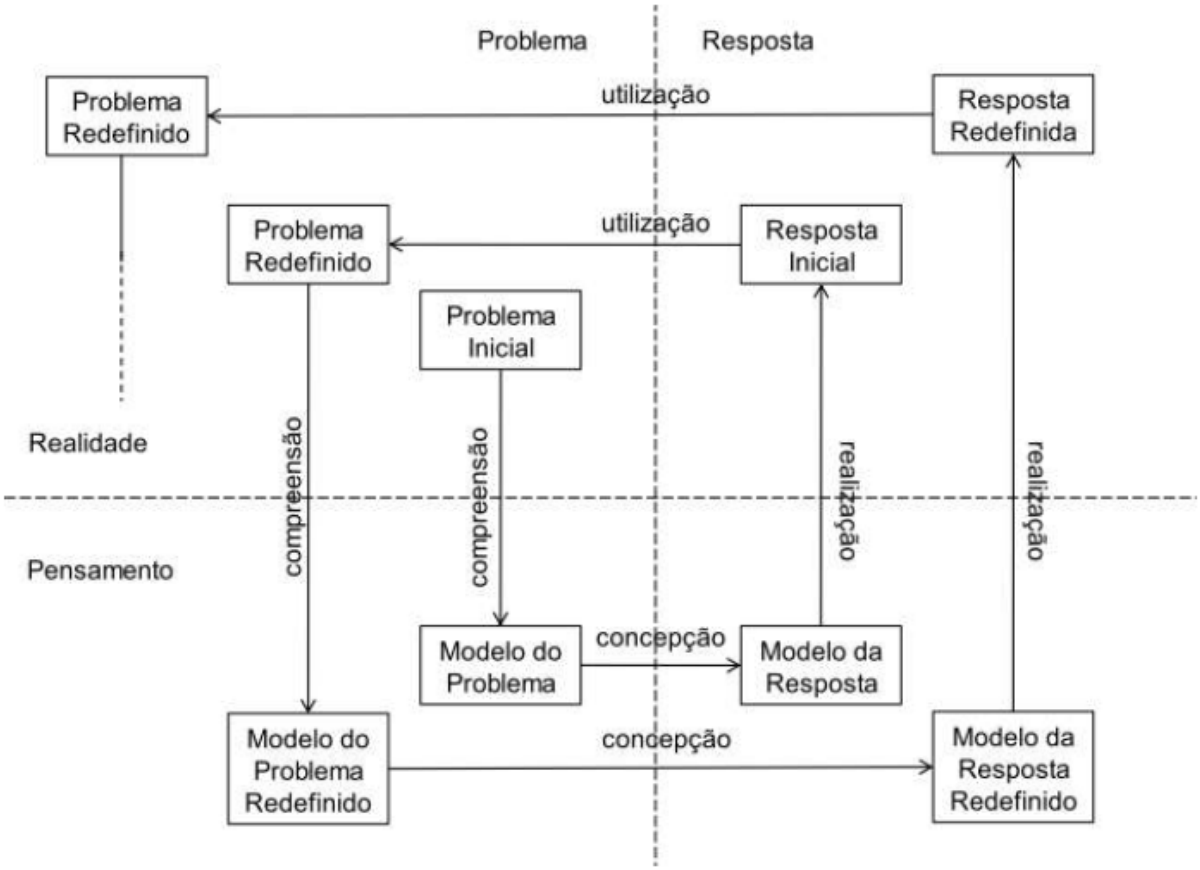

Figura 1: Espiral evolucionária (retirado de Simonette, 2010)

O fato de se estabelecer uma resposta ao problema pode trazer impacto à realidade e, com isso, podem surgir novos cenários que não existiam anteriormente. Isso dá

origem a novas demandas levando a uma redefinição do problema e o ciclo recomeça, sendo esse um processo contínuo.

Durante todo esse ciclo é importante estar atento às dimensões humanas e sociais para o sucesso do sistema, mas a primeira etapa do ciclo de vida, a elicitação de requisitos, é crucial para esse sucesso. Na atualidade, cada modelo de engenharia de software tem a sua própria forma de lidar com a elicitação de requisitos como explicado a seguir.

2.2 Engenharia de software

Uma definição direta para engenharia de software é a aplicação de métodos e a disciplina de engenharia no campo de software. Porém, essa definição pode ser insuficiente, por depender do termo engenharia que, em geral, não está bem definido ou porque falta uma descrição de como a área de aplicação, o software, afeta a definição básica de engenharia.

Por outro lado, o SWEBOK (2011) apresenta outra definição do IEEE *Computer Society*: "A engenharia de software é a aplicação de uma abordagem sistemática, disciplinada e quantificável para o desenvolvimento, operação e manutenção de software, e o estudo dessas abordagens, ou seja, a aplicação da engenharia ao software".

Pode-se também citar a definição da *American Engineers' Council for Professional Development* (1947), que define engenharia como a aplicação criativa de princípios científicos para projetar ou desenvolver estruturas, máquinas, aparatos, ou processos de manufatura, ou trabalhos utilizando-os isoladamente ou em combinação; ou para construir ou operá-los com pleno conhecimento de sua concepção; ou para prever seu comportamento sob condições específicas de operação; tudo respeitando uma função prevista, economia de operação e segurança à vida e propriedade. É possível fazer uso dessa definição para engenharia de software substituindo-se “trabalhos” por “trabalhos de software”.

Há diversos modelos de engenharia de software. Nessa seção serão explorados os principais modelos existentes, divididos em modelos tradicionais e ágeis.

2.2.1 Modelos Tradicionais

Nesse item serão apresentados os três principais modelos tradicionais de engenharia de software, conhecidos como cascata, espiral e processo unificado.

2.2.1.1 Cascata

De acordo com Larman e Basili (2003), o modelo cascata tem início nos anos 1970, quando Winston Royce compartilha suas opiniões sobre como deveria ser feito o desenvolvimento de software.

Pressman (2006) afirma que o modelo cascata sugere uma abordagem sistemática e sequencial para o desenvolvimento de software que começa com a especificação dos requisitos pelo cliente e progride ao longo do planejamento, modelagem, construção e implantação, culminando na manutenção progressiva do software acabado.

Segundo Sommerville (2007), esse modelo é dividido em cinco estágios principais e o resultado de cada um deles é um ou mais documentos aprovados:

1. Análise e definição de requisitos: Os usuários do software são consultados e, a partir disso, os serviços, as restrições e os objetivos do software são definidos detalhadamente e servem como uma especificação;
2. Projeto de sistema e software: Os requisitos são divididos em sistemas de hardware ou de software e uma arquitetura geral é estabelecida. O projeto de software envolve a identificação e a descrição das abstrações fundamentais do sistema de software e suas relações;

3. Implementação e teste de unidade: Um software é codificado como um conjunto de programas ou unidades de programa. O teste unitário tem por objetivo verificar que a unidade atende à especificação;
4. Integração e teste de sistema: As unidades individuais de programa ou os programas são integrados e testados como um sistema completo para garantir que os requisitos de software foram atendidos. Após os testes, o sistema de software é liberado para o cliente e
5. Operação e manutenção: O software é instalado e colocado em operação. A manutenção envolve a correção de erros não detectados nos estágios anteriores, no aprimoramento da implementação das unidades de software e na ampliação dos serviços de software à medida que novos requisitos são identificados.

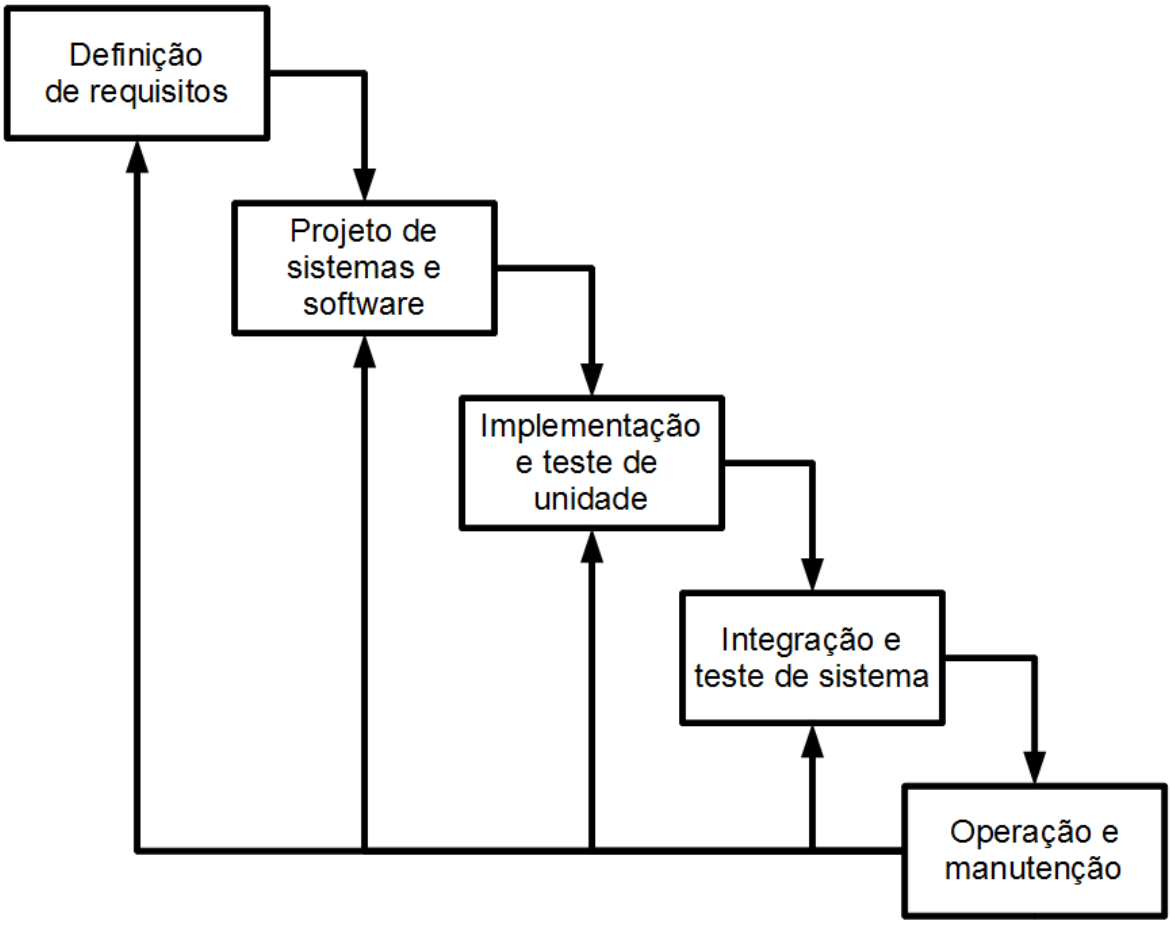

Figura 2: Modelo cascata (adaptado de Sommerville, 2007)

Muitos acham que Winston Royce, ao propor o modelo cascata, recomendou uma única passagem em cascata pela sequência da Figura 2. Mas, na verdade, ele recomenda fazer duas passagens (Larman e Basili, 2003).

O modelo em cascata é um dos mais usados hoje em dia. Larman e Basili (2003) dizem que isso se deve ao fato dele ser mais fácil de explicar e descrever, de que ao usá-lo se tem a ilusão de um processo ordenado, mensurável e responsável e também pelo fato de ser promovido em muitos textos sobre engenharia de software, engenharia de requisitos e gestão, cursos e organizações de consultoria.

Sommerville (2007) afirma que as vantagens do modelo em cascata consistem na documentação produzida em cada fase e sua aderência a outros modelos de processo de engenharia.

Apesar de sua hegemonia no mercado de trabalho, esse modelo vem sendo questionado por diversos motivos: projetos reais raramente seguem o fluxo sequencial proposto, a dificuldade do cliente em estabelecer todos os requisitos explicitamente e a paciência do cliente em esperar o programa ficar pronto, o que ocorre apenas no fim do projeto (Pressman, 2006).

2.2.1.2 Espiral

Apesar de sua hegemonia no mercado de trabalho, esse modelo vem sendo questionado por diversos motivos: projetos reais raramente seguem o fluxo sequencial proposto, a dificuldade do cliente em estabelecer todos os requisitos explicitamente e a paciência do cliente em esperar o programa ficar pronto, o que ocorre apenas no fim do projeto (Pressman, 2006).

Segundo Larman e Basili (2003), Barry Boehm publicou em 1985 um texto descrevendo o modelo espiral de software. O modelo espiral formaliza e torna proeminente o conceito de iterações guiadas por risco e o uso de uma etapa de avaliação de risco em cada iteração.

Esse modelo tem duas características principais. Primeiramente, possui uma abordagem cíclica, que permite que o grau de definição e implementação de um

software seja aumentado incrementalmente enquanto seu grau de risco é diminuído. A outra é um conjunto de marcos de ancoragem que garantem o comprometimento dos interessados com soluções possíveis de serem executadas e mutuamente satisfatórias para o software. Os marcos de ancoragem são uma combinação de produtos de trabalho e condições obtidas ao longo do caminho da espiral. De certa forma, esse modelo combina a natureza iterativa da prototipagem com os aspectos controlados e sistemáticos do modelo em cascata.

Sommerville (2007) considera que cada circuito na espiral está dividido em quatro setores (ilustrado na Figura 3):

1. Definição de objetivos: São definidos os objetivos específicos dessa fase do projeto; as restrições sobre o processo e o produto, além dos riscos de projeto serem identificados e um plano detalhado de gerenciamento ser elaborado. A partir disso, estratégias alternativas podem ser planejadas;
2. Avaliação e redução de riscos: É realizada uma análise detalhada para cada risco de projeto identificado e providências são tomadas para reduzi-los. Por exemplo, um protótipo do software poderá ser desenvolvido para reduzir o risco de que os requisitos não sejam apropriados;
3. Desenvolvimento e validação: Seleciona-se um modelo de desenvolvimento para o software após a avaliação de risco e
4. Planejamento: Revisa-se o projeto e é decidido se haverá prosseguimento para o próximo circuito da espiral. Nesse caso, serão elaborados planos para a próxima fase do projeto.

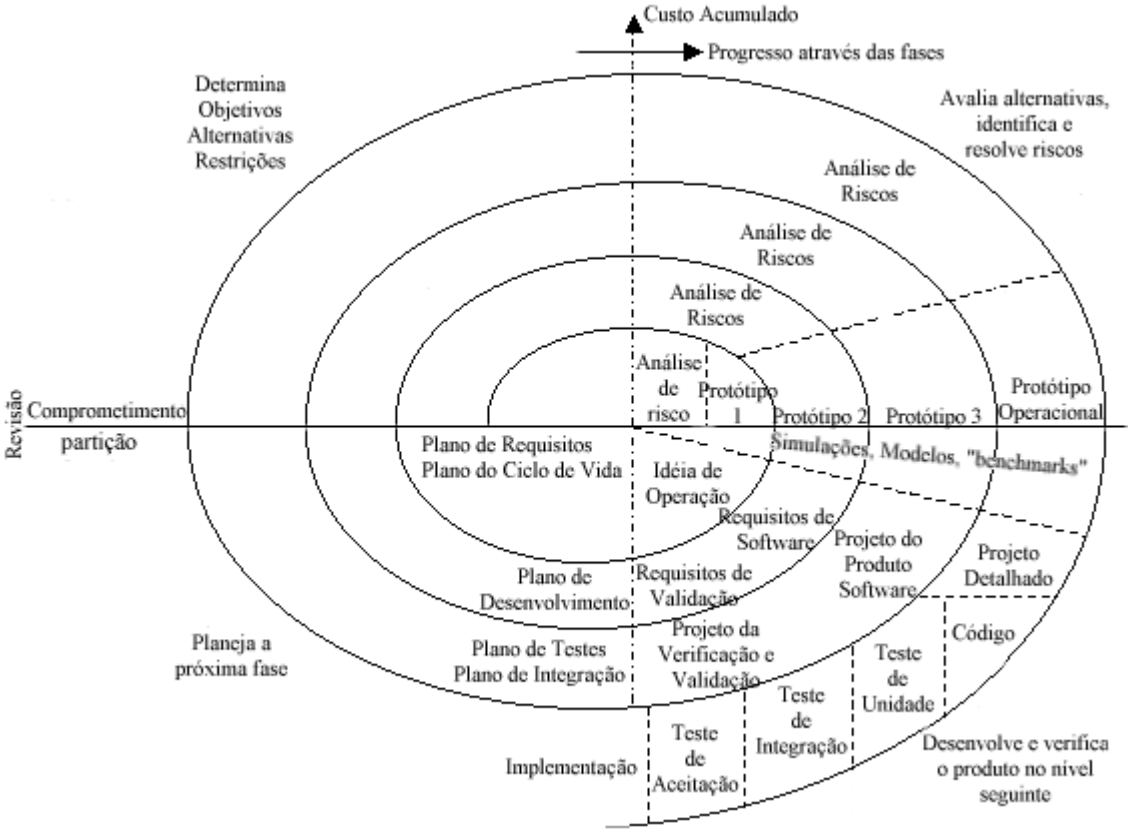

Figura 3: Modelo em espiral (adaptado de Sommerville, 2007)

De acordo com Pressman (2006), a equipe de software realiza atividades que são indicadas por um caminho em volta da espiral em sentido horário começando pelo centro. Além disso, o risco deve ser considerado a cada volta na espiral.

A primeira volta em torno da espiral poderia resultar no desenvolvimento da especificação ou de um modelo de papel do software; as próximas passagens poderiam resultar no desenvolvimento de um protótipo e depois, a cada nova passagem, versões cada vez mais sofisticadas do software seriam desenvolvidas.

Cada vez que se passa pela região do planejamento, o plano de projeto é ajustado, incluindo o custo e o cronograma com base na opinião do cliente após a entrega de uma versão. O número de iterações necessárias para completar o software também é revisto.

Apesar do modelo em espiral ser uma abordagem realista, a sua adoção pode ser difícil no processo de desenvolvimento de software de grande porte, uma vez que o

custo e o cronograma vão sendo alterados ao longo do projeto, dando uma impressão de que não é um processo controlável.

2.2.1.3 Processo unificado

Com a popularização do uso de linguagens de programação orientadas a objetos na década de 1980 e no início de 1990 diversos modelos de análise orientada a objetos (AOO) e projeto orientado a objetos (POO) foram propostos.

De acordo com Pressman (2006), no início dos anos 1990, James Rumbaugh, Grady Booch e Ivar Jacobson começaram a trabalhar em um modelo unificado que combinaria as melhores características de cada um dos modelos já existentes, além de adotar características adicionais propostas por outros especialistas. O resultado desse esforço foi uma linguagem unificada de modelagem (UML). Por volta de 1997, a UML torna-se uma norma industrial para o desenvolvimento de software orientado a objetos.

Nos cinco anos seguintes, Jacobson, Rumbaugh e Booch desenvolveram o Processo Unificado, um modelo de processo para engenharia de software orientada a objetos usando a linguagem UML. O Processo Unificado é também conhecido como Processo Unificado Racional (RUP - *Rational Unified Process*) por causa da Rational Corporation, uma contribuinte pioneira para o desenvolvimento e refinamento do processo.

Segundo Sommerville (2007), o processo unificado possui três perspectivas:

- Dinâmica: Mostra as fases do modelo ao longo do tempo;
- Estática: Mostra as atividades realizadas no processo e
- Prática: Sugere boas práticas a serem usadas durante o processo.

Sommerville (2007) ainda complementa dizendo que o processo unificado possui as seguintes fases:

- Concepção: Identificam-se todas as entidades externas (pessoas e sistemas) que irão interagir com esse software e também se definem quais são essas

interações. O projeto pode ser cancelado logo após essa fase caso a contribuição do software para o negócio não seja significativa;

- Elaboração: O domínio do problema deve ser entendido nessa etapa. A partir disso, é estabelecida uma arquitetura para o sistema e, por fim, desenvolve-se o plano de projeto, identificando-se os riscos principais. Ao fim dessa fase, deve-se obter um modelo de requisitos como, por exemplo, os casos de uso da UML já especificados, uma descrição da arquitetura e um plano de desenvolvimento para o software;
- Construção: Desenvolvem-se as partes do software de forma paralela e integradas. Logo, essa fase está associada ao projeto, programação e teste de software. O resultado final deve ser um sistema de software em funcionamento e a documentação associada pronta para ser liberada aos usuários e
- Transição: Essa fase está relacionada com a transferência do software da comunidade de desenvolvimento para a comunidade de usuários e funcionando corretamente em seu ambiente operacional.

O processo unificado é considerado um processo iterativo e há duas formas de iteração presentes nele. Cada fase pode ser realizada de forma iterativa, com os resultados desenvolvidos de forma incremental. O conjunto total de fases pode também ser realizado incrementalmente, retornando da fase de transição para a de Concepção como pode ser visto na Figura 4.

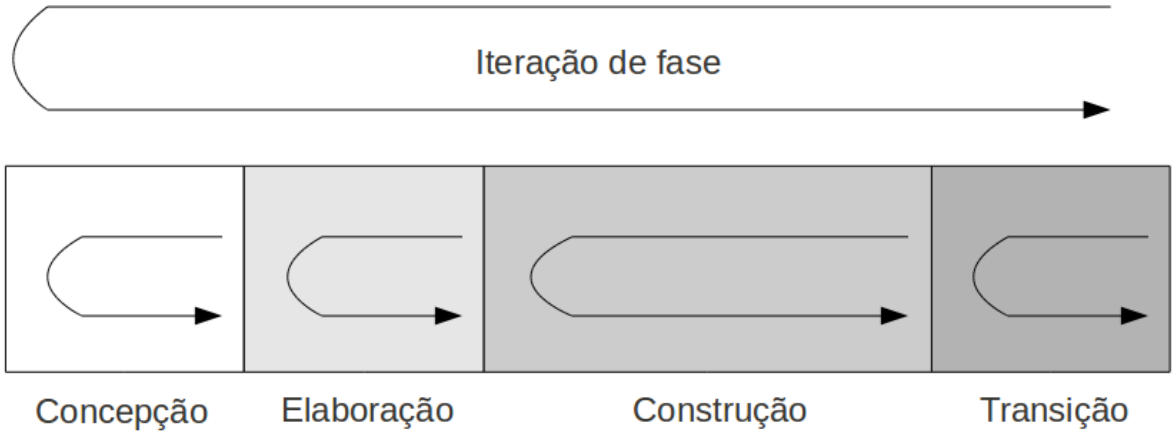

Figura 4: Fases no processo unificado (adaptado de Sommerville, 2007)

A perspectiva estática do processo unificado é constituída por atividades que ocorrem durante o processo de desenvolvimento, chamadas *workflows*. São seis *workflows* de processo principais e mais três de apoio:

- Modelagem de negócios: Modelam-se os processos de negócio usando casos de uso de negócios;
- Requisitos: Identificam-se os agentes que interagem com o software e desenvolvem-se os casos de uso para modelar os requisitos de software;
- Análise e projeto: Cria-se e documenta-se um modelo de projeto usando modelos de arquitetura, modelos de componente, modelos de objeto e modelos de sequência;
- Implementação: Os componentes do software são codificados de forma estruturada em subsistemas;
- Teste: É um processo iterativo realizado em conjunto com a implementação;
- Implantação: Cria-se uma versão do produto, distribui-se aos usuários e instala-se no local de trabalho;
- Gerenciamento de configuração e mudanças: gerencia as mudanças do software;
- Gerenciamento de projetos: gerencia o desenvolvimento do software e
- Ambiente: está relacionado à disponibilização de ferramentas apropriadas de software para a equipe de desenvolvimento.

A perspectiva prática versa sobre boas práticas de engenharia de software. As principais são:

- Desenvolvimento do software iterativamente: Planejam-se os incrementos de software de modo a entregar ao cliente primeiro o que é mais prioritário;
- Gerenciamento de requisitos: Documentam-se os requisitos do cliente e analisa-se o impacto das mudanças que ocorrem neles no software antes de aceitá-las;

- Uso de arquiteturas baseadas em componentes: A arquitetura de software deve estar estruturada com componentes;
- Modelagem visual de software: Devem-se usar os modelos gráficos da UML para modelar as visões estática e dinâmica do software;
- Verificação da qualidade do software: Deve-se garantir que o software atenda aos padrões de qualidade e
- Controle de mudanças no software: As mudanças do software precisam ser gerenciadas, usando para isso ferramentas apropriadas.

2.2.2 Modelos ágeis

Cronologicamente, os modelos tradicionais foram os primeiros a surgir. Porém, ao longo do tempo, perceberam-se algumas dificuldades impostas por esses modelos de desenvolvimento de software. O principal problema dos modelos tradicionais é a dificuldade em lidar com mudanças, seja de requisitos, de escopo ou de tecnologia. De acordo com Martinazzo (2011), o foco para um modelo de desenvolvimento capaz de superar esse problema é, justamente, preparar uma equipe para as mudanças inevitáveis de um projeto de software.

Para superar algumas dessas dificuldades identificadas nos modelos tradicionais, modelos de desenvolvimento com uma etapa de especificação inicial de software menos detalhada surgiram nos anos de 1990 (LARMAN; BASILI, 2003). Esses modelos são conhecidos como modelos ágeis.

Existem diversos modelos ágeis de engenharia de software. Apesar de possuírem práticas distintas entre eles, representantes desses modelos entraram em consenso sobre alguns princípios considerados como determinantes para a obtenção de bons resultados no desenvolvimento de software e publicaram o Manifesto Ágil. Esse manifesto defende quatro premissas (BECK et al., 2001):

- Indivíduos e interações são mais importantes do que processos e ferramentas;

- Software funcionando é mais importante do que documentação completa;
- Colaboração com o cliente é mais importante do que negociação de contratos e
- Adaptação a mudanças é mais importante do que seguir plano inicial.

Nesse item serão apresentados três modelos de engenharia de software considerados ágeis: crystal, scrum e programação extrema.

2.2.2.1 Crystal

Crystal é o nome de uma família de modelos descritos por Alistair Cockburn e Jim Highsmith. Segundo Cockburn (2002), cada um dos modelos é representado por uma cor: quanto mais escura a cor do modelo, maior é o nível de formalismo a ser usado no projeto, maior o tamanho da equipe de projeto e maior a criticalidade. A criticalidade se define pelo tipo de perda que um defeito de software causa ao se manifestar. Por exemplo, a Crystal *Clear* é o modelo usado para projetos com menor criticalidade, menor nível de formalidade e comporta equipes de 1 a 8 pessoas. Já o Crystal *Red* é um dos mais formais, para projetos com maior criticalidade e comporta equipes de 50 a 100 membros.

Pressman (2006) fala que a família Crystal possui elementos centrais comuns a todos os modelos, mas também há elementos específicos de cada modelo, como, por exemplo, os papéis desempenhados dentro da equipe, padrões de processos, produtos de trabalho e práticas. Na verdade, esses modelos são um conjunto de processos ágeis que se mostraram efetivos para diferentes tipos de projeto. A intenção é permitir que equipes ágeis selecionem o membro da família Crystal mais apropriado para o seu projeto e ambiente.

Antes de começar o trabalho, é necessário classificar o projeto usando duas características: a criticalidade e o tamanho da equipe. O número de pessoas é importante, uma vez que influencia nos meios de comunicação necessários para garantir um nível adequado de informação no projeto (Bassi, 2008). Quanto mais pessoas envolvidas, mais artefatos e formalismos são necessários para possibilitar a

comunicação. Pode-se ver na Figura 5, a classificação de projetos levando em conta a criticalidade e o tamanho da equipe.

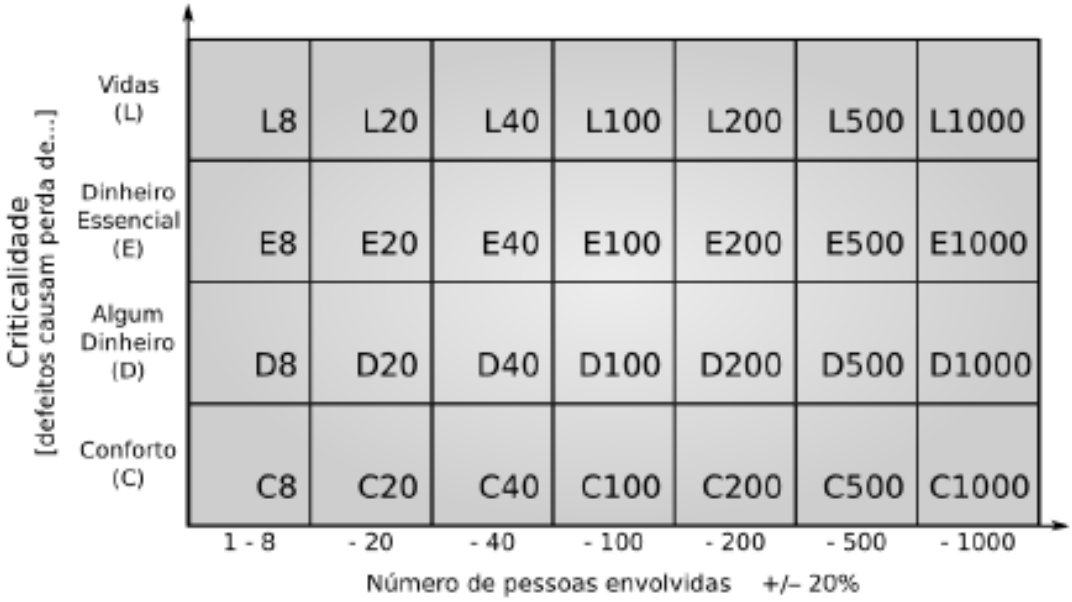

Figura 5: Classificação de projetos pela criticalidade e número de envolvidos (retirado de Bassi, 2008)

Pode-se observar na Figura 6 a quais tipos de projeto cada modelo se aplica. Vale ressaltar que esses modelos não se aplicam a projetos em que há riscos críticos, como o de perda de vidas, tais como sistemas de controle vitais.

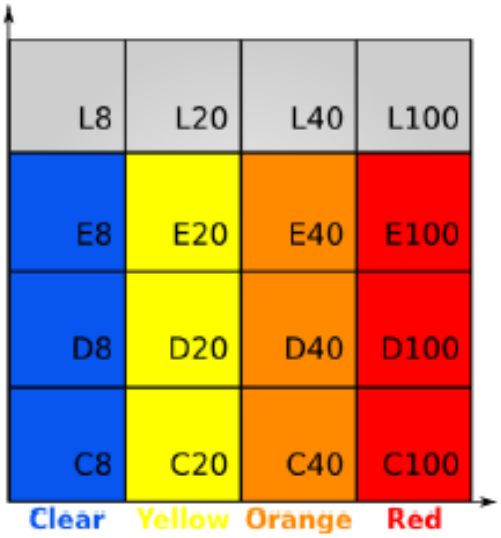

Figura 6: Aplicação de modelos Crystal conforme o número de envolvidos (retirado de Bassi, 2008)

Segundo Bassi (2008), a família Crystal se apoia em um conjunto de propriedades, estratégias e práticas. As propriedades definem as premissas do modelo que devem ser seguidos durante o desenvolvimento. Três delas são as principais: entregas frequentes, comunicação intensa e melhora por reflexão. Porém, para cada modelo da família, as propriedades podem ser aplicadas de acordo com as características e o escopo do projeto. Para valorizar essas propriedades, são sugeridas cinco estratégias:

- Exploração de 360 graus: A equipe identifica os pontos de maior dúvida no projeto e explora todas as possibilidades. Podem ser desenvolvidas amostras da solução com as possíveis tecnologias a serem usadas;
- Vitória antecipada: Deve-se estipular no início do projeto um objetivo que seja rápido de ser atingido. Isso estimula a equipe do projeto, bem como dá uma sensação positiva aos usuários e patrocinadores;
- Esqueleto que anda: Pode haver estruturas ou operações essenciais para o software que podem ser entendidas como o esqueleto do projeto (por exemplo, a comunicação entre tecnologias diferentes). Esse esqueleto deve estar funcional o quanto antes para permitir testes durante a implementação;
- Rearquitetura incremental: A arquitetura do software vai sendo evoluída ao longo do projeto a partir do Esqueleto que Anda. O esqueleto possui as estruturas básicas que podem ser substituídas durante o crescimento do software e
- Radiadores de informação: Painéis são colocados em lugares de fácil visualização pelos envolvidos com informações relevantes sobre o projeto.

As práticas propostas são nove, porém apenas uma é obrigatória que é o *reflection workshop*. O *reflection workshop* é uma reunião periódica focada em três questões principais: “o que deve ser mantido?”, “onde estamos tendo problemas?” e “o que vamos tentar na próxima iteração?”. O resultado do *workshop* são três listas com as respostas para essas perguntas apresentadas em um radiador de informações que fica exposto no ambiente de trabalho.

2.2.2.2 Scrum

Segundo Larman e Basili (2003), foi na década de 1990 que Jeff Sutherland e Ken Schwaber começaram a aplicar o que, posteriormente, foi chamado de scrum. Foi inspirado em métodos usados no desenvolvimento de produtos em empresas como Honda®, Canon® e Fujitsu® nos anos 1980.

O scrum é um modelo ágil para gerência de projetos de software, sem determinar como a equipe executará as tarefas de programação, permitindo a integração com outros modelos ágeis específicos para práticas de programação, como a programação extrema (Bassi, 2008).

Para seguir as práticas do scrum são usados cartões de funcionalidades e gráficos de acompanhamento. Esses cartões são agrupados em *backlogs*. O *backlog* do produto contém uma lista de todos os cartões de funcionalidades do produto que ainda não foram implementadas. O cliente escolhe um subconjunto de funcionalidades a partir do *backlog* do produto para ser implementado no ciclo atual, formando o *backlog* selecionado. A partir desses cartões são especificadas tarefas menores e priorizadas, o que forma o *backlog* do *sprint*. Há ainda o *backlog* de impedimentos que lista os obstáculos não pertencentes ao contexto de desenvolvimento.

Nesse modelo, um projeto é dividido em três fases, o planejamento, o *sprint* e a avaliação. No planejamento são realizadas reuniões (*Sprint Planning*) com o objetivo de estimar, definir a meta do *sprint* e definir as tarefas necessárias para atingir essa meta. Ou seja, um representante do cliente, chamado aqui de dono do produto, prioriza os itens do *backlog* do produto e a equipe seleciona as atividades que serão capazes de implementar durante o *sprint* que se inicia. As tarefas alocadas em um *sprint* são transferidas para o *backlog* do *sprint*.

O sprint é a fase de implementação em que a equipe irá trabalhar para atingir o objetivo de entregar a parte do software que está no backlog do sprint. Normalmente, um sprint possui a duração de 30 dias corridos, podendo ser alterada de acordo com o projeto e com a equipe. A Figura 7 mostra o ciclo de desenvolvimento do scrum.

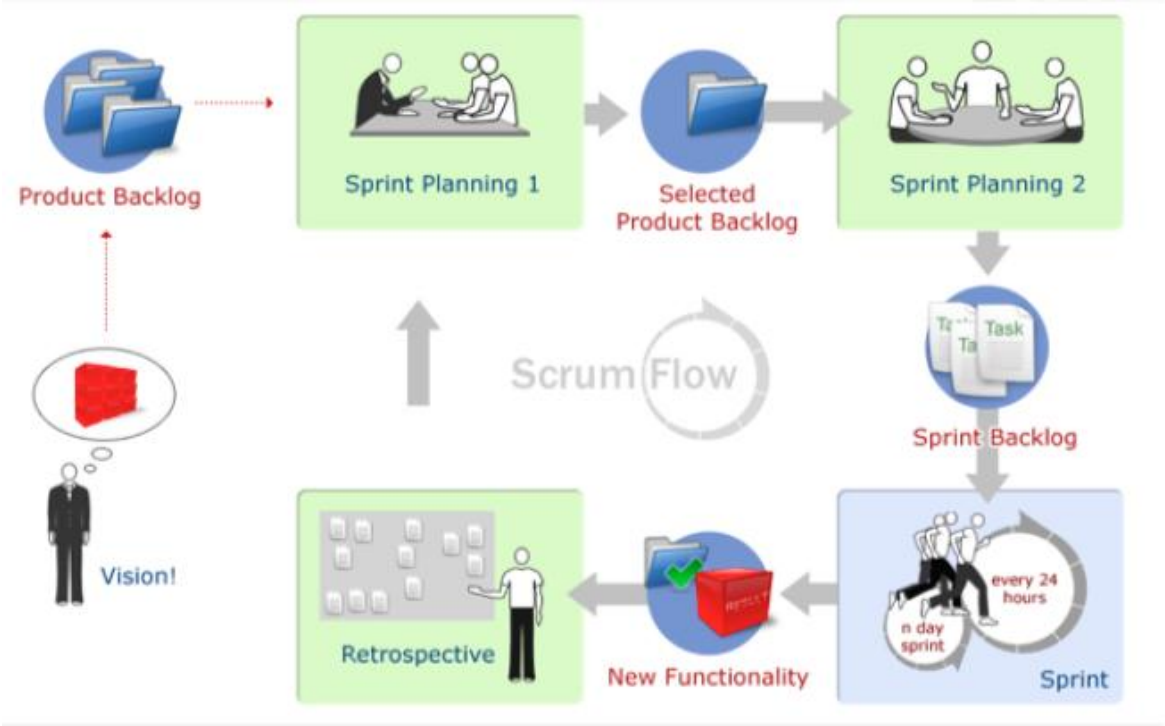

Figura 7: Ciclo de desenvolvimento do scrum (retirado de Bassi, 2008)

Durante o *sprint*, a equipe faz diariamente uma breve reunião (normalmente de manhã e com duração de 15 minutos), chamada de *daily scrum*. O objetivo dessa reunião é socializar o conhecimento, identificar problemas potenciais o mais cedo possível e priorizar o trabalho do dia que se inicia. Segundo Pressman (2006), três questões-chave são formuladas e respondidas por todos os membros da equipe:

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

Essa reunião é liderada pelo *scrum master*. Ele não é um líder de equipe no sentido convencional da palavra; atua como uma barreira entre a equipe e possíveis influências externas que possam distraí-la das tarefas definidas para um determinado *sprint*. Ele também garante que o processo scrum está sendo usado como pretendido.

Ao final de cada *sprint*, a equipe apresenta ao dono do produto as funcionalidades implementadas em uma reunião de revisão do *sprint*, ele testa cada item para verificar se atende às expectativas e determina se a meta foi atingida.

Finalmente, faz-se uma reunião de retrospectiva, na qual os desenvolvedores refletem sobre duas questões principais: quais práticas funcionaram bem e o que poderia ser melhorado no próximo *sprint*. Os resultados da retrospectiva são três listas, sendo a primeira com as ações que trouxeram resultados positivos e devem ser mantidas; outra com as ações sem resultado ou problemas que reduziram o rendimento e a terceira com mudanças e novas práticas para usar no próximo *sprint*, eliminando os problemas apontados (Bassi, 2008).

2.2.2.3 Programação extrema

Segundo Larman e Basili (2003), a programação extrema (em inglês, *eXtremme programming* ou XP) foi criada em 1996 por Kent Beck quando este assumiu a liderança no projeto do software de folhas de pagamento da Crysler.

Para Beck e Andres (2004), a programação extrema é um método leve de desenvolvimento de software usado quando os requisitos são vagos ou mudam constantemente e é adequado a equipes pequenas ou médias. Pela definição de seu autor, a programação extrema é leve por ser focada na realização das tarefas que criam valor ao cliente. O principal objetivo desse modelo é o desenvolvimento de software com qualidade, por ser focado nas melhores práticas de programação, comunicação clara e trabalho em equipe. Segundo Bassi (2008), a documentação da programação extrema é minimalista, sendo que, em um cenário ideal, apenas o código e os testes a compõem. Além do código, apenas cartões de história e radiadores de informação são recomendados.

A programação extrema assume a presença constante das mudanças durante o processo de desenvolvimento e propõe uma série de práticas para lidar com elas. Isso contraria uma dos preceitos mais difundidos nos modelos tradicionais de engenharia de software que é a possibilidade de prever todos os passos necessários

para o desenvolvimento, através de um detalhado levantamento de características do problema a ser resolvido e da solução a ser desenvolvida.

Segundo Sommerville (2007), outro preceito fundamental da engenharia de software tradicional é que se deve projetar para a mudança. Isso significa que as mudanças futuras no software devem ser antecipadas e deve-se projetá-lo de tal maneira que essas mudanças possam ser implementadas facilmente. Porém, a programação extrema discorda desse princípio alegando que projetar para a mudança é, geralmente, um esforço inútil. As mudanças antecipadas, muitas vezes, não ocorrem e as solicitações de mudança realizadas são completamente diferentes.

Por outro lado, não antecipar mudanças pode causar problemas, pois essas mudanças se tornam mais difíceis de implementar e tendem a degradar a estrutura do software. Para solucionar esse problema, a programação extrema defende que a equipe de desenvolvimento deve procurar constantemente por possíveis melhorias no software, implementando-as imediatamente, o que é conhecido como refatoração. Tendo isso em vista, o software deve ser sempre fácil de compreender e de alterar quando novas histórias são implementadas.

A programação extrema é descrita por meio de valores, princípios e práticas, como se pode ver na Figura 8. As práticas são um conjunto de técnicas que podem ser aplicadas no dia a dia de trabalho da equipe de desenvolvimento de software. Os valores são um conjunto de regras de conduta consideradas como válidas no relacionamento da equipe com o trabalho e entre si. Os valores fundamentam as práticas, porém, eles são universais e independem do contexto do desenvolvimento de software, estando assim distantes das práticas. A ligação entre os valores e as práticas são os princípios, que trazem orientações para um contexto específico.

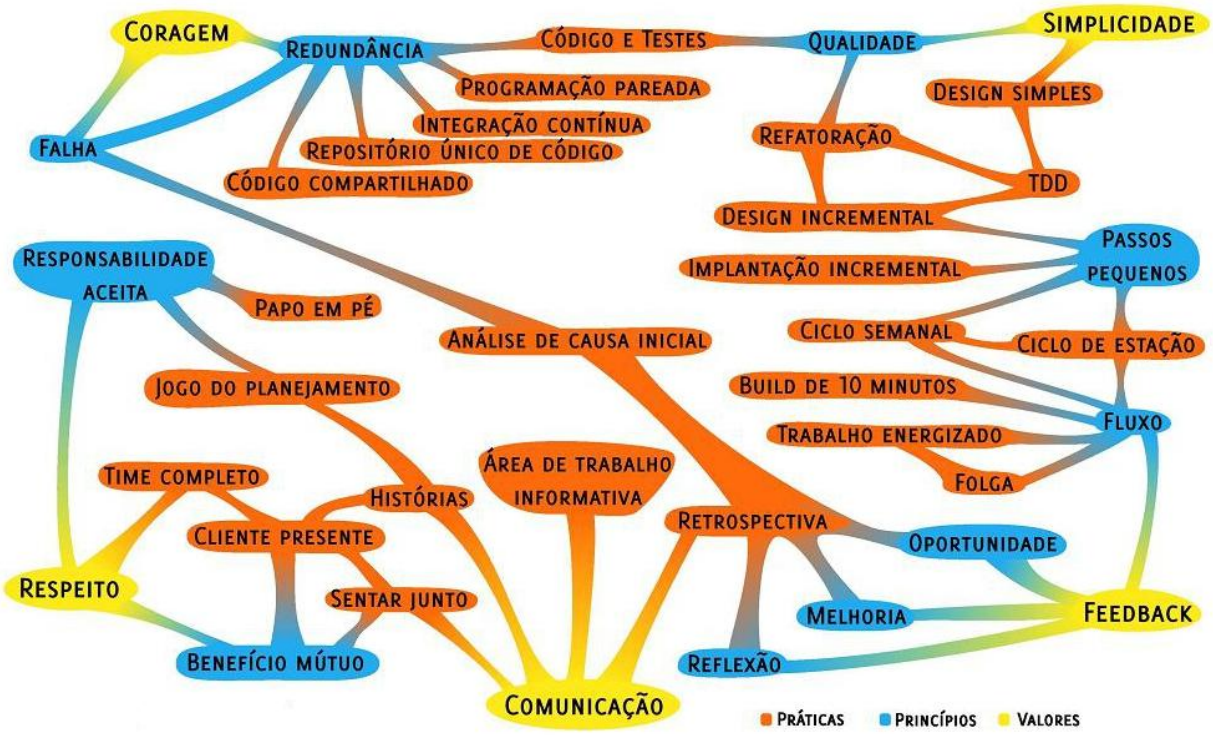

Figura 8: Princípios e práticas do XP (retirado de Locaweb, 2011)

Segundo Bassi (2008), o processo da programação extrema possui três etapas. Na fase de exploração, os programadores têm contato direto com o cliente para conhecer suas necessidades. Uma vez que a equipe adquire conhecimento sobre o que deverá ser desenvolvido, ela pesquisa e discute as tecnologias e ferramentas que podem prover a solução mais adequada. Ainda nessa fase, a equipe desenvolve pequenos aplicativos para avaliar as tecnologias em questão. Na fase de planejamento, são feitos cartões que servem para que os clientes descrevam as funcionalidades que desejam no software, chamadas de histórias. Com as tecnologias escolhidas, os programadores estimam o volume de trabalho de cada história e anotam no próprio cartão. Esses cartões são priorizados e agrupados de forma que as histórias neles descritos constituam uma versão do software. Depois, a equipe faz o planejamento das iterações, separando os cartões da versão em grupos menores: as iterações. Na fase de implementação, os programadores usam as práticas do modelo para implementar as histórias descritas nos cartões com os quais se comprometeram em uma iteração.

2.2.3 SEMAT

Como visto nesse capítulo, na área de engenharia de software há um grande número de modelos. Apesar disso, poucos deles são maduros ou foram suficientemente testados e validados em situações reais. Muitas vezes, até mesmo as diferenças entre modelos são pouco compreendidas e artificialmente ampliadas. Há ainda a prevalência de modas ao invés de uma disciplina de engenharia, a falta de bases teóricas amplamente aceitas e uma divisão entre as práticas da indústria e a pesquisa acadêmica (SEMAT, 2012).

Para se opor a esse cenário, foi lançada a iniciativa SEMAT (*Software Engineering Method and Theory*) que pretende redefinir a engenharia de software baseada em uma teoria sólida, princípios comprovados e melhores práticas que (SEMAT, 2012):

- Inclua um *kernel* de elementos amplamente definidos, extensível para usos específicos;
- Aborde tanto questões de tecnologia quanto de pessoas;
- Seja suportada pela indústria, academia, pesquisadores e usuários e
- Tenha suporte a extensões em face da mudança dos requisitos e da tecnologia.

Essa iniciativa foi originada em dezembro de 2009 por Ivar Jacobson, Bertrand Meyer e Richard Soley.

Na iniciativa SEMAT, a ideia fundamental é a do *kernel*. Esse *kernel* deve possuir a descrição dos elementos que são essenciais a todos os esforços de engenharia de software. Segundo Jacobson et al. (2012), o *kernel* possui os seguintes tipos de elementos:

- Alfas (*Alphas*): Representações das coisas essenciais a serem usadas no trabalho. Os alfas fornecem descrições do tipo de coisas que uma equipe irá gerenciar, produzir e usar no processo de desenvolvimento, manutenção e suporte de um bom software. Eles também podem ser vistos como elementos essenciais do esforço de engenharia de software que são relevantes para a avaliação do progresso e saúde do esforço;

- Espaços de atividade (*Activity Spaces*): Representações das coisas essenciais a fazer. Os espaços de atividade fornecem descrições dos desafios que uma equipe enfrenta no desenvolvimento, manutenção e suporte de sistemas de software, e os tipos de coisas que a equipe vai fazer para superá-los e
- Competências (*Competencies*): As capacidades essenciais necessárias. Podem ser pensadas como recipientes genéricos para habilidades específicas. Habilidades específicas como, por exemplo, programação Java, não são parte do *kernel*, porque essa habilidade não é essencial em todos os esforços de engenharia de software. Caberá às equipes individuais identificar as habilidades específicas necessárias para seu esforço de software específico.

Segundo SEMAT (2012a), o kernel é organizado em três áreas de interesse, cada uma representando um aspecto específico da engenharia de software, como pode ser visto na Figura 9. São elas:

- Cliente: Contém tudo para entender a utilização real e a exploração do sistema de software a ser produzido;
- Solução: Contém tudo para fazer a especificação e o desenvolvimento do sistema de software e
- Esforço: Contém tudo para entender a equipe, e de qual maneira as pessoas executam seu trabalho.

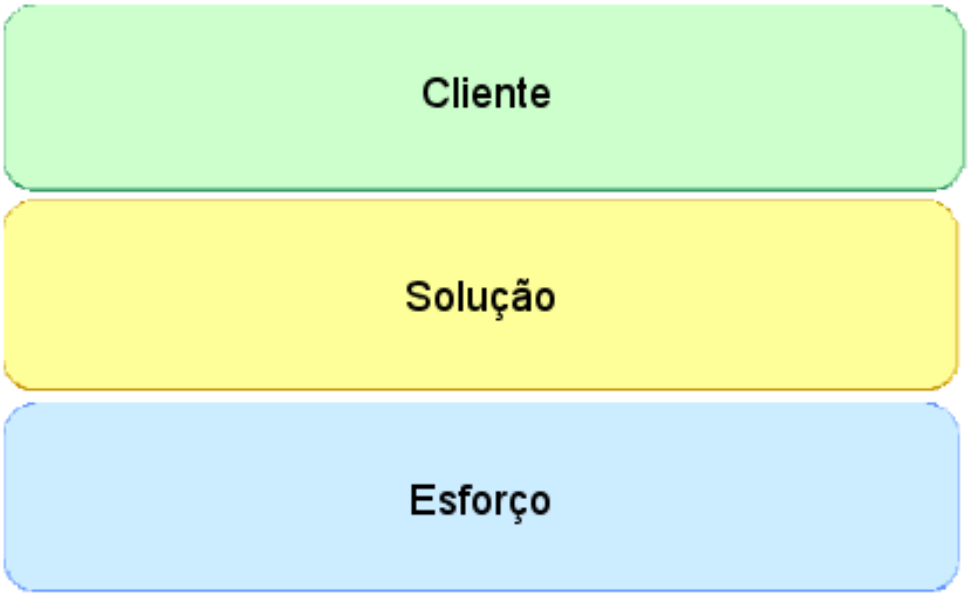

Figura 9: As três áreas de interesse (traduzido de SEMAT, 2012a)

2.2.3.1 Alfas

De acordo com SEMAT (2012a), os alfas do *kernel* (como visto na Figura 10):

- Capturam os conceitos chave envolvidos na engenharia de software;
- Permitem avaliar e monitorar o progresso e a saúde de qualquer esforço de engenharia de software e
- Fornecem a base comum para a definição de métodos e práticas de engenharia de software.

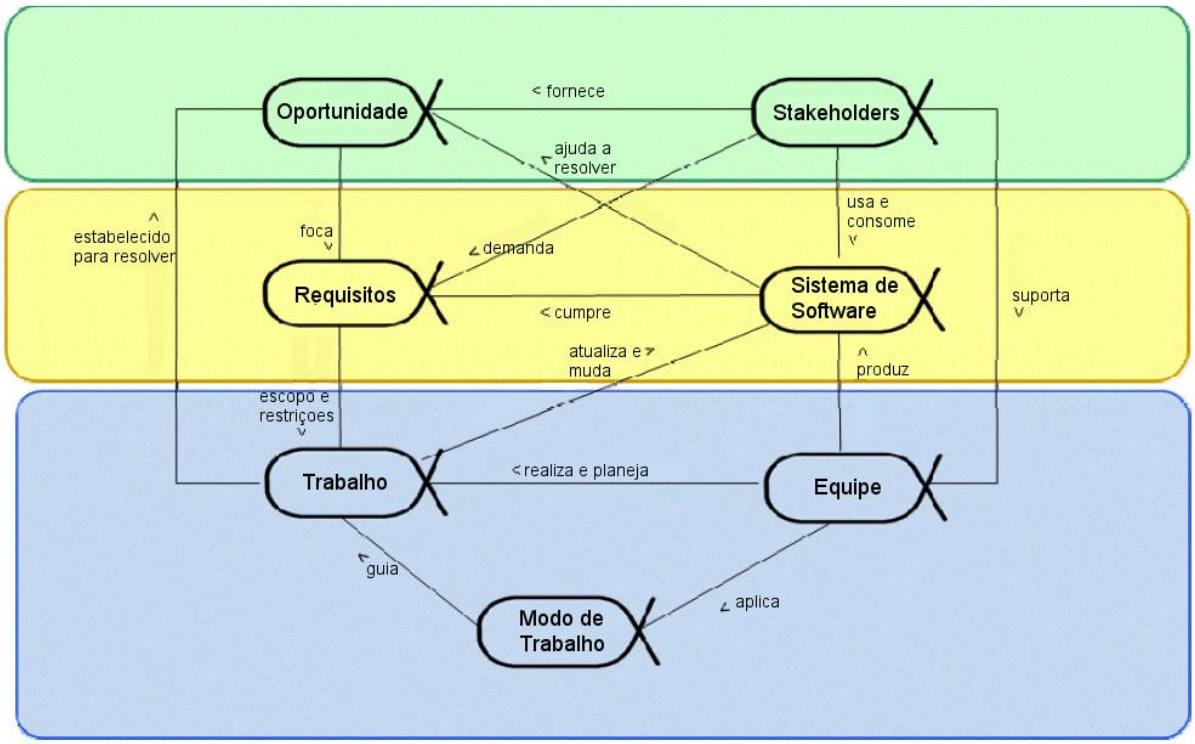

Figura 10: Alfas do *kernel* (traduzido de SEMAT, 2012a)

Na área de interesse do cliente, a equipe precisa entender os *stakeholders* e a oportunidade:

- Oportunidade: O conjunto de circunstâncias que torna apropriado desenvolver ou alterar um sistema de software e
- *Stakeholders*: As pessoas, grupos ou organizações que afetam ou são afetadas por um sistema de software.

Na área de interesse da solução, a equipe precisa estabelecer um entendimento compartilhado dos requisitos, e implementar, construir, testar, implantar e dar suporte ao sistema de software:

- Requisitos: O que o sistema de software deve fazer para atender a oportunidade e satisfazer os *stakeholders*. É importante descobrir o que é necessário do sistema de software e compartilhar esse entendimento entre os *stakeholders* e membros da equipe de forma a direcionar o desenvolvimento e testes do novo sistema e

- Sistema de software: Um sistema composto por software, hardware e dados que forneçam seu valor principal através da execução do software.

Na área de interesse do esforço, a equipe e seu modo de trabalho devem ser formados, e o trabalho deve ser feito:

- Trabalho: Atividade envolvendo esforço mental ou físico feito para atingir um resultado;
- Equipe: Grupo de pessoas engajadas ativamente no desenvolvimento, manutenção, distribuição e suporte de um sistema de software específico e
- Modo de trabalho: O conjunto adaptado de práticas e ferramentas utilizadas por uma equipe para orientar e apoiar o seu trabalho.

Cada alfa tem uma quantidade de estados, sendo que eles ajudam a compreender o estado atual do esforço de desenvolvimento e a determinar os próximos passos, como pode ser visto na Figura 11. Para cada estado existe um conjunto de critérios que indicam concretamente o que deve ser alcançado nele (JACOBSON et al., 2012).

Como sugerido por Ng e Magee (2010), o SEMAT adota cartões de estado para entender o estado de desenvolvimento da aplicação e para definir o ciclo de vida do projeto. Todos os cartões possuem a mesma estrutura, como pode ser visto na Figura 12. No topo aparece o nome do alfa do *kernel*. Logo abaixo aparece o nome do estado do alfa e os critérios para atingir aquele estado. No fim do cartão aparece o número de estado dentro do conjunto total de estados. O número de estado ajuda os usuários a se lembrar da ordem dos estados.

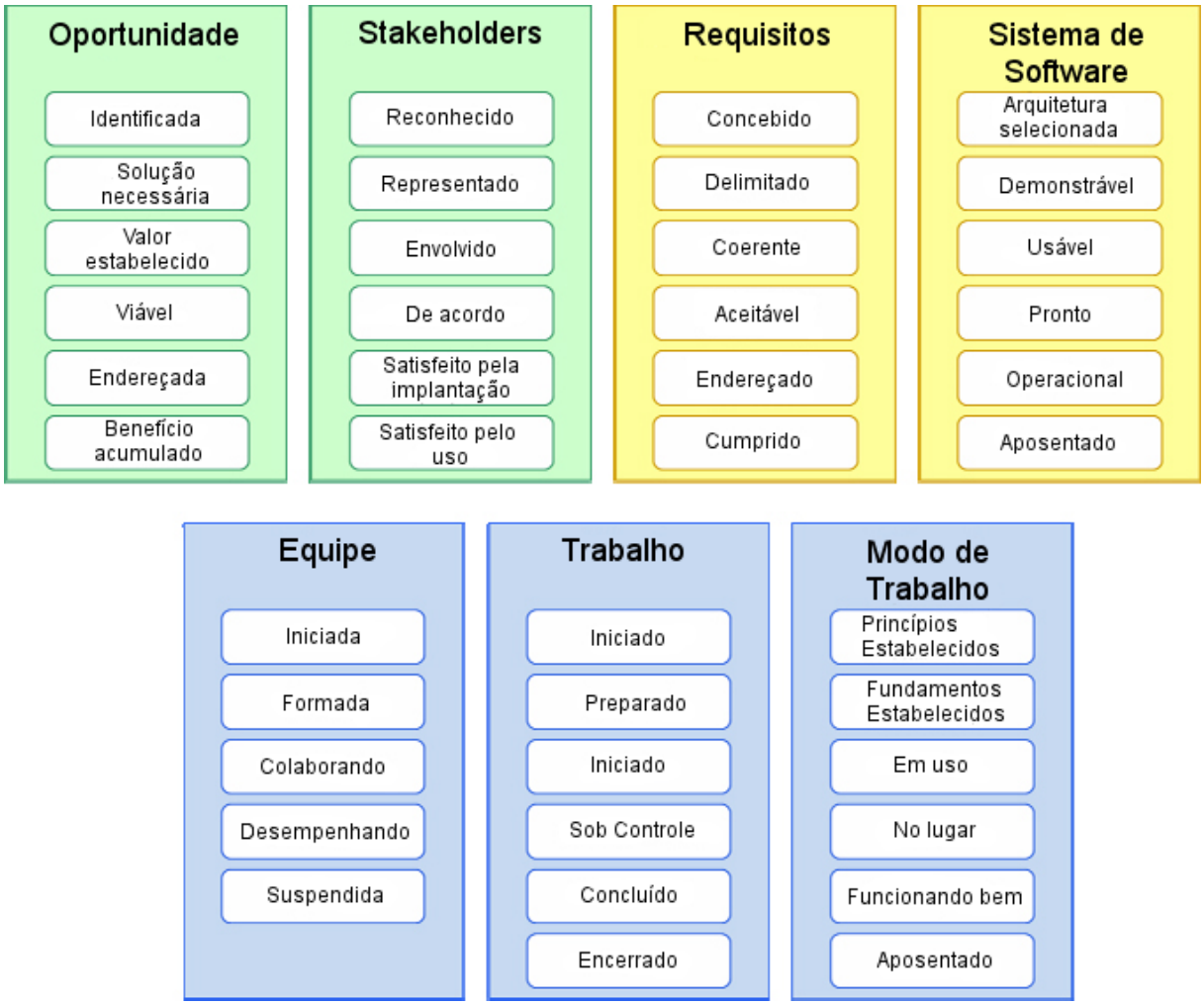

Figura 11: Estados dos alfas do *kernel* (traduzido de Jacobson et al., 2012)

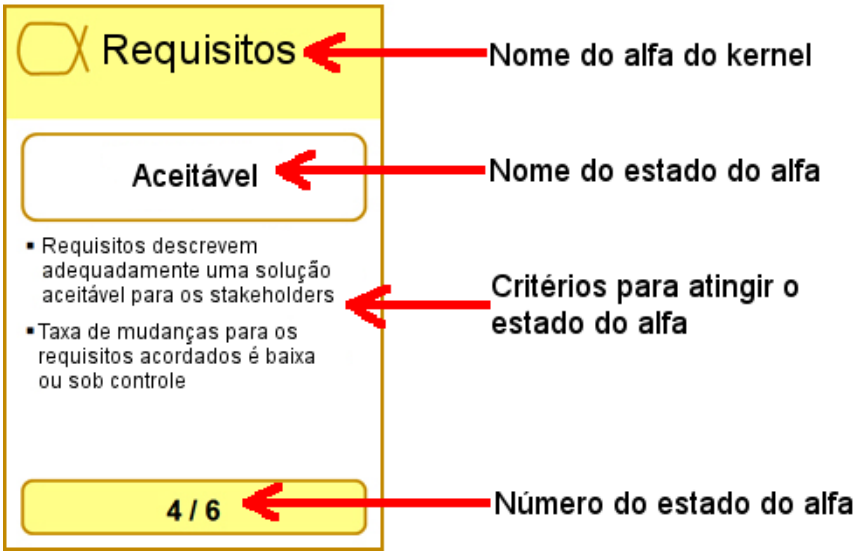

Figura 12: Anatomia do cartão (traduzido de Jacobson et al., 2012)

O desenvolvimento de software usando o *kernel* do SEMAT é feito em iterações. Segundo Jacobson et al. (2012), uma iteração é um espaço de tempo destinado a produzir um incremento estável de um sistema de software. A duração de uma iteração é tipicamente de 2 a 4 semanas e pode envolver todos os tipos de atividades, desde a elicitação de requisitos até a implantação do sistema de software resultante. Além de produzir o próximo incremento estável de software, devem ser definidos objetivos adicionais para a iteração. Esses objetivos podem se relacionar com o progresso do estado de um ou mais alfas do *kernel*. Quando a equipe concorda no que deve ser feito em uma iteração, os objetivos são adicionados no *backlog* da iteração. O *backlog* pode ser refinado adicionando as tarefas necessárias para atingir aos objetivos. Uma tarefa é uma porção de trabalho que pode claramente ser identificada, isolada e aceita por um ou mais membros da equipe para ser executada. Cada objetivo da iteração pode ser quebrado em uma ou mais tarefas a ser gerenciada no *backlog* da iteração.

Para desenvolver software, a equipe está continuamente planejando, fazendo, controlando e adaptando, o que é conhecido como ciclo PDCA:

- *Plan*: A equipe determina o estado atual de todo o esforço determinando o estado atual de cada um dos alfas. Baseado nisso, define-se quais alfas e estados serão desenvolvidos na próxima iteração. Então, é planejado como atingir esses estados identificando tarefas a serem completadas;
- *Do*: Em cada iteração, a equipe trabalha nas tarefas identificadas para atingir os estados alvo. Isso envolve escrever código, testar, configurar os ambientes, discutir requisitos, escrever documentação, etc.;
- *Check*: A equipe verifica os objetivos e as tarefas para ter certeza que foram completadas do modo que foram planejados e se ela seguiu o modo de trabalho combinado e
- *Adapt*: A equipe revisa seu modo de trabalho, identifica impedimentos e encontra melhores formas de fazer as coisas. Isso resulta em mudanças nos planos e no modo de trabalho da equipe.

Antes de iniciar um esforço de engenharia de software, é importante identificar a visão do *kernel* de cada um dos membros da equipe. Em cada iteração, a equipe sempre:

- Entende melhor os requisitos;
- Produz um novo incremento do sistema de software e
- Aplica e melhora seu modo de trabalho.

Pode-se pensar no planejamento de uma iteração da seguinte forma:

- Determinar onde está: Entender qual o estado atual do esforço;
- Determinar aonde ir: Decidir o que enfatizar em seguida e quais os objetivos da próxima iteração e
- Determinar como chegar lá: Concordar com as tarefas que devem ser feitas para atingir os objetivos.

Quando está sendo planejada uma iteração os alfas podem ajudar a descobrir onde se está e para onde ir em seguida. A relação entre os alfas, os objetivos e as tarefas no *backlog* da iteração está mostrada na Figura 13.

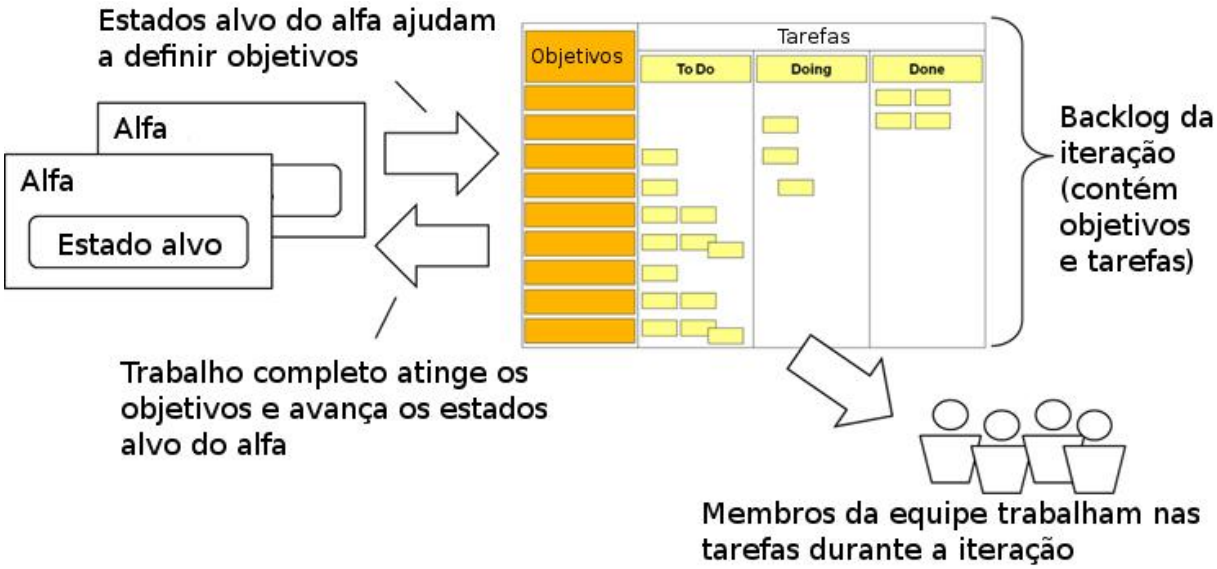

Figura 13: *Backlog* de iteração no SEMAT (traduzido de Jacobson et al., 2012)

Há duas formas de determinar em que ponto está o desenvolvimento do esforço de engenharia de software com o uso dos cartões de estado do alfa:

- *Walkthrough*: Uma abordagem simples usando um conjunto de cartões.
    - Devem-se colocar os cartões de cada alfa em linha sobre uma mesa com o primeiro estado a esquerda e o último estado à direita;
    - Seguir passo a passo cada estado e perguntar para a equipe se foi atingido aquele estado;
    - Se o estado foi atingido, deve-se mover aquele cartão de estado para a esquerda. Deve-se continuar com o próximo cartão de estado até que chegue a um estado que a equipe não tenha atingido e
    - Deve-se mover esse cartão de estado e o restante dos cartões de estado pendentes para a direita.
- *Poker*: Outra abordagem que às vezes funciona melhor é o *poker*.
    - Para cada membro da equipe é dado um conjunto de cartões de estado;
    - Para cada alfa, cada membro seleciona o cartão de estado que ele acha que melhor representa o estado atual do esforço de desenvolvimento de software;
    - Todos os membros colocam o cartão de estado selecionado com a face virada para baixo na mesa;
    - Quando todos colocaram os cartões na mesa, deve-se virar o cartão com a face para cima;
    - Se todos os membros selecionaram o mesmo cartão de estado, então há consenso e
    - Se os cartões de estado selecionados são diferentes há interpretações diferentes dos *checklists* dos estados. A equipe deve discutir os *checklists* para o estado para entrar em um acordo.

Uma vez que a equipe determinou quais os estados atuais dos alfas, é fácil selecionar quais serão os estados a serem atingidos na próxima iteração. Há critérios bem definidos para a conclusão dos estados alvos através de suas *checklists*. Selecionar os estados alvos pode ser facilmente feito como uma extensão das técnicas de *walkthrough* e *poker*.

Tendo identificado um conjunto candidato de objetivos para a iteração, a equipe deve decidir como irá alcançá-los e se conseguirão atingi-los durante a iteração. Tipicamente isso é feito identificando uma ou mais tarefas a ser completada para atingir o objetivo. Os estados do alfa ajudam a equipe com uma *checklist* para cada estado, fornecendo dicas de quais tarefas devem ser feitas para atingir o objetivo.

2.2.3.2 Espaços de atividades

O *kernel* ainda fornece um conjunto de espaços de atividade que complementam os alfas ao prover uma visão baseada em atividades de engenharia de software, como pode ser visto na Figura 14.

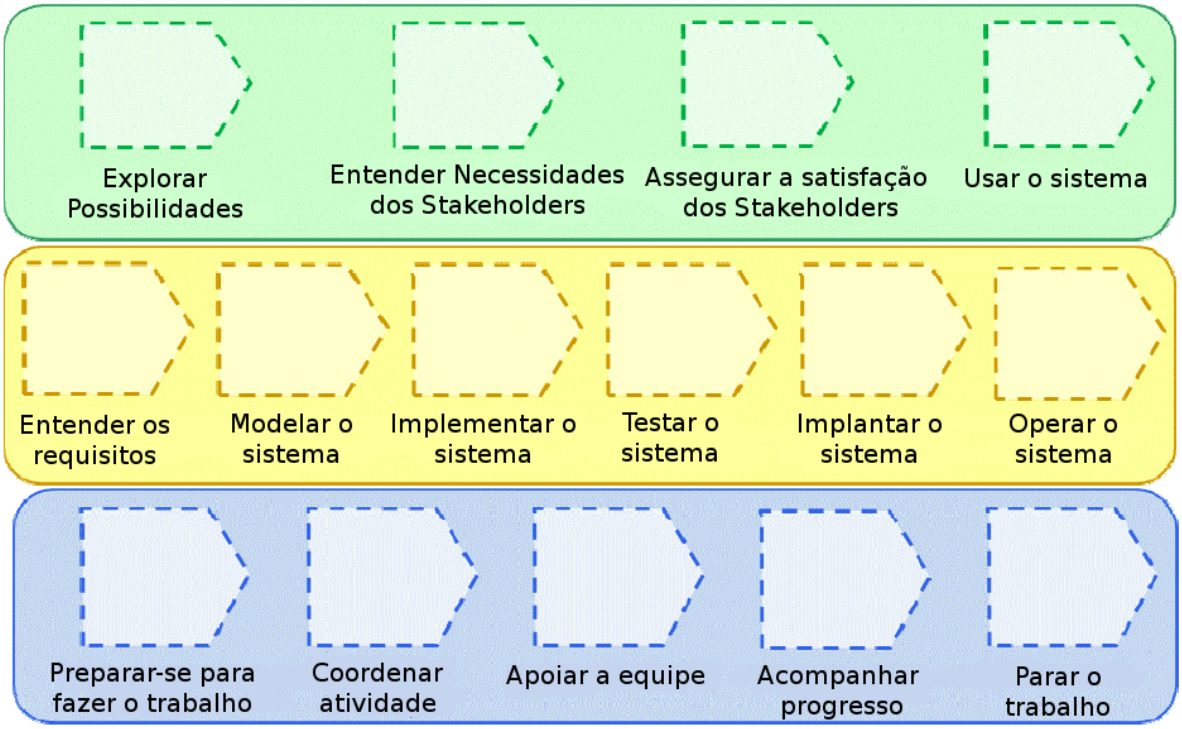

Figura 14: Espaços de atividade do *kernel* (traduzido de SEMAT, 2012a)

Na área de interesse do cliente, a equipe precisa entender a oportunidade, e dar apoio e envolver os *stakeholders*:

- Explorar as possibilidades: Devem-se explorar as possibilidades apresentadas pela criação de um sistema de software novo ou melhorado. Para isso, é importante analisar a oportunidade e identificar os *stakeholders*;
- Entender as necessidades dos *stakeholders*: É preciso envolver as partes interessadas para entender suas necessidades e garantir que os resultados corretos sejam produzidos. Isto inclui identificar e trabalhar lado a lado com os representantes dos *stakeholders* para entender a oportunidade;
- Assegurar a satisfação dos *stakeholders*: Compartilhar os resultados do trabalho de desenvolvimento com os *stakeholders* para ganhar sua aceitação do sistema produzido e verificar se a oportunidade foi atendida com sucesso e
- Usar o sistema: Usar o sistema em um ambiente vivo para beneficiar os *stakeholders*.

Na área de interesse da solução, a equipe deve desenvolver uma solução apropriada para explorar a oportunidade e satisfazer os *stakeholders*:

- Entender os requisitos: Estabelecer um entendimento compartilhado do que o sistema de software a ser produzido deve fazer;
- Modelar o sistema: Modelar o sistema de software de modo que seja fácil desenvolvê-lo, alterar e fazer manutenção, e possa lidar com as atuais e futuras demandas. Isto inclui a concepção global da arquitetura do sistema a ser produzido;
- Implementar o sistema: Construir um sistema de software através da implementação, testes e integração de um ou mais elementos do sistema de software. Isto inclui a correção de *bugs* e testes unitários;
- Testar o sistema: Verificar se o sistema de software produzido atende aos requisitos dos *stakeholders*;

- Implantar o sistema: Assim que o sistema de software estiver testado, deve-se torná-lo disponível para uso por pessoas fora da equipe de desenvolvimento e
- Operar o sistema: Apoiar o uso do sistema de software no ambiente de produção.

Na área de interesse do esforço, a equipe deve ser formada e executar o trabalho de acordo com o modo de trabalho:

- Preparar-se para fazer o trabalho: Montar a equipe e seu ambiente de trabalho. Compreender e comprometer-se com a conclusão do trabalho;
- Coordenar atividade: Coordenar e dirigir o trabalho da equipe. Isto inclui todo o planejamento em curso e replanejamento do trabalho, e adicionar os recursos necessários para completar a formação da equipe;
- Apoiar a equipe: Ajudar os membros da equipe a se ajudarem mutuamente, colaborar e melhorar o seu modo de trabalho;
- Acompanhar o progresso: Medir e avaliar os progressos realizados pela equipe e
- Parar o trabalho: Encerrar o esforço de engenharia de software e entregar as responsabilidades da equipe.

2.3 Engenharia de requisitos

Uma das etapas típicas de qualquer processo de desenvolvimento de software é a da engenharia de requisitos. É nessa etapa que o engenheiro tenta entender quais são as reais necessidades dos usuários, ou seja, o problema que se deve resolver, solucionar ou dissolver. Também é nessa etapa em que se começa a identificar as dimensões humanas e sociais relacionadas ao software a ser desenvolvido.

Kotonya e Sommerville (1998) argumentam que a engenharia de requisitos engloba todas as atividades relacionadas à descoberta, documentação e manutenção do conjunto de requisitos de um sistema.

Segundo Simonette (2010), a interação com as pessoas para identificar desejos e o problema a ser atendido pelo sistema permite identificar o processo de engenharia de requisitos como um sistema dentro do sistema a ser desenvolvido. Logo, a engenharia de requisitos é um sistema onde seus componentes são atividades humanas.

Segundo Maciaszek (2001), um requisito é uma condição ou capacidade que deve ser alcançada ou possuída por um sistema ou componente deste para satisfazer um contrato, padrão, especificação ou outros documentos formalmente impostos.

A norma IEEE Std 610.12-1990 (IEEE, 1990) define requisitos como:

> (1) Uma condição ou capacidade necessária para um usuário solucionar um problema ou atingir um objetivo;
>
> (2) Uma condição ou capacidade que precise ser atendida ou estar presente em um software, ou componente do software, para satisfazer um contrato, norma, especificação ou outro documento imposto formalmente e
>
> (3) Uma documentação que represente uma condição ou capacidade como em (1) ou (2).

Em Sommerville (2007), os requisitos de um software são classificados em funcionais, não funcionais e de domínio. Os requisitos funcionais são aqueles que descrevem o que o software deve fazer, quais serviços devem fornecer ao usuário e como deve reagir a entradas específicas.

Já os requisitos não funcionais não estão ligados diretamente às funcionalidades do software; estão ligadas às restrições sobre serviços e funções oferecidas por ele. Os requisitos não funcionais podem ser subdivididos ainda em três:

- <u>Requisitos de produto</u>: Especificam o comportamento do produto. Como exemplo desse tipo, têm-se os requisitos de desempenho, os requisitos de confiabilidade, os requisitos de portabilidade e os requisitos de usabilidade;

- Requisitos organizacionais: São derivados de políticas e procedimentos da organização do cliente e do desenvolvedor. Exemplos incluem padrões de processo que devem ser usados, requisitos de implementação, como a linguagem de programação ou método de projeto usado, e requisitos de entrega que especificam quando o produto e a sua documentação devem ser entregues e
- Requisitos externos: Abrange todos os requisitos derivados de fatores externos ao software e seu processo de desenvolvimento. Por exemplo, os requisitos de interoperabilidade, os requisitos legais e os requisitos éticos.

Os requisitos de domínio são provenientes do domínio de aplicação do software e usam terminologia específica de domínio. Podem ser novos requisitos funcionais, podem restringir os requisitos funcionais existentes ou estabelecer como cálculos específicos devem ser realizados. Os engenheiros de software frequentemente encontram dificuldade em compreender como esses requisitos estão relacionados a outros do software pelo fato de serem requisitos especializados. Porém, eles são importantes por refletirem os fundamentos do domínio da aplicação. Sendo assim, se esses requisitos não forem satisfeitos, pode ser difícil fazer o software funcionar satisfatoriamente.

Outra classificação interessante para requisitos é apresentada em Bolt e Mazur (1999), o modelo de Kano, no qual se considera que existem três tipos de requisitos dos clientes:

- Requisitos normais: Tipicamente aqueles que podem ser identificados apenas perguntando aos clientes o que eles desejam. Caso não sejam contemplados, levarão a insatisfação. Já se forem, levam a satisfação do cliente;
- Requisitos esperados: São frequentemente tão básicos que os clientes se esquecem de mencioná-los. Só irão ser mencionados caso os projetistas falhem em executá-los. Caso não sejam contemplados, levam a uma grande insatisfação. Já se forem, não mudam a satisfação do cliente e

- Requisitos excitantes: São difíceis de descobrir. Estão acima das expectativas dos clientes. Se forem contemplados levam a uma grande satisfação do cliente. Caso não sejam, não trazem insatisfação aos clientes.

2.3.1 Estágios da engenharia de requisitos

Para Pressman (2006), a engenharia de requisitos possui sete estágios: concepção, levantamento, elaboração, negociação, especificação, validação e gestão. A maioria dos esforços de engenharia de software é iniciada a partir de uma necessidade de negócio identificada ou de um serviço novo. A concepção tem como objetivo estabelecer um entendimento básico do problema e normalmente é feito de forma informal.

Em seguida, passa-se ao estágio de levantamento de requisitos. A ideia é bem fácil de entender: pergunte ao cliente, usuários e demais interessados no software os objetivos do software. Apesar de parecer uma tarefa simples, na verdade, não é. Há três problemas principais que podem aparecer nesse processo: problemas de escopo, quando os limites do software não são bem definidos; problemas de entendimento, quando os clientes não estão completamente certos do que é necessário; e por fim, problemas de volatilidade, quando os requisitos mudam ao longo do tempo.

Após o levantamento, há a elaboração, na qual as informações obtidas são refinadas e é desenvolvido um modelo técnico das funções, características e restrições do software.

Muitas vezes, é necessário um estágio de negociação para conciliar conflitos entre pedidos de diferentes clientes e usuários. Nessa etapa, também se avalia o risco de cada requisito e faz-se uma priorização deles.

A especificação é o produto final do trabalho do engenheiro de requisitos, servindo como base para as atividades das etapas seguintes de um modelo de engenharia de software. Pode ser um documento escrito, um modelo gráfico, um modelo matemático formal, uma coleção de cenários de uso, um protótipo ou qualquer

combinação desses elementos. O importante é que descreva a função e o desempenho de um software e as restrições que devem ser levadas em conta durante seu desenvolvimento.

O resultado da fase de especificação é avaliado no passo de validação. É feito um exame detalhado da especificação para garantir que todos os requisitos de software foram declarados de modo não ambíguo e que as inconsistências, os erros e as omissões tenham sido detectados e corrigidos. Para isso, é feita uma revisão técnica por uma equipe composta por engenheiros de software, clientes, usuários e outros interessados no software.

A gestão de requisitos é um conjunto de atividades que ajudam a identificar, controlar e rastrear requisitos e modificações de requisitos em qualquer passo do desenvolvimento do software. Para isso, cada requisito recebe um identificador. Uma vez identificados, são criadas tabelas que relacionam os requisitos a um ou mais aspectos do software ou de seu ambiente. Essas tabelas são conhecidas como tabelas de rastreamento. Um exemplo é a tabela de rastreamento de dependência, que indica como os requisitos estão relacionados uns com os outros.

Já Sommerville (2007) resume tudo em apenas três estágios, no quais as atividades estão organizadas como um processo iterativo em espiral. São eles: estudo de viabilidade, elicitação e análise de requisitos e validação de requisitos.

O estudo de viabilidade é um estudo breve e focalizado que deve investigar se o software contribui para os objetivos gerais da organização, se ele pode ser implementado com tecnologia atual e dentro das restrições definidas de custo e prazo e se pode ser integrado a outros sistemas implantados. Esse estudo é feito a partir de um conjunto preliminar de requisitos de negócios, um esboço da descrição do sistema e como o sistema pretende apoiar os processos de negócios. O resultado é um relatório que recomenda ou não prosseguir o desenvolvimento do software e também pode propor mudanças de escopo, orçamento e prazo e sugerir requisitos de alto nível adicionais para o software.

O estágio seguinte é a elicitação e análise de requisitos. Ele pode ser subdividido em quatro atividades: obtenção dos requisitos, que é a interação com os interessados no software para coletar seus requisitos; classificação e organização de requisitos,

que envolve a coleção de requisitos não estruturados, agrupam os que são relacionados e os organiza em conjuntos coerentes; priorização e negociação de requisitos, que está relacionada à resolução de conflitos entre os desejos dos diversos interessados no software; e documentação de requisitos.

O último estágio é a validação de requisitos, que funciona de forma semelhante ao explicado anteriormente na proposta de Pressman (2006). Sommerville (2007) também cita o gerenciamento de requisitos como um estágio adicional que ocorre durante todo o desenvolvimento de software e até mesmo depois de sua implantação na organização.

2.3.2 Elicitação de requisitos em modelos tradicionais

Nesse trabalho, entende-se por modelos tradicionais aqueles que possuem um maior formalismo em seu processo. Os modelos tradicionais contemplam o cascata, o espiral e o processo unificado explicados anteriormente. Para esses modelos, a elicitação de requisitos pode ser feita como explicado em Kouri (2007):

- *Joint Application Development* (JAD): Promove a cooperação, entendimento e formação de equipes de trabalho entre os envolvidos no universo de informações;
- *Participatory Design* (PD): Concentra-se mais fortemente no envolvimento dos usuários, se comparada ao JAD, por facilitar o processo de aprendizado entre desenvolvedores e usuários através de experiências conjuntas em situações de trabalho simuladas. Em linhas gerais, os usuários são introduzidos no ambiente dos desenvolvedores, conhecendo possibilidades técnicas e, da mesma maneira, os desenvolvedores colaboram com os usuários em suas tarefas. Ocorre um aprendizado que contribui no processo de definição dos requisitos;
- Cenários: É uma descrição parcial do comportamento da aplicação, que ocorre em um dado momento, em uma situação específica;

- Casos de Uso: É um documento narrativo que descreve uma sequência de transações realizadas por um ator. Um caso de uso é uma técnica de modelagem usada para descrever o que um novo sistema deve fazer, sem se preocupar em como ele vai fazer. Cada caso de uso define um requisito funcional do sistema;
- Pontos de Vista: É um encapsulamento de uma informação parcial sobre os requisitos de um sistema. Informações de diferentes pontos de vista devem ser integradas para se gerar a especificação final do sistema e
- Volere: É um método baseado na utilização de casos de uso. Através de um modelo padrão de especificação de requisitos, define passo a passo como os requisitos do sistema devem ser levantados, analisados, validados e documentados para que se obtenha, no final, um documento de especificação de requisitos do sistema a ser desenvolvido.

Em Sommerville (2007) é apresentada também outra forma que é através de entrevista com os interessados no software. As entrevistas podem ser fechadas, seguindo perguntas predefinidas pela equipe de engenharia de requisitos, ou abertas, nas quais não existe um roteiro a ser seguido.

2.3.3 Elicitação de requisitos em modelos ágeis

Os modelos ágeis de engenharia de software não seguem os estágios convencionais da engenharia de requisitos propostos em Sommerville (2007) e Pressman (2006). Nesse item serão discutidas algumas práticas que colaboram na elicitação dos requisitos nesses modelos.

2.3.3.1 Crystal

Na família de modelos Crystal, há duas práticas que colaboram para a elicitação de requisitos.

A primeira delas é o planejamento a jato, que é baseada no jogo do planejamento usado na programação extrema (Bassi, 2008). No início do projeto é feita uma reunião em que desenvolvedores, patrocinadores, usuários representantes e analistas de negócios fazem um plano de curto prazo (períodos de até três meses). Todos os participantes discutem e escrevem cartões de história. Posteriormente, esses cartões são organizados por dependência.

A equipe de desenvolvimento anota no cartão as estimativas de duração, o nome de quem se compromete a executar a tarefa e quais são as dependências externas ao projeto existentes para sua execução (se houver).

Para ajudar os usuários e a equipe de negócios a alinhar as expectativas com a realidade do desenvolvimento, o esqueleto que anda, a primeira versão e a primeira fonte de receita são planejados, fazendo-se um esforço para visualizar como esses marcos se posicionam em meio às tarefas técnicas. Além da primeira versão, outras podem ser definidas simplesmente agrupando os cartões.

Para finalizar o processo, o resultado do planejamento é fotografado para virar um radiador de informações, ou os cartões são numerados e depois colados em uma parede de forma ordenada.

A outra prática que colabora na elicitação de requisitos é a descoberta das interações do usuário. Essa prática é uma adaptação para o contexto ágil do design centrado no uso. Uma reunião é feita com um grupo de 8 a 12 participantes que representam os principais envolvidos e grupos de usuários com interesses distintos. O ambiente do encontro deve ser confortável e ter a disposição quadros brancos, cartões em branco, canetas coloridas, papel para rascunho e guloseimas. Durante a reunião, são criados diagramas e cartazes que servem de base para a criação da interface e de guia para a compreensão dos objetivos e prioridades do projeto.

Os tipos de perfis de usuários, seus principais objetivos, a frequência de acesso ao sistema e a importância do ponto de vista dos analistas de negócios são colocados em cartões. Depois o modelo de papéis de usuários é criado através do agrupamento dos cartões de acordo com os objetivos, eles são colados em um quadro branco e linhas são desenhadas, unindo os grupos relacionados. Os tipos de usuários determinantes para o sucesso do software são escolhidos através de votação pelos mais experientes no negócio.

De cada um dos cartões são extraídas as operações que os usuários precisam realizar para alcançar seus objetivos e escrevem-se cartões de operações. Cria-se o modelo de tarefas através do agrupamento de cartões similares, unidos através de linhas que mostram os relacionamentos e a ordem em que as operações acontecem. Por votação novamente, as tarefas que possuem maior valor de negócio ou que são realizadas mais vezes são escolhidas.

A partir do modelo de tarefas, os lugares no software nos quais essas operações se encontram são identificados e, ligando-os, é obtido um mapa de navegação do software.

Em outro quadro, dois eixos são desenhados: a importância para o negócio de uma operação versus tempo (momento em que o usuário irá realizar cada operação). Colocam-se os cartões nesse eixo de forma que os do topo sejam os de valor de negócio mais altos, geralmente formando um dos fluxos de operações que o usuário fará; esta fileira é uma possível primeira versão.

Os resultados desta reunião são a identificação das principais funcionalidades e prioridades do software e os vários radiadores de informação que serão colocados no ambiente de desenvolvimento.

2.3.3.2 Scrum

O scrum é um modelo gerencial e organizacional. Ele diz como deve ser o processo, mas não sobre como atingir ou executar. O que deve ser feito durante o sprint para implementar o software não está no escopo do modelo.

No scrum, o que pode ser considerado mais próximo ao levantamento de requisitos é a construção do backlog. Porém, a abordagem do scrum é bem mais leve do que a dos modelos tradicionais, sendo que os requisitos contidos no backlog possuem um nível de detalhes muito menor. Um segundo momento em que novos requisitos são identificados é durante a reunião de planejamento com o dono do produto. Nessa reunião, as histórias mais prioritárias ganham maior detalhamento.

2.3.3.3 Programação extrema

Segundo Sommerville (2007), na programação extrema, os requisitos de software são expressos como cenários, chamados de histórias do usuário, que são implementados diretamente como uma série de tarefas.

A cada iteração de desenvolvimento de software, é necessário ser feito um planejamento de quais histórias serão desenvolvidas. Para isso, é feita uma reunião chamada jogo do planejamento com a participação dos desenvolvedores e dos clientes.

O planejamento nesse modelo é feito com histórias escritas em pequenos cartões. Cada cartão é escrito pelo cliente e deve descrever uma unidade de funcionalidade, que geralmente representa um requisito funcional desejado (Sato, 2007).

Após os cartões serem escritos, os desenvolvedores os dividem em tarefas (caso seja necessário) e estimam o esforço e os recursos necessários para implementação de cada um deles. O cliente prioriza as funcionalidades para implementação, escolhendo as que mais agreguem valor ao negócio imediatamente. Sabendo a prioridade e o esforço necessário para cada história, os envolvidos negociam um subconjunto de funcionalidades, cuja prioridade seja mais alta e cuja complexidade seja possível de lidar no período de desenvolvimento.

A programação extrema já prevê que os requisitos podem mudar ao longo do desenvolvimento e, quando isso acontece, as funcionalidades que não foram implementadas mudam ou são descartadas. Se forem necessárias mudanças em um sistema que já foi entregue, novos cartões de histórias são escritos e,

novamente, o cliente decide se essas mudanças devem ter prioridade sobre a nova funcionalidade (Sommerville, 2007).

2.3.4 Elicitação de requisitos no SEMAT

No modelo SEMAT, os requisitos são um alfa e possuem seis estados (SEMAT, 2012a):

- Concebido (*Conceived*): A necessidade de um novo sistema foi acordada;
- Delimitado (*Bounded*): O propósito e o tema do novo sistema estão claros;
- Coerente (*Coherent*): Os requisitos fornecem uma descrição coerente das características essenciais do novo sistema;
- Aceitável (*Acceptable*): Os requisitos descrevem um sistema que seja aceitável para os *stakeholders*;
- Endereçado (*Addressed*): Os requisitos que tenham sido endereçados satisfazem a necessidade de um novo sistema de uma forma que seja aceitável para os *stakeholders* e
- Cumprido (*Fulfilled*): Os requisitos que tenham sido endereçados satisfazem totalmente a necessidade de um novo sistema.

Na Figura 15 são mostrados os cartões de estado do alfa requisitos.

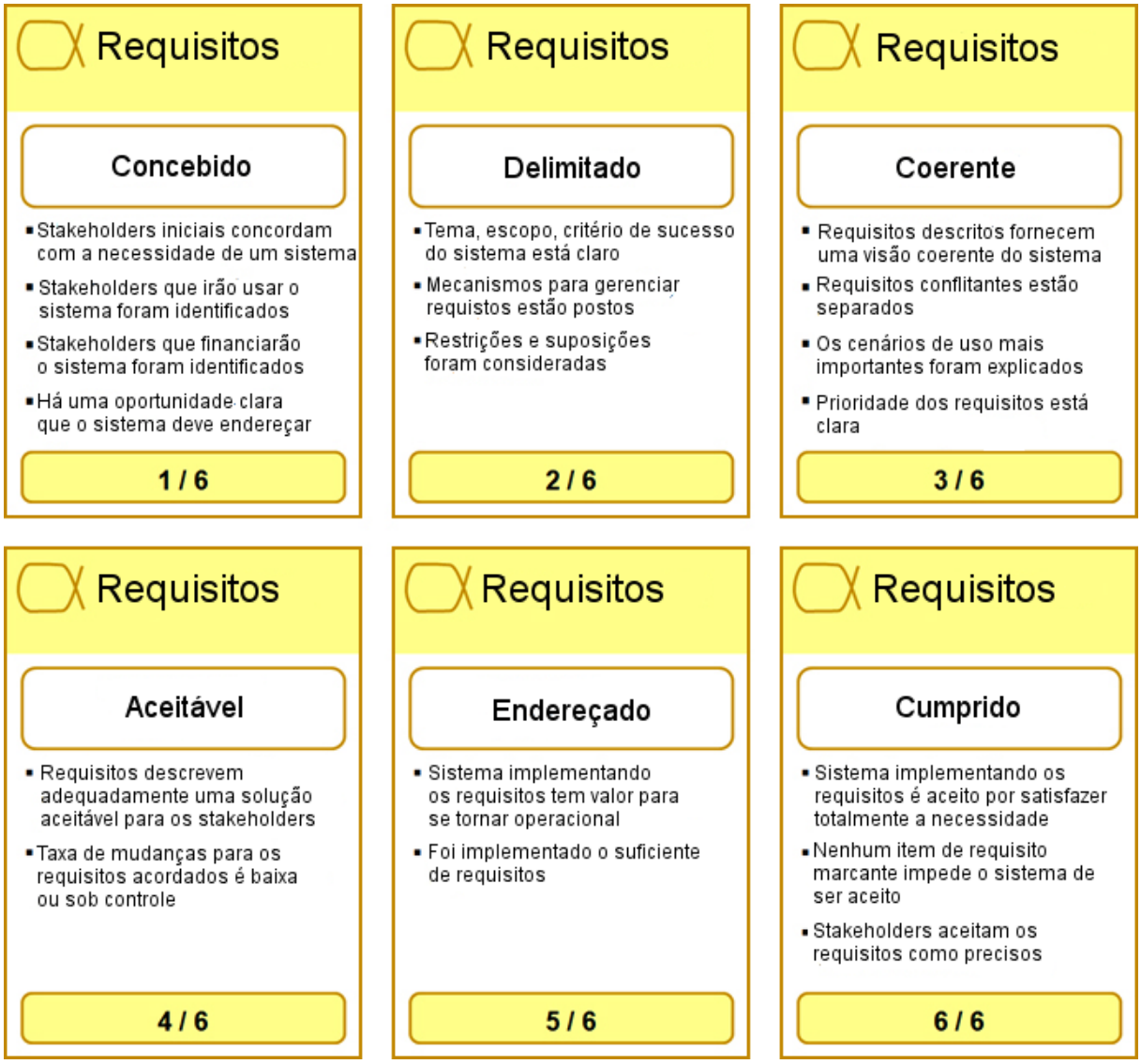

Figura 15: Cartões de estado do alfa requisitos (traduzido de Jacobson et al., 2012)

Cada alfa do SEMAT possui alguns estados e, para verificar em qual estado do alfa um esforço de engenharia de software está, é usada uma *checklist*. No caso dos requisitos, é usada a *checklist* mostrada na Tabela 1.

Tabela 1: *Checklist* do alfa requisitos

Estado	**Checklist**
Concebido	O conjunto inicial de *stakeholders* concorda que o sistema deve ser produzido. Os *stakeholders* que irão usar o novo sistema estão identificados. Os *stakeholders* que irão financiar o trabalho inicial do novo sistema estão identificados. Há uma oportunidade clara para o novo sistema.
Delimitado	Os *stakeholders* envolvidos no desenvolvimento do novo sistema estão identificados. Os *stakeholders* concordam com o propósito do novo sistema. Está claro o que significa sucesso para o novo sistema. Os *stakeholders* têm um entendimento compartilhado da extensão da solução proposta. A forma como os requisitos serão descritos está acordada. Os mecanismos para gerenciar os requisitos estão claros. O esquema de priorização está claro. As restrições estão identificadas e foram consideradas. As premissas estão claramente definidas.
Coerente	Os requisitos foram capturados e compartilhados com a equipe e os *stakeholders*. A origem dos requisitos é clara. A lógica por trás dos requisitos é clara. Requisitos conflitantes estão identificados e atendidos. Os requisitos comunicam as características essenciais do sistema a ser entregue. Os cenários de uso mais importantes para o sistema podem ser explicados. A prioridade dos requisitos estão claras.

Coerente (cont.)	O impacto da implementação dos requisitos é compreendido. A equipe entende o que tem que ser entregue e se compromete a entregá-lo.
Aceitável	Os *stakeholders* aceitam que os requisitos descrevem uma solução aceitável. A taxa de mudanças dos requisitos acordados é relativamente baixa e sob controle. O valor fornecido através da implementação dos requisitos é claro. As partes da oportunidade satisfeitas pelos requisitos são claras.
Endereçado	Foi endereçado o suficiente dos requisitos para que o sistema resultante seja aceitável para os stakeholders. Os *stakeholders* aceitam os requisitos como um reflexo preciso do que o sistema faz e não faz. O conjunto de itens de requisitos implementados fornecem valor claro para os *stakeholders*. O sistema implementando os requisitos é aceito pelos *stakeholders* como importante de se tornar operacional.
Cumprido	Os *stakeholders* acreditam que os requisitos capturam com precisão o que eles exigem para satisfazer plenamente a necessidade de um novo sistema. Não há itens de requisitos pendentes que impedem o sistema de ser aceito de forma a satisfazer plenamente os requisitos. O sistema é aceito pelos *stakeholders* como satisfazendo plenamente os requisitos.

3 SOFTWARE PARA *E-LEARNING* E ENSINO DE BANCO DE DADOS

Nesse capítulo serão apresentados os conceitos de *e-learning* e alguns exemplos de software para a área, além de apresentar os conceitos básicos de ensino de banco de dados.

3.1 Software para *e-learning*

Hoje em dia uma das expressões mais usadas quando se quer referir à aprendizagem por meio de diversas mídias eletrônicas é o *e-learning*. Segundo Tuparova e Tuparov (2005), há definições diferentes para essa expressão:

- O processo de aprendizagem formal e informal, de atividades de treinamento, de comunidades e de eventos, via o uso de todas as mídias eletrônicas como a Internet, Intranet, Extranet, CD-ROM, vídeo, TV, telefones celulares e agendas;
- O uso de novas tecnologias multimídia e da Internet para aumentar a qualidade da aprendizagem por facilitar o acesso aos recursos e serviços assim como as trocas remotas e de colaboração e
- Uma definição ampla do campo de uso e distribuição da tecnologia para a disponibilidade da aprendizagem e de programas de treinamento, usualmente utilizada para descrever mídias tais como: CD-ROM, Internet, Intranet, de aprendizagem sem fio e móvel. Alguns incluem a administração do conhecimento como uma forma de *e-learning*. Por volta do ano de 1995 os termos utilizados eram o de “treinamento baseado na Internet”, depois “treinamento baseado na web” (para esclarecer que a disponibilidade poderia somente ocorrer na Internet ou Intranet), mais adiante de “aprendizagem online”, e finalmente o termo *e-learning* foi adotado corretamente e empregado de forma popular.

Tuparova e Tuparov (2005) sintetizam que todas as definições incluem a combinação e a interação dos seguintes componentes básicos: atividades de ensino e aprendizagem por meio de diferentes mídias eletrônicas.

Horton (2006) propõe uma classificação para os diferentes tipos de software para *e-learning*:

- Cursos standalone: cursos realizados pelo aluno sozinho. Fazem no seu ritmo próprio, sem interação com um instrutor ou colegas;
- Cursos em sala de aula virtual: aula on-line estruturada de forma semelhante a um curso presencial. Pode ou não incluir reuniões online síncronas;
- Jogos de aprendizagem e simulações: aprendizagem através da realização de atividades simuladas que exigem exploração e levam a descobertas;
- *Embedded e-learning*: software para *e-learning* incluso em outro software, como um programa de computador, um procedimento de diagnóstico, ou a ajuda online;
- *Blended Learning*: uso de diferentes formas de aprendizagem para alcançar um objetivo único. Pode misturar aulas presenciais e software para *e-learning* ou várias formas de software para *e-learning*. Outra definição seria a convergência entre o virtual e o presencial na educação;
- *Mobile learning*: o aprendizado sobre o mundo pode ocorrer enquanto o aprendiz se move pelo mundo. Auxiliado por dispositivos móveis, como PDAs e *smartphones* e
- Gestão do conhecimento: amplo uso de software para *e-learning*, documentos online e mídia convencional para educar populações inteiras e organizações ao invés de apenas indivíduos.

3.2 Learning Management System

Não há um acordo entre os pesquisadores da área sobre a nomenclatura usada na área de software para *e-learning*. Segundo Tori (2010), os sistemas de gerenciamento de conteúdo e aprendizagem são conhecidos por diversas denominações, tais como AVA (Ambiente Virtual de Aprendizagem), LMS (*Learning Management System*), CMS (*Course Management System* ou *Content Management System*), LCMS (*Learning Content and Management System*) ou IMS (*Instructional Management System*). Qualquer que seja a nomenclatura que seja usada trata-se de ambientes, em geral baseados na *web*, que se destinam ao gerenciamento eletrônico de cursos e atividades de aprendizagens virtuais, podendo ser empregados em cursos a distância, como apoio a atividades presenciais ou em *blended learning*. Nesse trabalho será usada a denominação LMS para esses sistemas.

Um LMS disponibiliza uma série de recursos, síncronos e assíncronos, que dão suporte ao processo de aprendizagem, permitindo seu planejamento, implementação e avaliação, sendo que os recursos mais comumente encontrados nele são:

- Gerenciamento de curso: Criação de cursos, disciplinas, matrícula de alunos, gerenciamento de senhas, registro de atividades e de acesso realizados pelos usuários, cálculo e publicação de notas, etc.;
- Gerenciamento de conteúdo: Armazenamento, gerenciamento, edição e exibição de conteúdo multimídia;
- Disco virtual: Área de trabalho, que pode se individual ou compartilhada, na qual o usuário pode fazer *downloads*, *uploads* e visualização de conteúdos;
- Correio eletrônico (e-mail): Serviço de correio convencional; alguns permitem o envio e o recebimento de mensagens apenas dentro do sistema, outros possibilitam troca de mensagens também com o exterior;
- Mensagem instantânea: Serviço de mensagem que possibilita a comunicação síncrona e a troca de documentos entre usuários que estejam conectados ao sistema;

- Sala de bate-papo (*chat*): Sala virtual para encontros e troca de mensagens síncronas, podendo ser de texto, voz ou vídeo;
- Fórum de discussão: Recurso de comunicação assíncrona que possibilita a organização das discussões por assunto, por disciplina, por curso, por turma, por grupo, etc.;
- Quadro de avisos: Área para a publicação de informes de interesse geral e
- Avaliação: Recursos para gerenciamento da aplicação e correção de avaliações (testes de múltipla escolha ou provas dissertativas), com possibilidade de sorteio de questões e de alternativas, programação de horário para disponibilização da avaliação aos alunos, controle de tempo para realização, correção automática, cálculo e publicação de médias, geração de estatísticas e até mesmo realimentação automático ao aluno sobre o seu desempenho.

Existem diversos LMS usados no mercado. Nessa seção serão explorados dois dos mais usados atualmente aqui no Brasil.

3.2.1 Sakai

O Sakai é um LMS com código fonte aberto desenvolvido em Java por uma comunidade de universidades e empresas ao redor do mundo. Sua primeira versão foi lançada em março de 2005. De acordo com Sakai (2012), ele está em uso em mais de 350 instituições dando suporte a ensino, aprendizagem, pesquisa e colaboração.

Aqui no Brasil essa plataforma foi adotada pelo Projeto TIDIA-Ae. Esse projeto integrou diversos grupos de pesquisa de universidades paulistas e foi financiado pela FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo). O objetivo principal era desenvolver um sistema de *e-learning* que pudesse explorar o potencial da Internet Avançada e fosse capaz de oferecer suporte a diferentes necessidades dentro do contexto educacional.

3.2.2 Moodle

O Moodle, *Modular Object-Oriented Dynamic Learning Environment*, é um dos LMS mais conhecidos e usados atualmente. Ele tem evoluído desde 1999, apoiado por uma comunidade mundial de desenvolvedores, e já foi traduzido para mais de 100 línguas (Moodle, 2012).

Essa plataforma permite a criação de cursos online, páginas de disciplinas, grupos de trabalho e comunidades de aprendizagem. É um sistema de código fonte aberto feito na linguagem PHP e está em desenvolvimento constante. Como filosofia educacional ele usa uma abordagem social construtivista da educação.

O Moodle conta com 629 módulos disponibilizados em seu site. É usado por cerca de 34 milhões de usuários em 209 países diferentes, em mais de 48 mil sites registrados. Quanto a comunidade de desenvolvedores, pode-se ver no site do Moodle que ele possui 285 desenvolvedores com permissão de escrita no sistema de controle de versão utilizado. Porém, muitos módulos são desenvolvidos independentemente, o que pode elevar esse número.

3.3 Ensino de banco de dados

Em Moss (2010) é apresentado um breve histórico da área de banco de dados. A ideia do modelo entidade-relacionamento foi introduzida em um artigo de Peter Chen em 1976. Seis anos depois, Carol Chrisman introduz o uso do modelo entidade-relacionamento no ensino de banco de dados. Em 1987, Judith D. Wilson analisa alguns dos problemas associados com a forma com que os alunos usam o modelo entidade-relacionamento e como os contornam durante o processo de aprendizagem.

Segundo Heuser (2004), banco de dados é o conjunto de dados integrados que tem por objetivo atender a uma comunidade de usuários.

Nesse trabalho, entende-se que o ensino de banco de dados pode ser dividido em duas partes. Uma delas é o projeto de banco de dados e a outra se constitui de

administração e gerenciamento de banco de dados. O projeto de banco de dados inclui modelos para a descrição da estrutura de um banco de dados, enquanto na parte de administração e gerenciamento de banco de dados há uma preocupação mais prática, incluindo a aprendizagem de uma linguagem de manipulação de dados.

Para fins didáticos, um projeto de banco de dados é normalmente dividido nos seguintes tópicos:

- Abordagem entidade-relacionamento: É a técnica de modelagem conceitual de dados mais difundida. Nessa técnica, um modelo conceitual é usualmente representado através de um diagrama, chamado diagrama entidade-relacionamento (DER). Essa técnica independe do software usado para fazer a manipulação dos dados em um banco de dados, conhecido como sistema de gerência de banco de dados (SGBD);
- Abordagem relacional: É uma descrição de um banco de dados no nível de abstração visto pelo usuário de SGBD, conhecida como modelo lógico de banco de dados. Nesse modelo, os dados estão organizados na forma de tabelas;
- Transformação entre modelos: É a relação entre os dois tópicos apresentados anteriormente. O projeto de um novo banco de dados se inicia pela abordagem entidade-relacionamento, obtendo como resultado um DER. Através de algumas técnicas, é possível traduzir o DER para o formato de tabelas da abordagem relacional e
- Engenharia Reversa e Normalização: A engenharia reversa permite obter um modelo lógico relacional a partir do modelo lógico de um banco de dados não relacional, muito comum em sistemas legados. A normalização é uma técnica que tem por objetivo eliminar redundância de dados no modelo lógico relacional obtido.

Pahl, Barrett e Kenny (2004) afirmam que o desenvolvimento de banco de dados é uma atividade que requer o entendimento dos conceitos e teorias centrais, mas também a aquisição de habilidades. As atividades de modelagem e de programação

são melhores suportados em um ambiente de aprendizado pela técnica de aprender fazendo.

Dentro da parte de administração e gerenciamento de banco de dados, um dos tópicos comumente abordados é a SQL (*Structured Query Language*, ou, em português, linguagem de consulta estruturada). A SQL é uma linguagem padrão da ANSI (*American National Standards Institute*) para acesso e manipulação de banco de dados. Nesse tópico, os alunos aprendem as diversas cláusulas existentes para se formar um comando ou consulta no banco de dados, conhecido como query.

Brusilovsky et al (2008) afirmam que as metodologias de ensino de SQL existentes são focadas em apresentar conceitos e exemplos selecionados com relativamente poucos exercícios práticos. Ainda continuam dizendo que há vários sistemas para o aprendizado de SQL. Eles podem ser classificados em dois grupos. Um grupo suporta o aprendizado do estudante através de exemplos interativos, que demonstram os conceitos básicos de SQL e ilustram o comportamento dos seus componentes. Esses exemplos são frequentemente criados usando multimídia. Outro grupo suporta o aprendizado oferecendo aos estudantes problemas em SQL e avaliando suas soluções. As duas formas foram provadas efetivas.

Já para Harezlak e Werner (2012), dentre os vários problemas no ensino de banco de dados, o treinamento para a construção de *queries* SQL parece ser o desafio mais importante a ser superado. Normalmente, em software para *e-learning* da área de banco de dados, são usadas apenas páginas web simples, *quizzes*, lições e chats para a verificação da aquisição de conhecimento da SQL. Mas há ainda o problema do treinamento dos alunos em um servidor online com um sistema de gerência de banco de dados real instalado. Em muitos casos, a única solução possível é haver a exigência dos alunos instalarem um sistema desse em seus computadores pessoais, criando um banco de dados e importando dados que o professor disponibiliza através de um arquivo em um página web (HAREZLAK; WERNER, 2012).

3.4 Software para ensino de banco de dados

Nessa seção será feita uma revisão bibliográfica sobre software para e-learning existentes para o ensino de banco de dados.

3.4.1 Exploratorium

Brusilovsky et al (2010) apresentam um software para o ensino de banco de dados chamada Exploratorium. O Exploratorium pode ser visto como uma arquitetura de integração de conteúdos interativos.

Sua arquitetura é aberta e facilmente se pode acrescentar outras atividades e componentes além daquelas que já possui:

- WebEx: Há 64 exemplos interativos disponíveis de SQL com mais de 300 linhas de código comentadas. Cada exemplo consiste em um fragmento de *queries* SQL com uma explicação para cada linha importante. A interface permite aos estudantes explorar interativamente as explicações de código em uma ordem arbitrária, através do clique em um campo de seleção. É guardado o histórico de cliques de cada estudante;
- SQL-KnoT: Oferece a oportunidade para os estudantes avaliarem e praticarem suas habilidades em resolver problemas em SQL. Ele gera perguntas que requerem que um estudante escreva uma consulta SQL, avalia se a resposta está correta e provê uma realimentação ao estudante. Ele pode gerar aproximadamente 400 questões a partir de 50 modelos existentes que estão agrupados em 20 tópicos diferentes;
- SQL-Lab: Suporta a exploração interativa de cenários de problemas em SQL. A interface permite que os estudantes formulem queries SQL e observem seus resultados. Pode ser acessado diretamente do SQL-KnoT no modo de solução de problema. Nesse modo, o estudante pode ver a questão, trabalhar na solução e submeter a solução testada para o SQL-Lab.

Segundo Brusilovsky et al (2008), o Exploratorium usa um serviço adaptativo chamado QuizGuide, que é responsável por escolher as questões, baseado em dois fatores: desempenho pessoal do estudante no tópico e a relevância do tópico para o objetivo do curso.

Os professores podem utilizar o Exploratorium como se utiliza um LMS, estruturando os conteúdos segundo a ordem do curso que pretendem ministrar.

Para testar a eficácia do Exploratorium, foi feito um estudo com estudantes de graduação e pós-graduação de três universidades. A intenção era verificar vários aspectos do software: se os alunos usaram todos os conteúdos, se os alunos preferiam o modo adaptativo ou não do site e qual a percepção deles da utilidade de cada recurso.

3.4.2 SQLT-Web

Mitrovic e Hausler (2008) apresentam o SQLT-Web, a versão web do software SQL-Tutor usado para ensinar SQL. Ele foi originalmente desenvolvido em Allegro Common Lisp e disponibilizado para Windows e Solaris.

O SQL-Tutor possui uma interface gráfica, um módulo pedagógico que determina o tempo e o conteúdo das ações pedagógicas, e um modelador de estudante que analisa as respostas dos estudantes.

O software possui um conjunto de problemas e as soluções ideais a eles. Para verificar a correção da solução dos alunos, o SQL-Tutor as compara com a solução correta (especificada pelo professor).

No começo de uma sessão, o SQL-Tutor seleciona um problema para o estudante. Quando um estudante formula uma solução, o módulo pedagógico envia ao modelador de aluno, que analisa a solução, identifica erros (se houver algum) e atualiza o modelo de estudante apropriadamente. Dependendo do modelo do estudante, o módulo pedagógico gera uma ação pedagógica apropriada (isto é, uma realimentação). Quando o problema é resolvido, ou o estudante solicita um novo

problema, o módulo pedagógico seleciona um que seja apropriado com base no modelo de estudante.

O SQLT-Web segue a mesma lógica, porém, implementada usando uma arquitetura centralizada, na qual todas as funções de tutoria são feitas no servidor e as únicas funções desempenhadas no lado do cliente são as interações de usuário.

O software desenvolvido foi testado com 33 alunos em uma sessão de 2 horas de laboratório e o resultado está apresentado em Mitrovic e Hausler (2008).

3.4.3 Software *The Open University*

Em Moss (2010) é apresentada um software baseado em simulação desenvolvido na *The Open University*. Esse software pode ser personalizado usando nomes dos componentes do curso e também da notação de Chen ou notação de Crows.

O software possui diversos modos. No primeiro deles, há uma descrição das entidades e do relacionamento com uma linguagem própria, além de explicações de como elas se relacionam em termos de condição de participação.

Há um segundo modo que estabelece a relação entre o diagrama entidade-relacionamento e tabelas. Nesse modo, há menus, um que permite escolher o banco de dados e outros para escolher tabelas desse banco. As tabelas são mostradas em dois painéis e logo abaixo o diagrama entidade-relacionamento correspondente a elas.

Há um terceiro modo para a linguagem SQL. Nesse modo, há duas áreas de texto, uma para escrever a *query* SQL e a outra para mostrar a saída da *query* executada no banco de dados. É possível também escolher a *query* através de um menu.

É possível utilizar no simulador as cláusulas SELECT, FROM, WHERE, GROUP BY, HAVING e ORDER BY, com apenas um nível de aninhamento. Também é possível ver a execução passo a passo de uma *query*.

Há ainda o modo de Dependência Funcional. Há três áreas, que servem para mostrar os resultados dos termos determinantes *single*, *double* ou *triple* para achar

algumas dependências funcionais na tabela. Para isso, é possível escolher um banco de dados e uma tabela daquele banco ou uma fonte de dados.

O último modo é o de normalização. É possível escolher um banco de dados e uma tabela deste. Inicialmente o botão *Show Selection* está ativado e quando pressionado irá mostrar as colunas. Para fazer uma nova seleção, o botão *Reset Selection* deve ser pressionado, o que limpa os campos de seleção. O botão *Recombine* serve para juntar as tabelas separadas em uma só.

3.4.4 IDLE

Em Pahl, Barrett e Kenny (2004), é apresentado o IDLE (*Interactive Database Learning Environment*). O IDLE consiste de diversos serviços educacionais: leituras, tutoriais, laboratórios, auto avaliação, uma central de recursos e uma parte administrativa.

No artigo é dado foco aos três primeiros serviços. As leituras são apoiadas por textos e áudio, os tutoriais por animações e os laboratórios por mídias interativas.

Cada serviço suporta diferentes tópicos de banco de dados. Os serviços são implementados através de componentes educacionais individuais que suportam o aprendizado e o desenvolvimento de projeto:

- Um sistema de leitura com áudio provê as bases teóricas para todos os estágios de um projeto de banco de dados;
- No estágio de *design*, há um tutorial de modelagem entidade-relacionamento e um editor de entidade-relacionamento (um componente de laboratório) e
- No estágio de implementação, há um tutorial de modelo de dados e álgebra relacional, e um processador SQL (um componente de laboratório).

O editor Entidade-Relacionamento (ER) permite:

- Criar um diagrama entidade-relacionamento, apoiado pelos recursos gráficos da ferramenta, que orientam o aprendiz a criar um modelo sintaticamente correto, e

- Transformar o diagrama em uma representação de modelo de dados relacional (SQL) e armazenar esta representação SQL em um espaço de trabalho fornecido por um servidor de banco de dados subjacente.

O aprendiz faz os tutoriais de álgebra relacional para entender como um sistema de gerenciamento de banco de dados executa consultas. Assim, compreende o lado do sistema, isto é, como ele reage à entrada do usuário, que é um objetivo essencial de aprendizagem.

Então, o aprendiz usa a interface SQL para escrever e executar (isto é, definir e testar) consultas SQL. Esta parte prática começa com um tour guiado através de alguns problemas simples. A interface do processador SQL contém duas áreas. A parte de execução permite que o aluno execute consultas relacionadas ao projeto. A parte de aconselhamento fornece conselhos, comentários e links de contexto.

Pahl, Barrett e Kenny (2004) também fazem uma pesquisa sobre a opinião dos alunos. Cerca de 85% disseram apreciar o aprendizado virtual. Não foi possível distinguir a preferência dos alunos pesquisados entre laboratórios presenciais e virtuais.

Também foi feito um estudo do desempenho dos alunos. Conforme a ferramenta foi evoluindo ao longo dos anos (ela é usada na universidade desde 1999), o desempenho dos alunos melhorou, tendo ultrapassado o de cursos presenciais nos últimos dois anos.

Ainda foi estudado o comportamento dos alunos em seu uso da ferramenta. Em cerca de 84% das sessões eles fizeram uso de vários materiais, sendo que 77% das vezes foram acessados os materiais de bases teóricas. O número de requisições de aconselhamento reduziu-se em 35% nas sessões posteriores à primeira (aumento da confiança dos estudantes). Mais para o final do curso se percebeu uma mudança de comportamento dos estudantes: o aumento no uso de diversas mídias em uma mesma sessão.

3.4.5 Software *Georgia Southern University e Troy University*

Hsiang-Jui e Hui-Lien (2006) apresentam uma ferramenta para o ensino do tópico de normalização de banco de dados. A ferramenta construída é um Java *Applet* com as seguintes telas:

- Principal: É possível inserir e mostrar as dependências funcionais, normalizar o banco de dados e mostrar as relações normalizadas, além de mostrar o passo a passo usado no processo de normalização;
- Tela de resultado: Preenchendo todas as dependências funcionais, o aluno poderá ver o resultado normalizado na tela de relações normalizadas. A ferramenta normaliza o banco de dados com base nas dependências funcionais. Normaliza para a terceira forma normal e
- Tela passo a passo: Normalizando o banco de dados, o aluno poderá ver a tela que mostra o processo de normalização passo a passo. Através de botões, ele pode navegar pelo processo.

Os pesquisadores fizeram um estudo com 45 alunos para verificar a eficácia da ferramenta. Os alunos estudaram o tópico de normalização em quatro aulas de 75 minutos cada. Na primeira aula foi falada sobre a importância da normalização. Já na segunda, houve uma demonstração da técnica de normalização do livro-texto, enquanto na terceira, foram feitos exercícios. Na quarta e última, fez-se uso da ferramenta.

Foram aplicados dois questionários, um de pré-teste e outro de pós-teste. Através dele, os pesquisadores puderam verificar a preferência pelo uso da ferramenta sobre a técnica apresentada pelo livro-texto.

3.4.6 Software *Dublin City University*

Kenny e Pahl (2005) apresentam um sistema de software focado no comando SELECT. Esse comando é responsável por recuperar informação em um banco de dados, sendo o comando SQL mais fundamental.

A Escola de Computação da *Dublin City University* oferece uma disciplina online de introdução a banco de dados no segundo ano de graduação. O curso é composto de leituras em formato textual, ilustrações e áudio, além de tutoriais animados. Também estão incluídas sessões de laboratório interativo, os quais permitem aos alunos o aprendizado da SQL.

Apesar de existir outros sistemas de tutoria de SQL, nenhum deles usa uma avaliação do progresso anterior do estudante para oferecer um processo de tutoria personalizado.

A arquitetura consiste em quatro componentes principais: interface, modelo de correção, modelo pedagógico e modelo de estudante.

Toda a interação com o aluno ocorre através da interface. Há um número de tarefas primárias (ou questões) sobre um conceito SQL particular. Escolhendo umas das questões, o aluno é levado para uma tela onde pode digitar a *query* e ver seu resultado.

O sistema oferece três níveis de realimentação:

- *Error flagging*: Mostra se a submissão é considerada correta ou incorreta;
- *Hints*: Indica qual a parte da *query* está causando o erro. Uma mensagem de exemplo: “Você está usando os símbolos corretos da SQL?” e
- *Partial solutions*: Mostra um conselho preciso, sem dar a resposta. Uma mensagem de exemplo: “A questão especifica partes que não sejam de Paris. Portanto, você precisa usar o símbolo <> na resposta”.

O sistema fornece também dicas personalizadas de que atividades o aluno deve fazer dependendo dos dados gravados de sessões anteriores.

3.4.7 Software *Silesian University of Technology*

No ensino da linguagem SQL há o problema de como oferecer acesso aos estudantes a um servidor online com um sistema de gerência de banco de dados instalado. Em muitos casos, a solução usada é a instalação de um sistema desse tipo nos computadores pessoais dos alunos.

Para resolver esse problema, Harezlak e Werner (2009) propõem uma abordagem na qual muitos bancos de dados de estudantes são substituídos por um único gerenciado por um professor. Para a implementação da proposta foi escolhido o LMS Moodle, pois é possível estender as funcionalidades dele através do desenvolvimento de módulos específicos. Harezlak e Werner (2012) apresentam alguns módulos desenvolvidos para o ensino da linguagem SQL.

Os módulos foram divididos de acordo com os subtipos do SQL: SQL DCL, SQL DDL e SQL DML. Também foram feitos os módulos SQL ACID e SQL TSQL para que os usuários mais experientes pudessem praticar tópicos de banco de dados mais complexos. Apesar de diferirem em funcionalidade, todos os módulos possuem uma forma de definir os parâmetros de comunicação com o servidor de banco de dados, sendo possível escolher entre os SGBDs MySQL, Oracle, MS SQL Server e IBM DB2.

O módulo, chamado de SQL, foi desenvolvido para a realização de *queries* usando o comando SELECT. Para o professor fazer uso desse módulo através do Moodle, é preciso configurar alguns parâmetros do SGBD usado, como host, nome do banco de dados, usuário e senha.

Além disso, esse módulo também implementa um mecanismo de criação de apenas um banco de dados por aluno. Assim, evita-se o problema de duplicação de nomes de objetos e remoção de objetos de forma descontrolada, *deadlock*, dificultando ou tornando as tarefas impossíveis de serem executadas. Além disso, esse mecanismo possibilita o acesso ao professor de todas as atividades de cada estudante individualmente, permitindo uma análise detalhada das tarefas realizadas por ele. Esse mecanismo é usado nos demais módulos. A possibilidade de verificação da correção das tarefas realizadas por um estudante foi alcançada através da criação

de tabelas adicionais no servidor. Elas armazenam informações como o identificador do módulo usado, conteúdo da *query* enviada, se a cópia do módulo já havia sido executada anteriormente, etc.

Outro módulo criado foi o SQL DDL. Esse módulo permite o uso de comandos SQL do tipo DDL (*Data Definition Language*), que são aqueles que definem e modificam a estrutura de um banco de dados, incluindo tabelas, suas colunas e índices. Esse módulo analisa a correção dos comandos desse tipo, além de verificar se há violação de restrições de integridade de chave em outros comandos.

Já o módulo SQL DML é responsável pela verificação dos comandos do subtipo *Data Manipulation Language*, que é utilizado para realizar inclusões, consultas, alterações e exclusões de dados presentes em registros.

O subconjunto de comandos SQL chamado de *Data Control Language* (DCL) permite ao usuário controlar a permissão de acesso dos usuários a leitura ou manipulação de dados dentro do banco de dados. Para isso, foi feito um módulo específico para o Moodle lidar com essa categoria de comandos. Com ele, os alunos podem, depois de criar um novo usuário e conceder a ele alguns privilégios de acesso aos objetos no banco de dados, conectar como esse novo usuário e tentar executar uma operação no objeto.

Como parte do treinamento em SQL, os alunos precisam adquirir conhecimento em na definição do escopo de uma transação (*transaction*) e como o servidor de banco de dados garante suas propriedades. Uma transação é uma sequência de operações em um SGBD que são tratadas como um bloco único e indivisível. É usada para manter a consistência dos dados no banco de dados em caso de falhas e para se recuperar corretamente dessas falhas. Para isso, foi desenvolvido o módulo SQL ACID, que permite aos alunos verificar as propriedades de uma transação: Atomicidade, Consistência, Isolamento e Durabilidade. Além disso, esse módulo possibilita que os alunos façam um backup do banco de dados através de um comando SQL ou através de um botão na interface gráfica.

O último módulo desenvolvido é o SQL TSQL. Nesse módulo, há a preocupação com os procedimentos armazenados (*stored procedures*) e os gatilhos (*triggers*). Os procedimentos armazenados permitem alocar a lógica de negócio no servidor e

basicamente são uma coleção de comandos SQL. São usados para tarefas repetitivas. Os gatilhos são procedimentos armazenados especiais, que são executados sempre que um determinado evento associado ocorre. São muito utilizados para ajudar a manter a consistência dos dados ou para propagar alterações em um determinado dado de uma tabela para outras. Para acessar o módulo TSQL, o aluno deve ter passado antes pelos módulos SQL, DML, DDL e DCL, ou passar por um teste interativo que permite que ele demonstre seu conhecimento nos diversos subtipos de SQL. Se o aluno atender aos requisitos, ele terá acesso a interface de execução dos comandos de programação de banco de dados.

3.4.8 Comparação entre software para ensino de banco de dados

Analisando os tópicos de ensino de banco de dados que podem ser explorados com o uso de cada software apresentado foi feita uma comparação que é mostrada na Tabela 2.

O tópico mais abordado é a linguagem SQL, pois requer treinamento dos alunos e um aprendizado efetivo envolve aulas práticas em laboratório.

Tabela 2: Comparação entre sistemas de software para ensino de banco de dados

	Abordagem entidade-relacionamento	**Abordagem relacional**	**Transformação entre modelos**	**Normalização**	**Comandos SQL**
Exploratorium					X
SQLT-Web					X
The Open University	X	X	X	X	X
IDLE	X				X
Southern University e *Troy University*				X	
Dublin City University					X
Silesian University of Technology					X

4 USO DO SEMAT PARA LEVANTAMENTO DE REQUISITOS EM SOFTWARE PARA *E-LEARNING*

Nesse capítulo será apresentado um estudo de caso aplicando o modelo SEMAT para o levantamento de requisitos de um software para *e-learning*. O estudo de caso é na área de ensino de banco de dados.

Na aplicação em questão, será dada ênfase a apenas quatro alfas do SEMAT: oportunidade, *stakeholders*, requisitos e sistema de software, uma vez que são aqueles relacionados diretamente aos requisitos do software, foco desse trabalho. Os alfas relativos à área de interesse do esforço (trabalho, equipe e modo de trabalho) não serão analisados, pois não há uma equipe com divisão de tarefas que possibilite essa análise. O método escolhido para determinar o estado atual do esforço a cada iteração foi o método de *walkthrough*.

Também se escolheu usar iterações curtas de uma semana e colocar as tarefas a serem executadas em um *backlog* de iteração por permitir um melhor controle das atividades e por possibilitar um replanejamento mais rápido no caso de haver alguma mudança de rumo. Apesar de ser interessante o uso de uma lousa com as tarefas para o acompanhamento da evolução dos alfas e das tarefas, como sugerido por Jacobson et al. (2012), isso não será usada nesse projeto pelo fato da equipe possuir apenas uma pessoa nesse momento. O acompanhamento será feito através da ferramenta de gerenciamento de tarefas existente na agenda do Google por possibilitar o envio de lembretes das tarefas pendentes por e-mail ou por alertas no celular com sistema Android.

Para demonstrar a aplicação do modelo SEMAT e o desenvolvimento iterativo do sistema de software, serão explicados os detalhes de cada iteração.

4.1 Primeira iteração

Não existem estados atuais do esforço na primeira iteração pois o projeto está sendo iniciado. Já os estados que devem ser alcançados ao fim da iteração são os primeiros estados de cada um dos alfas, uma vez que é um projeto novo. Porém, entre os alfas existem prioridades. Primeiramente, é necessário reconhecer quais são os stakeholders, depois identificar a oportunidade, posteriormente deve-se trabalhar nos requisitos e selecionar a arquitetura a ser utilizada, como pode ser visto na Tabela 3.

Tabela 3: Estados alvo da iteração 1

Estado alvo	**Como se planeja atingir o estado alvo**
Stakeholders – Reconhecidos • Todos os diferentes grupos de *stakeholders* que são ou serão afetados pelo desenvolvimento e operação do sistema de software são identificados. • Há um acordo sobre os grupos de *stakeholders* a serem representados. No mínimo, os grupos de stakeholders que fundaram, usam, suportam e mantém o sistema devem ser considerados. • As responsabilidades dos representantes dos *stakeholders* estão definidas.	Para reconhecer quais são os stakeholders deve ser observada e conhecida uma realidade de sala de aula e de ensino de banco de dados. Além disso, deve ser observada a estrutura administrativa em que se insere essa realidade. Depois de observada a realidade escolar, deve ser elaborado um relato escrito compilando essas informações.
Oportunidade – Identificada • Uma ideia para melhorar formas atuais de trabalho, aumentando a participação no mercado ou a aplicação de um novo ou inovador sistema de software foi identificada. • Pelo menos um dos *stakeholders* deseja fazer um investimento por ter uma melhor compreensão da oportunidade e pelo valor	Para identificar a oportunidade, será elaborado e aplicado um questionário para os *stakeholders* reconhecidos. Também serão usadas entrevistas com stakeholders para entender melhor a oportunidade. Ainda deve ser analisada a bibliografia existente para conhecer as soluções existentes na área de aplicação.

associado ao abordá-lo. • Os outros *stakeholders* que compartilham a oportunidade foram identificados.	
Requisitos - Concebido • O conjunto inicial de *stakeholders* concorda que um sistema deve ser desenvolvido. • Os *stakeholders* que usarão o novo sistema estão identificados. • Os *stakeholders* que fundarão o trabalho inicial no novo sistema estão identificados. • Há uma oportunidade clara para um novo sistema.	Atingir esse estado está relacionado com atingir os estados dos alfas anteriores. O questionário realizado com os *stakeholders* deve ser analisado para entender as possibilidades de solução, bem como os resultados das entrevistas.
Sistema de Software - Arquitetura Selecionada • Os critérios a serem usados para selecionar a arquitetura foram acordados. • Plataformas de hardware foram identificadas. • Linguagens de programação e tecnologias a serem usadas foram selecionadas. • Os limites do sistema são conhecidos. • Decisões significativas sobre a organização do sistema foram tomadas. • Decisões sobre compras, construção e reuso foram tomadas.	Serão escolhidos critérios de seleção para a arquitetura do software a ser desenvolvido.

A partir dos itens das *checklists* de cada estado dos alfas foi construído o *backlog* de iteração, pensando-se em tarefas que contribuíssem para se atingir os objetivos daquela iteração. O *backlog* é mostrada na Tabela 4.

Tabela 4: *Backlog* da iteração 1

Alfas	**Tarefas**
Stakeholders	Tarefa 1: Observar a realidade escolar e do ensino de banco de dados em uma instituição de ensino Tarefa 2: Sistematizar por escrito quais são os *stakeholders* e qual seu papel em relação ao projeto
Oportunidade	Tarefa 3: Elaborar um questionário a ser aplicado para um tipo de *stakeholder* identificado Tarefa 4: Aplicar o questionário para um tipo de *stakeholder* Tarefa 5: Fazer entrevista não-estruturada com os demais tipos de *stakeholders* Tarefa 6: Analisar a bibliografia existente sobre software existente na área de aplicação
Requisitos	Tarefa 7: Analisar os resultados obtidos a partir da aplicação do questionário Tarefa 8: Analisar os resultados obtidos a partir da entrevista
Sistema de software	Tarefa 9: Elaborar uma listagem de critérios de seleção para a arquitetura do software a ser desenvolvido

A seguir será feita uma descrição detalhada de como cada tarefa foi cumprida.

4.1.1 Tarefas um e dois

A partir da observação da realidade escolar de uma instituição em que ocorre o ensino de banco de dados, foi possível identificar os seguintes *stakeholders* para o projeto: alunos, professores, coordenadores e direção da instituição de ensino, além dos desenvolvedores do sistema de software.

Os alunos e os professores serão usuários do sistema de software a ser desenvolvido. Portanto, serão os beneficiados direto do projeto e precisam ter suas opiniões levadas em consideração. A coordenação e a direção não necessariamente usarão o sistema diretamente, mas têm interesse em proporcionar melhores condições de ensino e aprendizagem de banco de dados aos alunos da instituição.

Os desenvolvedores do sistema de software são responsáveis pela especificação, codificação e implantação do sistema a ser desenvolvido.

4.1.2 Tarefas três e quatro

Uma vez que os *stakeholders* foram identificados, foi elaborado um questionário a ser aplicado aos alunos, já que é o grupo com a maior quantidade de membros, impossibilitando o uso do método de entrevista. Também foram desenvolvidos a carta de informação ao sujeito de pesquisa, a carta de informação à instituição e o termo de consentimento livre e esclarecido como uma forma de formalizar a pesquisa realizada perante a instituição e esclarecer a participação dos alunos.

O questionário proposto foi validado com uma amostra de seis alunos, e a partir de suas sugestões e dúvidas demonstradas durante o seu preenchimento, foram feitos melhorias nele.

O questionário final, apresentado no Apêndice A, apresenta três partes. A primeira tem por objetivo colher alguns dados demográficos, como idade, turma e gênero. Na segunda parte, há perguntas que procuram identificar o perfil de uso da internet do usuário (de onde e com qual frequência ocorre o acesso, quais serviços são

utilizados, etc.). O objetivo da terceira parte é saber a opinião dos alunos sobre as dificuldades enfrentadas no aprendizado dos tópicos referentes a banco de dados e sobre seu interesse em possíveis ferramentas oferecidas para facilitar o estudo de algum dos tópicos dessa disciplina.

A carta de informação ao sujeito de pesquisa teve por objetivo esclarecer aos alunos, assim como à instituição, sobre o assunto da pesquisa e o questionário usado na coleta de dados. Além disso, foram apresentadas informações quanto à identificação da pesquisadora e a instituição na qual cursa o mestrado. Foram abordadas as questões quanto ao sigilo absoluto sobre as questões respondidas pelos alunos e também sobre o nome dos participantes, além de esclarecer que a pesquisa possuía finalidade acadêmica. A carta de informação ao sujeito de pesquisa e a carta de informação à instituição são apresentadas nos Apêndices B e C, respectivamente.

O termo de consentimento livre e esclarecido foi apresentado na mesma página da carta de informação e possui um campo para o preenchimento do nome do aluno ou do nome do representante da instituição. Através deste termo, o participante concorda em participar da pesquisa proposta após sua assinatura. Ambos são apresentados nos Apêndices B e C, respectivamente.

Primeiramente, foi pedida autorização da instituição de ensino para a aplicação do questionário com os alunos. Posteriormente, foi explicado em cada uma das turmas pesquisadas o propósito da pesquisa e do questionário. Os alunos que concordaram em participar da pesquisa assinaram o termo de consentimento livre e esclarecido e preencheram os questionários.

4.1.3 Tarefa cinco

Para os grupos de *stakeholders* professores, coordenadores e direção escolar foi escolhida ser feita uma entrevista não estruturada para colher suas expectativas em relação a um novo sistema de software na área de banco de dados.

As entrevistas não seguiram um roteiro pré-definido, tendo um caráter de conversa informal. Foram entrevistados dois professores da área de banco de dados da instituição pesquisada, o professor responsável pelos laboratórios de informática, o diretor e uma coordenadora. A análise da entrevista será feita na **Tarefa sete**.

4.1.4 Tarefa seis

O levantamento bibliográfico da pesquisa foi feito no capítulo 3 e foram encontrados sete sistemas de software para ensino de banco de dados: Exploratorium, SQLT-Web, Software *The Open University*, IDLE, Software *Georgia Southern University* e *Troy University*, Software *Dublin City University* e os módulos para Moodle da *Silesian University of Technology*.

Como se pode perceber, a maioria dos sistemas de software é usada no ensino da linguagem SQL por ser um tema de ordem prática e que exige treinamento dos alunos.

4.1.5 Tarefa sete

Os questionários aplicados na instituição de ensino escolhida foram compilados e sua análise é apresentada nesse item.

4.1.5.1 Amostra dos questionários

Os questionários foram aplicados para alunos de cursos técnicos em Informática e Informática para Internet. Os alunos participantes da pesquisa eram dos três semestres do curso técnico em Informática e dos dois últimos semestres do curso técnico em Informática para Internet, tanto do período vespertino quanto noturno. Esses estudantes cursam ou cursaram as duas disciplinas de banco de dados que existem na grade curricular da instituição. A primeira disciplina é composta por

conteúdos pertinentes a parte de projeto de banco de dados, enquanto que a segunda versa sobre o gerenciamento e administração de banco de dados, tendo como foco a linguagem SQL.

Foram respondidos 106 questionários, uma boa amostra já que, segundo os registros escolares, há 160 alunos nas turmas pesquisadas, de acordo com a Tabela 5. Foram desconsiderados seis questionários devido a um preenchimento incompleto.

Tabela 5: Número de alunos e questionários por turma

Turma	**Nº alunos**	**Nº questionários**
2AWT	25	9
3AWT	14	13
2AWN	24	15 (1 desconsiderado)
3AWN	19	16 (2 desconsiderados)
1AIN	32	29
2AIN	18	11 (2 desconsiderados)
3AIN	28	13 (1 desconsiderado)

4.1.5.2 Automação do processamento de dados coletados

O método escolhido para a aplicação do questionário foi em papel. Porém, com o objetivo de facilitar tanto a inserção de dados no computador quanto à compilação desses dados foi decidido pelo uso de alguma ferramenta baseada na rede.

A ferramenta escolhida foi o Google *Forms*. Com ele, os dados de cada questionário puderam ser inseridos de forma mais rápida e prática pelo fato de estar disponível *online* e acessível via *browsers*. Além disso, a ferramenta possibilita a visualização e exportação dos dados colhidos em formato de planilha. Foi escolhida a alternativa de

se exportar os dados e sua análise foi realizada no software Open Office *Spreadsheets*.

4.1.5.3 Análise dos dados coletados

Nessa seção é feita a análise dos dados coletados nos questionários aplicados aos alunos.

Para ter um perfil etário, foi feito um levantamento das idades dos alunos participantes da pesquisa. Segundo o Gráfico 1, 60% deles são jovens com até 24 anos.

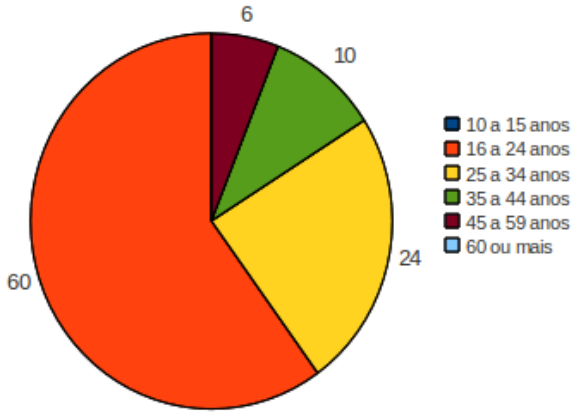

Gráfico 1: Idade dos alunos pesquisados

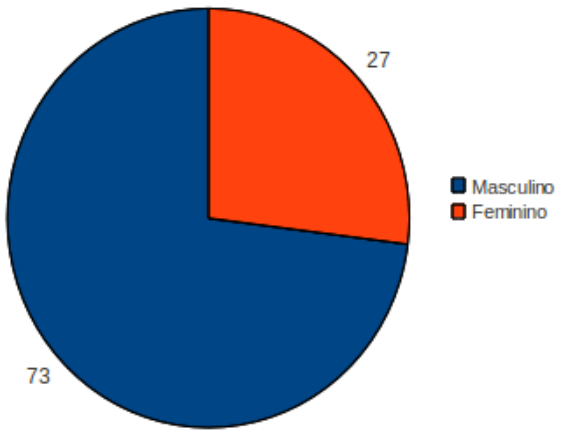

Gráfico 2: Gênero dos alunos pesquisados

Dente os entrevistados, pode-se observar pelo Gráfico 2 que há uma predominância do sexo masculino, representando 73% da amostra.

Das pessoas entrevistadas, nenhuma relatou não usar a internet. Sendo assim, 100% dos entrevistados usam a rede.

No Gráfico 3, é mostrado a distribuição dos entrevistados pelas turmas.

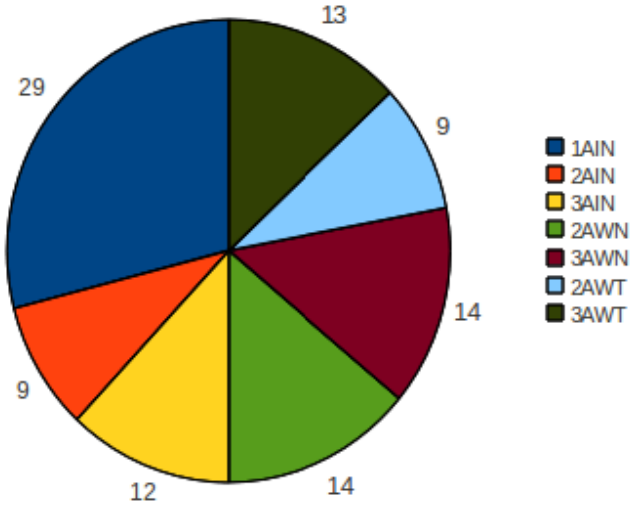

Gráfico 3: Alunos por turma

Quanto ao uso de internet em diferentes locais, cerca de 95% acessam a rede em casa e 84% o fazem da escola. O acesso em centros públicos de acesso pago (lanhouses) é de 40%, enquanto o acesso em centros públicos de acesso gratuito é de 38% na pesquisa realizada (Gráfico 4).

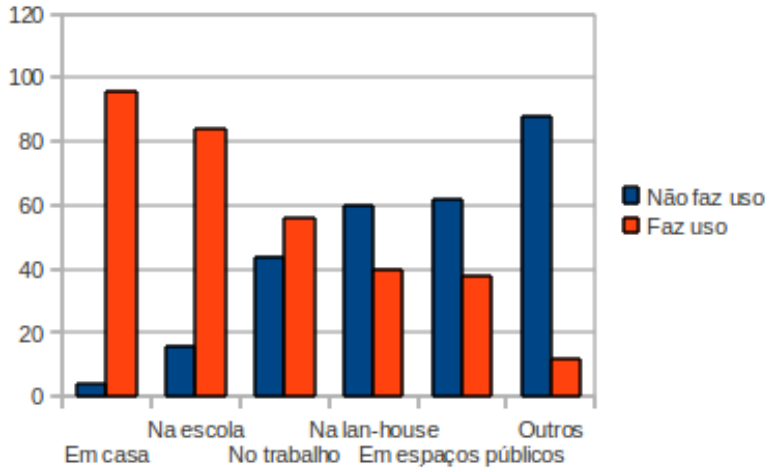

Gráfico 4: Locais de uso da internet

O Gráfico 5 apresenta a distribuição do uso da internet em diferentes locais.

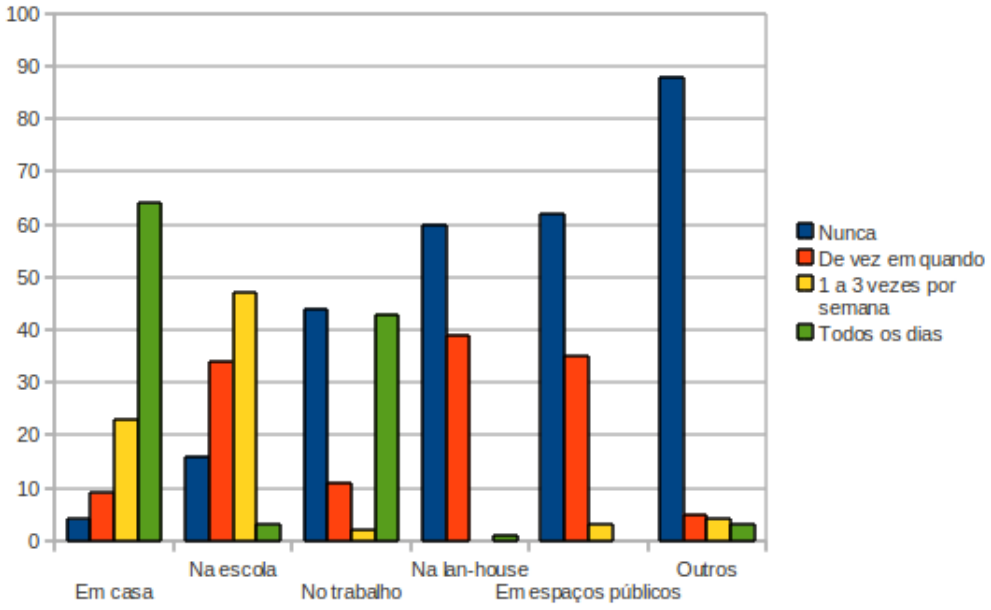

Gráfico 5: Acesso a internet por locais

A pesquisa realizada mostra que apenas cerca de 5% dos participantes não utilizam a internet em casa, porém não é possível saber o motivo pelo qual isso acontece (se por falta de computadores no domicílio ou por falta de acesso à rede).

Dentre os alunos pesquisados, pode-se perceber que quase 65% dizem acessar a internet diariamente em casa. Mais de 80% dos participantes acessam a internet na escola, apesar de o acesso diário apresentar uma baixa porcentagem. O uso em espaços públicos (pagos e gratuitos) tem caráter esporádico, sendo que cerca de 60% dos participantes não faz uso de *lanhouses* e 62% não acessa a internet através de espaços públicos gratuitos como os telecentros. Um pouco mais de 10% dos pesquisados fazem uso da internet em outros locais, sendo que os locais mais indicados nesse caso foram casas de amigos e parentes.

Quanto aos dispositivos utilizados no acesso a internet, pode-se perceber pelo Gráfico 6 que a grande parcela dos acessos é feitos ainda por computadores *desktop*. Nota-se também que os dispositivos móveis, como celular e *handhelds*, possuem uma penetração semelhante ao dos *laptops*; um pouco mais de 20% dos participantes dizem já os terem usado para acessar a internet, sendo que a maior parte dos usuários os utiliza esporadicamente.

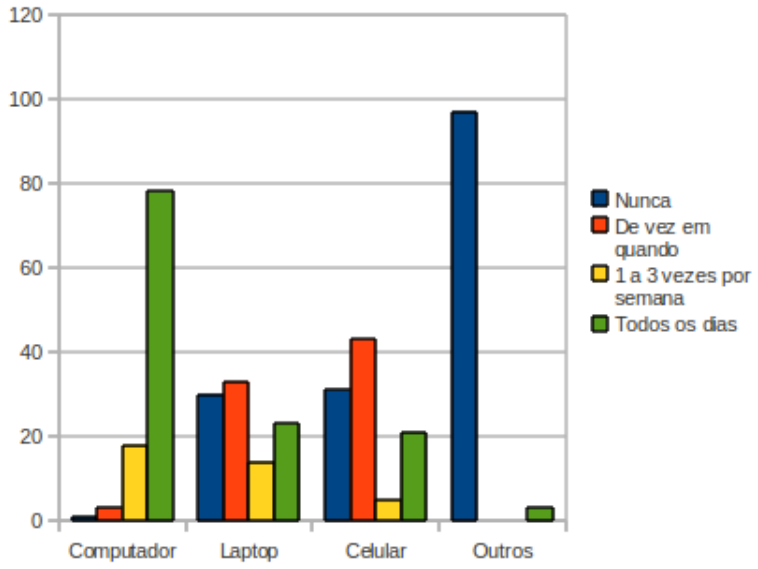

Gráfico 6: Uso de dispositivos para acesso a internet

No Gráfico 7, percebe-se que o e-mail é o serviço que os participantes acessam com maior frequência: aproximadamente 70% faz uso dele diariamente. Além disso, não houve participante que tenha declarado não fazer uso dele ainda que seja esporadicamente. Ao contrário do que é senso comum, os sites de relacionamento

aparecem como o segundo serviço mais acessado diariamente, com quase 50% das pessoas fazendo uso diário deles. O terceiro serviço mais usado diariamente é o acesso a *blogs* e portais de notícias com pouco mais de 40% dos pesquisados, seguido bem de perto pelo uso diário de mensageiros instantâneos. Como se podem constatar jogos, fóruns e cursos a distância possuem acesso esporádico entre os participantes da pesquisa, sendo grande a porcentagem de pessoas que dizem nunca fazer uso deles. O acesso a vídeos, apesar de esporádico (com mais de 30% das pessoas dizendo que usam “de vez em quando"), possui uma baixa porcentagem de participantes que não fazem uso desse serviço.

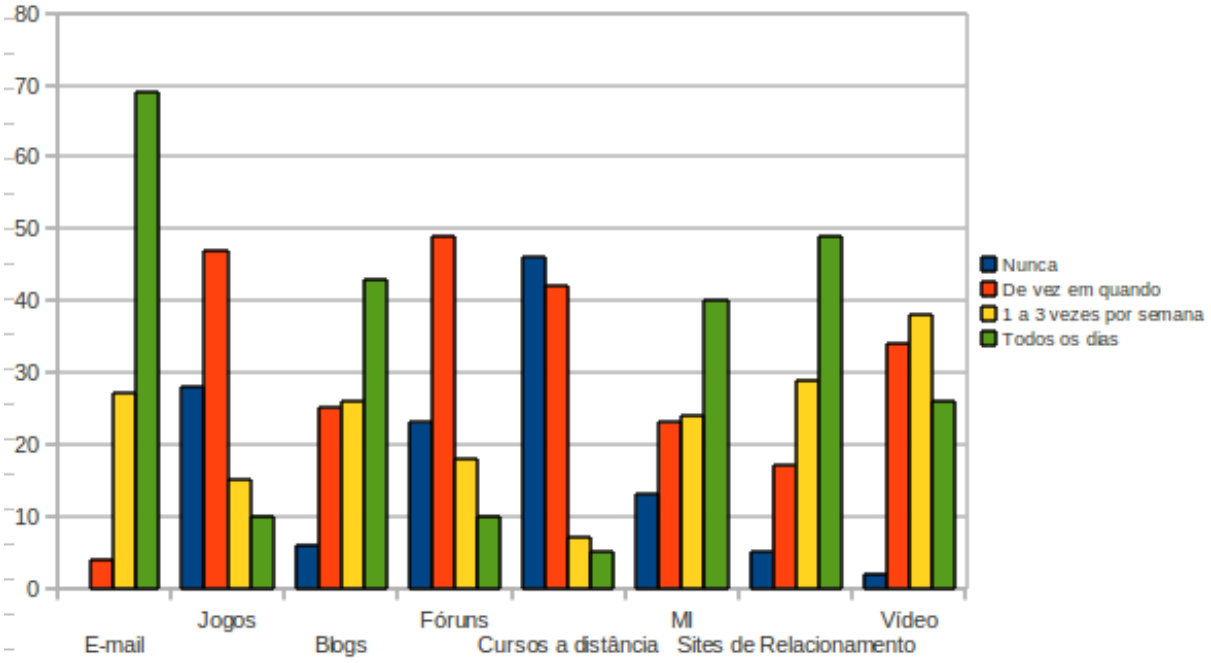

Gráfico 7: Serviços usados na internet

Quanto a perguntas específicas sobre o nível de dificuldade no aprendizado na área de banco de dados, pode-se verificar no Gráfico 8 que o tópico de abordagem entidade-relacionamento foi tido como difícil por 16% dos pesquisados, o de abordagem relacional por 25%, o de transformação de modelos por 39%, o de normalização por 34% e o de comandos SQL por 23%. Porém, como foram também pesquisados estudantes que estão cursando a primeira disciplina de banco de dados, alguns tópicos ainda não haviam sido contemplados, tendo uma

porcentagem elevada de resposta “Não foi ensinado”: 40% para o tópico de normalização e 30% para comandos SQL. Se forem descontadas as pessoas que responderam “Não foi ensinado”, os tópicos de normalização e comandos SQL são considerados difíceis por 56,7% e 32,8% dos alunos respectivamente.

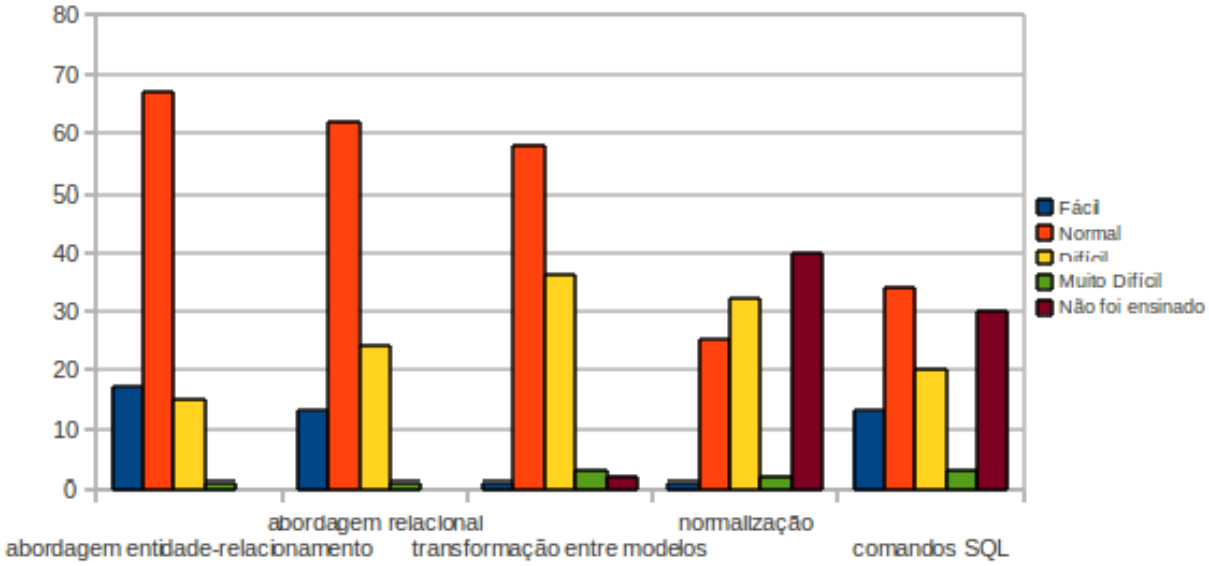

Gráfico 8: Dificuldade de aprendizado por tópico

De acordo com o Gráfico 9, 94% dos participantes acham que a ideia de um site ou software para o ensino de banco de dados é interessante, enquanto apenas 6% possuem uma visão neutra da ideia. Isso reforça que a ideia do desenvolvimento de um sistema de software para área apresenta relevância entre pessoas interessadas na temática.

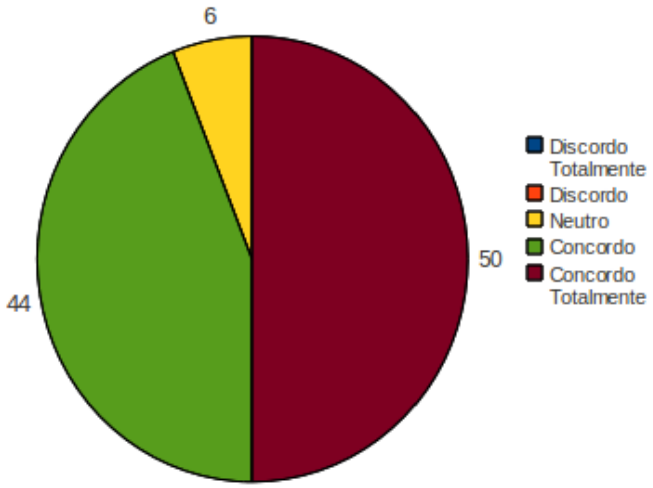

Gráfico 9: Interesse em uma ferramenta para ensino de banco de dados

O tópico em que os estudantes acreditam que a existência de um software poderia ajudar no seu entendimento é o dos comandos SQL, com 45% das respostas, como pode ser visto no Gráfico 10, apesar de ser o terceiro tópico em nível de dificuldade segundo a pesquisa, atrás de normalização e transformação entre modelos.

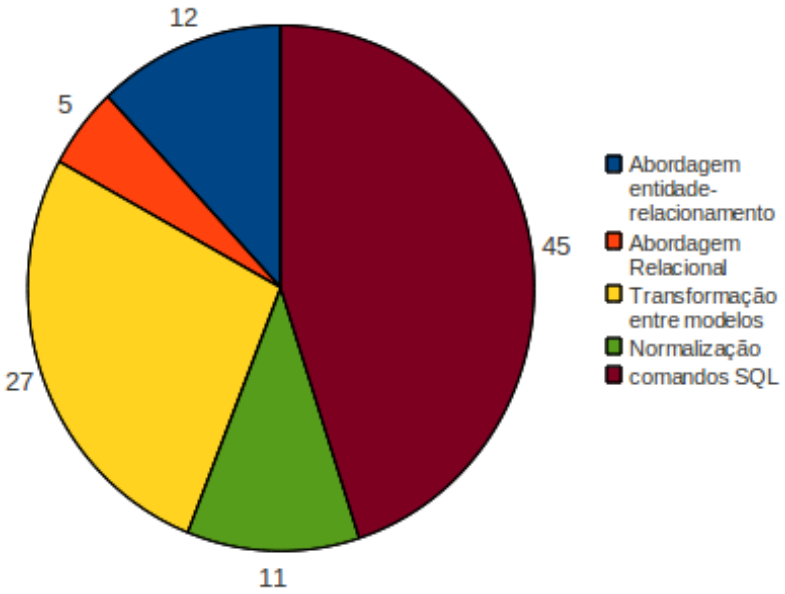

Gráfico 10: Utilidade de uma ferramenta para um tópico

Ainda foi perguntado se os participantes da pesquisa já fizeram algum curso a distância, a qual 32% responderam afirmativamente (Gráfico 11). Dentre os cursos feitos nessa modalidade, predominou cursos na área de informática, como os de linguagens de programação, de Excel e de servidores. Foram também citados cursos fora da área de informática, como gestão de negócios, contabilidade, entre outros.

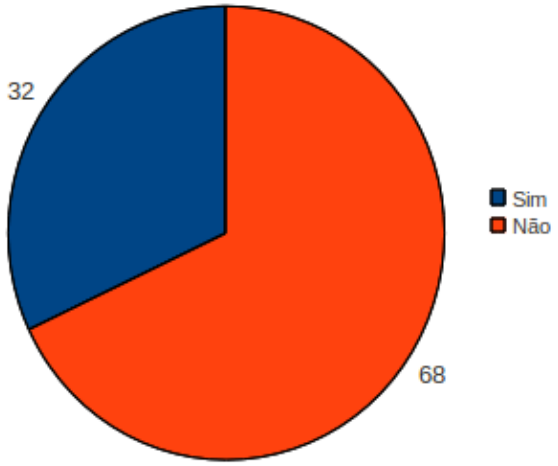

Gráfico 11: Alunos que fizeram cursos a distância

4.1.6 Tarefa oito

A partir da entrevista com os *stakeholders* professores, coordenação e direção foi possível identificar alguns problemas e oportunidades a serem pensadas em um novo sistema de software.

Quanto à infraestrutura tecnológica, a instituição pesquisada conta atualmente com um *link* de internet de apenas oito megabytes a ser distribuído entre seis laboratórios de informática com 20 computadores cada, além de cerca de 10 computadores na área administrativa. A configuração dos computadores é Pentium Duo Core, com 4GB de RAM, 350 GB de HD e o sistema operacional Windows 7. A rede da instituição está dentro da rede do governo, conhecida como Intragov. Com isso, há

restrições de acesso a vários *sites*, além da impossibilidade dos alunos e professores fazerem *download*. Apenas o professor responsável pelos laboratórios possui a permissão de *download* e de instalação de software nas máquinas.

Para o ensino das disciplinas de banco de dados, os computadores possuem três SGBDs instalados: Microsoft Access, MSSQL e MySQL. Além disso, há uma interface gráfica via *web* instalada para administração do SGBD MySQL chamada PHPMyAdmin. Isso dá flexibilidade aos professores da área para que possam usar as ferramentas que desejam ou até mesmo uma combinação delas.

Além disso, há a disposição dos professores uma plataforma Moodle para a disponibilização dos conteúdos e materiais das disciplinas que lecionam. Não há uma obrigatoriedade no uso da plataforma, mas seu uso é incentivado.

Segundo relatos dos professores entrevistados da área de banco de dados, há um problema ao se lecionar a linguagem SQL que é a continuidade das atividades em diferentes momentos (em diferentes aulas ou em casa). Algumas vezes, os alunos estão desenvolvendo alguma atividade prática e não conseguem finalizá-la em laboratório no tempo de aula. Atualmente, as atividades práticas são feitas usando algum dos SGBDs instalados localmente nas máquinas do laboratório de informática da instituição. Para poderem continuar em casa ou na próxima aula, é necessário exportar o banco de dados toda aula e terem o software instalado em seus computadores (o que algumas vezes não é possível devido usarem máquinas de parentes, *lanhouses* e outras).

O uso do PHPMyAdmin poderia resolver de certa forma o problema citado acima se ele fosse disponibilizado em um servidor da instituição com acesso externo para os alunos, no qual pudessem deixar seus bancos de dados *online*. Porém, essa opção ainda não foi possível de ser implantada.

Com o uso do PHPMyAdmin e do MySQL no ensino da linguagem SQL, os professores puderam observar que os alunos tem dificuldade em usá-los devido à interface ser toda em inglês, além dos erros e *feedbacks* também serem nessa língua. No caso do MySQL ainda se soma a dificuldade em lidar com a linha de comando.

Outro desafio citado é a questão da internet. Para a quantidade de computadores da escola, o *link* disponibilizado não é suficiente para um acesso concomitante de todos os laboratórios, provocando lentidão. Além disso, os alunos atendidos pela instituição relatam que, na região, há uma grande dificuldade quanto aos *links* de banda larga.

Outra questão em aberto é a avaliação das atividades práticas realizadas nessas ferramentas. Devido à quantidade de alunos em uma turma, pode ser difícil para o professor dar atenção individualizada e conseguir diagnosticar as dificuldades no uso da linguagem SQL de cada um deles para propor ações que possam superá-las.

4.1.7 Tarefa nove

A partir do que foi analisado nas tarefas anteriores dessa iteração, foram estabelecidos alguns critérios de seleção para a arquitetura do sistema de software a ser desenvolvido:

- Integração com os SGBDs utilizados na instituição;
- Integração com outros sistemas de software para *e-learning* usados na instituição;
- Acesso *online* de banco de dados pelos alunos;
- Possibilidade de sincronização com dados *off-line* e
- Possibilidade de extensão para uso em diferentes tópicos dentro do ensino de banco de dados.

4.1.8 Resultado da primeira iteração

Após o término da primeira iteração, foi feita a análise dos estados atuais dos alfas escolhidos para serem trabalhados. Para analisar se o alfa chegou ao estado alvo utilizou-se da *checklist* sugerida no modelo SEMAT. Abaixo será detalhado como foram atingidos cada um dos estados alvo desses alfas.

4.1.8.1 *Stakeholders*

O estado alvo para o alfa *stakeholders* era o primeiro, chamado de reconhecidos. Esse estado foi plenamente atingido, pois foram identificados os diferentes grupos de *stakeholders* que são ou serão afetados pelo desenvolvimento e operação do sistema de software e há um acordo sobre os grupos de *stakeholders* que devem ser representados. Os grupos são os alunos, os professores, os coordenadores, a direção da instituição de ensino e os desenvolvedores.

Ainda estão definidas as responsabilidades dos representantes dos *stakeholders*. Os alunos e os professores serão os usuários do sistema. Portanto, serão consultados pelos desenvolvedores durante todo o processo de desenvolvimento do sistema de software para o entendimento de suas reais necessidades. Os coordenadores e a direção serão consultados para decisões estratégicas envolvendo o sistema de software, como, por exemplo, como será disponibilizado aos alunos. Já os desenvolvedores tem o papel de consultar os demais *stakeholders*, entender suas necessidades, implementar e implantar o sistema de software desenvolvido.

4.1.8.2 Oportunidade

No caso do alfa oportunidade, o estado alvo era o de identificada. Foi plenamente atingido ao fim dessa iteração. Foi identificada uma ideia para melhorar formas atuais de trabalho, ou seja, de melhorar a forma como se ensina banco de dados. Através da análise dos questionários, das entrevistas e da bibliografia existente sobre a área de aplicação, pode-se perceber que uma oportunidade a ser explorada seria o desenvolvimento de um sistema de software para ensino de banco de dados.

Além disso, foi identificado um dos *stakeholders* que deseja fazer um investimento na abordagem da oportunidade. O *stakeholder* em questão possui papéis múltiplos, já que tanto é professor da área de banco de dados quanto é desenvolvedor do sistema de software. O investimento feito será em tempo.

Ainda os outros *stakeholders* que compartilham a oportunidade foram identificados, que são os mesmos já citados no alfa *stakeholders*.

4.1.8.3 Requisitos

O estado alvo para o alfa requisitos era o de concebido. Esse estado foi plenamente atingido, pois o conjunto inicial de *stakeholders* concorda que um sistema de software deve ser desenvolvido, como se pode perceber pelos dados obtidos com os questionários e com as entrevistas.

Além disso, os *stakeholders* que usarão o novo sistema de software foram identificados e os *stakeholders* que fundarão o trabalho inicial no novo sistema estão identificados (como exposto no alfa *stakeholders*).

Para finalizar, há uma oportunidade clara para um novo sistema de software, como foi explicado no alfa oportunidade.

4.1.8.4 Sistema de software

No caso do alfa sistema de software, o estado alvo era o de arquitetura selecionada. Esse estado não foi plenamente atingido ao fim dessa iteração. Apenas foram acordados critérios a serem usados para selecionar a arquitetura.

Para ser plenamente atingido esse estado ficou faltando identificar as plataformas de hardware, selecionar as linguagens de programação e tecnologias a serem usadas, conhecer os limites do sistema e tomar decisões significativas sobre a organização do sistema e sobre compras, construção e reuso.

4.2 Segunda iteração

A partir da segunda iteração serão feitas descrições mais breves sobre o processo, já que foi possível verificar como uma iteração é completada em detalhes através da explicação da primeira iteração.

De início, os estados atuais dos alfas para essa iteração são aqueles que foram atingidos ao fim da iteração anterior: reconhecidos (alfa *stakeholders*), identificada (alfa oportunidade) e concebido (alfa requisitos). Para o alfa sistema de software, o estado de arquitetura selecionada não foi plenamente atingido na primeira iteração.

Tendo em vista os estados atuais, os estados que devem ser alcançados ao fim dessa iteração devem ser os próximos: representados (*stakeholders*), solução necessária (oportunidade) e delimitado (requisitos). Para o alfa sistema de software, deve ser atingido plenamente o estado de arquitetura aelecionada.

Para os *stakeholders* serem considerados representados, os mecanismos para envolvê-los foram acordados entre todos e os representantes dos *stakeholders* foram nomeados. A oportunidade atinge o estado de solução necessária quando a necessidade para uma solução baseada em software for confirmada. Já para os requisitos estarem delimitados, o propósito e o tema do novo sistema de software devem estar claros.

A partir da checklist de cada um dos estados foram pensadas as tarefas necessárias para se atingir o estado de cada um dos alfas. As tarefas estão mostradas na Tabela 6.

Tabela 6: *Backlog* da iteração 2

Alfas	**Tarefas**
Stakeholders	Tarefa 1: Fazer uma reunião com os *stakeholders*
Oportunidade	
Requisitos	Tarefa 2: Elaborar um modelo de como os requisitos serão descritos Tarefa 3: Elaborar um mecanismo de gerência de requisitos Tarefa 4: Elaborar uma listagem com as restrições do projeto Tarefa 5: Elaborar uma listagem com as premissas do projeto
Sistema de Software	Tarefa 6: Elaborar uma listagem com as plataformas de hardware necessárias Tarefa 7: Elaborar uma listagem com as linguagens de programação e as tecnologias a serem usadas Tarefa 8: Decidir arquitetura a ser usada na construção do sistema de software

A descrição de como cada tarefa foi cumprida é feita nos tópicos seguintes.

4.2.1 Tarefa um

Foi planejada uma reunião conjunta com representantes de todos os grupos de *stakeholders* reconhecidos na primeira iteração. O intuito era de apresentar os resultados da análise de dados dos questionários e das entrevistas feitas e para entrar em acordo sobre a forma de trabalho durante o desenvolvimento do sistema de software.

Porém, não foi possível articular a agenda de todos os envolvidos para que houvesse uma reunião única. Alternativamente a isso foi feita uma reunião com os *stakeholders* coordenadores e, em outro momento, uma conversa informal com alguns representantes dos alunos. A direção não pode estar envolvida nessas reuniões. No caso do grupo de *stakeholders* professores, o papel está sendo duplamente exercido pela autora desse projeto, que também atua como professora na instituição. Logo, não houve a necessidade de uma reunião com esse grupo.

Além de serem apresentados os resultados obtidos nessas reuniões, foi pedida a colaboração dos coordenadores e dos alunos durante o processo de desenvolvimento do sistema de software, sendo que eles concordaram e houve a permissão por parte da instituição em dar continuidade no projeto. Também foi explicado o modo de trabalho durante o desenvolvimento. Em um primeiro momento, os coordenadores irão acompanhar o trabalho a distância sem influenciar diretamente no levantamento de requisitos do sistema de software. Já para representar os *stakeholders* alunos durante o processo, alguns alunos se voluntariaram a ajudar de uma forma constante.

Nessas reuniões, foi possível também confirmar que os *stakeholders* acreditam que uma solução baseada em software para o ensino de banco de dados é necessária. Foi apresentada uma proposta de solução que seria a construção de *plugins* para o Moodle para o ensino da linguagem SQL, o que foi bem aceita pelos *stakeholders*.

4.2.2 Tarefa dois

É preciso elaborar um modelo de como os requisitos serão descritos. Para esse projeto, foi escolhido descrever os requisitos através de cartões de histórias como os usados no modelo da programação extrema. Essa escolha se deve ao fato de ser mais leve, não gerando uma documentação excessiva de requisitos e por facilitar também a gerência de requisitos especificada na tarefa três. Os cartões de história serão de papel e servirão para os *stakeholders* alunos e professores descreverem as funcionalidades. Eles serão escritos a lápis em um primeiro momento e, após isso, serão passados para o meio digital. Como sugerido por Cohn (2006), serão usados

cartões que explicitem o tipo de usuário e o valor de negócio que a funcionalidade traz segundo a seguinte estrutura:

Como um <usuário/papel>, eu gostaria de <funcionalidade> para <benefício/valor de negócio>

O cartão de história também conterá outras informações referentes às funcionalidades: estimativas e o número de horas gastas na implementação. O layout do cartão usado no projeto será um dos apresentados por Sharp e Robinson (2007) e que pode ser visto na Figura 16.

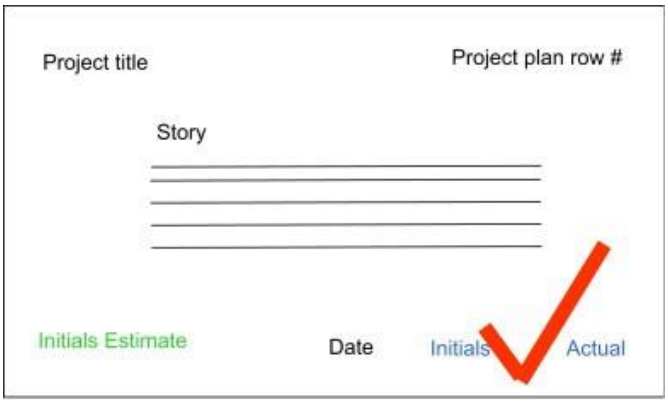

Figura 16: Layout do cartão XP (retirado de Sharp e Robinson, 2007)

4.2.3 Tarefa três

É necessário especificar um mecanismo de gerência dos requisitos e priorização dos requisitos. Os requisitos serão gerenciados através de *backlogs* como no modelo scrum e também no próprio modelo SEMAT. Serão usados dois *backlogs*: *backlog* de produto e *backlog* de iteração. O *backlog* de produto conterá todos os requisitos iniciais levantados com os *stakeholders*, enquanto o *backlog* de iteração têm os requisitos selecionados para serem implementados em uma iteração do desenvolvimento do sistema de software.

Para priorizar os requisitos, que serão colocados no *backlog* da iteração, será usado o jogo do planejamento do modelo programação extrema para que haja a participação efetiva dos grupos de *stakeholders* alunos e professores no processo.

4.2.4 Tarefa quatro e cinco

Para o projeto em questão, foram identificadas as seguintes restrições:

- Não há financiamento da instituição para o seu desenvolvimento;
- A instituição não fornecerá um ambiente de software e hardware durante o desenvolvimento;
- A implantação efetiva do sistema de software na instituição depende da submissão e aprovação de um projeto para a administração central a qual ela é submetida;
- A administração central é a responsável pela manutenção e suporte do LMS Moodle na instituição e
- Há a preferência pelo uso de soluções *open source* ou que não promovam gastos em licenças de software.

Já as premissas do projeto são:

- A desenvolvedora do projeto acumula também a posição de professora da disciplina de banco de dados na instituição e
- A versão usada no LMS Moodle é a última disponível no *site*.

4.2.5 Tarefas seis, sete e oito

As três tarefas do alfa sistema de software estão interligadas entre si. O sistema de software será desenvolvido com uma arquitetura modular, pois ele se integrará ao software para *e-learning* usado na instituição, ou seja, serão desenvolvidos *plugins* para a plataforma Moodle.

As plataformas de hardware necessárias para o sistema de software são aquelas exigidas pela plataforma Moodle. São recomendadas através de Moodle (2012) as seguintes características:

- Espaço em disco: No mínimo 160MB livre, além do espaço necessário para armazenamento dos materiais dos cursos. Um valor mínimo realista é de 5GB e
- Memória: No mínimo 256MB, sendo que 1GB ou mais é fortemente recomendado. A regra geral é que o Moodle pode suportar de 10 a 20 usuários concorrentes para cada 1GB de RAM, mas isso pode variar em cada tipo específico de hardware, combinação de software e o tipo de uso.

As linguagens de programação e as tecnologias usadas para o desenvolvimento do sistema de software são as mesmas do Moodle:

- Sistema operacional: Windows XP/2000/2003, Solaris 10 (Sparc e x64), Mac OS X ou Netware 6. O Linux e o Windows são as escolhas mais comuns (e que possuem um melhor suporte), sendo que o Linux é considerado a plataforma ideal;
- Servidor *web*: Apache ou ISS. Não foi totalmente testado (ou suportado), mas deve funcionar com o lightttpd, o nginx, o cherokee, o zeus e o LiteSpeed. A versão do servidor *web* não é considerada um aspecto crítico, mas é recomendável usar a última disponível;
- Linguagem PHP: A versão mínima a ser usada deve ser a 5.3.2. São necessárias diversas extensões do PHP; se alguma delas faltar, a instalação do Moodle irá parar durante a checagem do ambiente;
- Sistema de gerência de banco de dados: O MySQL e o PostgreSQL são aqueles mais testados e com uma documentação e um suporte mais extensos. O Oracle e o MSSQL são completamente suportados, mas a documentação e a ajuda *online* não são tão boas quanto as do MySQL e PostgreSQL. Para cada um dos SGBDs, as versões requeridas são: MySQL (versão mínima 5.1.33), PostgreSQL (versão mínima 8.3), MSSQL (versão mínima 9.0), Oracle (versão mínima 10.2) e SQLite (versão mínima 2.0) e
- Navegador *web*: As versões mínimas para acessar o Moodle são Firefox 4, Internet Explorer 8, Safari 5, Google Chrome 11, Opera 9; todos eles com

plugins e aplicações que serão necessárias para exibir os conteúdos dos cursos como, por exemplo, o Flash Player e o Java.

4.2.6 Resultado da segunda iteração

Foi feita a análise dos estados atuais dos alfas escolhidos para serem trabalhados durante essa iteração através da *checklist* do modelo SEMAT. Abaixo será detalhado como foram atingidos cada um dos estados alvo desses alfas.

4.2.6.1 *Stakeholders*

O estado alvo para esse alfa era o de representados. Ele foi plenamente atingido após essa iteração, tendo em vista que os *stakeholders* concordaram em assumir suas responsabilidades e representantes deles foram autorizados a fazer isso. Os coordenadores e a direção autorizaram a continuidade do trabalho com os alunos. Já alguns alunos se voluntariaram a participar do levantamento de requisitos e serem consultados sempre que necessário durante o processo de desenvolvimento. A abordagem de colaboração entre os representantes dos *stakeholders* foi acordada e eles suportam e respeitam o modo de trabalho da equipe.

4.2.6.2 Oportunidade

Para o alfa oportunidade, o estado alvo era o de solução necessária. Segundo a *checklist* desse estado, ele foi plenamente atingido na iteração, uma vez que os *stakeholders* na oportunidade e a solução proposta foram identificados. As necessidades que geraram a oportunidade, além dos problemas e suas causas profundas foram identificadas, sendo elas:

- A continuidade das atividades práticas em linguagem SQL em diferentes momentos do curso;

- A dificuldade de uso dos SGBDs devido à interface ser toda em inglês, além dos erros e *feedbacks* também serem nessa língua;
- A conexão da internet da instituição não ser adequada à demanda atual e
- A dificuldade de avaliação das atividades práticas realizadas com os SGBDs.

Através das reuniões feitas, foi confirmado junto aos *stakeholders* que uma solução baseada em software é necessária. Também foi proposta uma solução e apresentada aos *stakeholders*, sendo que esta é um sistema de software para o ensino da linguagem SQL que seja integrado ao LMS Moodle.

4.2.6.3 Requisitos

Ao final dessa iteração, os requisitos deveriam ser considerados delimitados. Esse estado foi plenamente atingido, uma vez que os *stakeholders* envolvidos no desenvolvimento já foram identificados, concordam com o propósito do sistema de software e têm um entendimento compartilhado da extensão da solução proposta. Isso foi conseguido através de reuniões com os *stakeholders*.

Para atingir esse estado, também é necessário que a forma como os requisitos serão descritos, os mecanismos para gerenciá-los e seu esquema de priorização estejam definidos. Os requisitos serão descritos por cartões de histórias, gerenciados através de *backlogs* e priorizados através do jogo do planejamento.

Ainda, as restrições estão identificadas e serão consideradas, bem como as premissas estão claramente definidas como exposto nas tarefas quatro e cinco.

4.2.6.4 Sistema de software

No caso do alfa sistema de software, o estado alvo era o de arquitetura selecionada. Da iteração anterior ficou faltando identificar as plataformas de hardware, selecionar as linguagens de programação e tecnologias a serem usadas, conhecer os limites do sistema e tomar decisões significativas sobre a organização do sistema e sobre

compras, construção e reuso. Esse estado foi plenamente atingido ao fim da segunda iteração.

Usando os critérios acordados para selecionar a arquitetura, foi possível tomar a decisão pelo o uso de uma arquitetura modular, com a construção de *plugins* para o LMS Moodle. Essa arquitetura possibilitará a integração com o sistema de software para *e-learning* usado na instituição que, no caso, é o Moodle, e também a extensão para o uso em diferentes tópicos dentro do ensino de banco de dados.

O Moodle é flexível, permitindo a integração com os SGBDs utilizados na instituição que são o MySQL e o MSSQL. O uso dessa plataforma como base para a solução de software permitirá o acesso *online* de banco de dados pelos alunos e a possibilidade de sincronização com dados *off-line*.

4.3 Terceira iteração

Os estados iniciais da terceira iteração para cada um dos alfas são: *stakeholders* (representados), oportunidade (solução necessária), requisitos (delimitados) e sistema de software (arquitetura selecionada). Já os estados que devem ser atingidos ao final dessa iteração são: *stakeholders* (envolvidos), oportunidade (valor estabelecido), requisitos (coerentes) e sistema de software (demonstrável).

Isso significa que ao fim dessa iteração, espera-se que:

- Os representantes dos *stakeholders* estejam envolvidos ativamente no trabalho e cumprindo suas responsabilidades;
- O valor de uma solução de sucesso tenha sido estabelecido;
- Os requisitos forneçam uma descrição consistente das características essenciais de um novo sistema de software e
- Uma versão executável do sistema esteja disponível, demonstrando que a arquitetura é aderente ao propósito e suporta testes.

As tarefas para essa iteração estão mostradas na Tabela 7 e foram estabelecidas a partir da *checklist* de cada um dos estados dos alfas.

Tabela 7: *Backlog* da iteração 3

Alfas	**Tarefas**
Stakeholders	
Oportunidade	Tarefa 1: Listagem dos critérios de sucesso e dos resultados desejados necessários a solução
Requisitos	Tarefa 2: Jogo do planejamento para levantamento de requisitos
Sistema de Software	Tarefa 3: Instalação da plataforma Moodle

Nos tópicos a seguir, é feita a descrição de como cada tarefa foi cumprida.

4.3.1 Tarefa um

Para avaliar e validar a solução de software após a sua implementação e implantação, foram listados alguns critérios de sucesso pelos quais a implantação do sistema de software será julgada, bem como os resultados desejados necessários a ele. Esses critérios e resultados esperados foram pensados pela autora e desenvolvedora a partir de sua experiência no ensino de banco de dados e da análise de outros sistemas de software da literatura:

- Facilidade de uso da interface pelos professores;
- Facilidade de uso da interface pelos alunos;
- Facilidade de configuração do sistema de software como *plugin* no Moodle pelos professores;
- Desempenho satisfatório quando estiver em uso;
- Disponibilidade de acesso *online* pelos alunos e professores;

- Aceitação da solução pela coordenação e pela direção da instituição;
- Disponibilização da solução para a comunidade (tanto de outras unidades quanto para a comunidade do Moodle);
- Uso efetivo pelos professores e pelos alunos em aula;
- Melhoria do desempenho escolar dos alunos no tópico relativo à linguagem SQL e
- Requisitos levantados implementados corretamente.

4.3.2 Tarefa dois

Para realizar o levantamento de requisitos, foi escolhido fazer o jogo do planejamento usado na programação extrema para escrever cartões de história com os *stakeholders* desenvolvedores, professores e alunos. As histórias foram escritas conforme o formato definido na iteração anterior:

- Como aluno, eu gostaria de visualizar uma lista de exercícios sobre a linguagem SQL para que possa treinar os comandos aprendidos em aula.
- Como aluno, eu gostaria de visualizar uma lista de exercícios sobre a linguagem SQL separada por comando para que possa treinar os comandos aprendidos em aula.
- Como aluno, eu gostaria de visualizar uma lista de exercícios sobre a linguagem SQL separada por nível de dificuldade para que possa treinar os comandos aprendidos em aula.
- Como aluno, eu gostaria de selecionar um exercício da lista para poder resolvê-lo para que possa treinar os comandos aprendidos em sala de aula.
- Como aluno, eu gostaria de digitar uma *query* SQL como solução de um determinado exercício e depois executá-la em um SGBD conhecido para testar meus conhecimentos na linguagem SQL.

- Como aluno, eu gostaria de receber *feedbacks* dos erros em uma *query* SQL executada em um determinado exercício na língua portuguesa para que possa entender o meu erro e consertá-lo.
- Como aluno, eu gostaria de ver os *feedbacks* dos erros na *query* SQL executada em um determinado exercício na língua original do SGBD para que possa me acostumar com as mensagens originais do SGBD usado.
- Como aluno, eu gostaria de fazer um teste livre de qualquer *query* SQL, podendo escolher em qual SGBD será executado, para que possa treinar os comandos aprendidos em aula.
- Como aluno, eu gostaria de ver os erros na *query* SQL destacados com outra cor para perceber exatamente onde errei.
- Como aluno, eu gostaria de fazer *backup* dos bancos de dados criados nos exercícios para poder utilizar em outros sistemas ou para ter uma cópia de segurança.
- Como aluno, eu gostaria de importar um banco de dados criado em outro sistema ou até mesmo um *backup* gerado pelo próprio sistema para que possa recuperar os meus dados, caso necessário.
- Como aluno, eu gostaria de ter acesso a dicas de como corrigir uma *query* SQL quando digitá-la de forma incorreta para que possa perceber onde errei.
- Como aluno, eu gostaria de acessar ao histórico de tentativas de *queries* SQL em um determinado exercício para que eu possa ver a minha evolução.
- Como professor, eu gostaria de inserir o enunciado de um novo exercício para os alunos, podendo escolher sobre qual comando SQL versa, qual o nível de dificuldade, em qual SGBD dos suportados pelo sistema será executado e qual a forma de pontuação, para que os alunos possam treinar os comandos ensinados em aula.
- Como professor, eu gostaria de editar o enunciado de um exercício para que possa corrigir erros por ventura inseridos na hora da inserção dele ou para melhorar a explicação.

- Como professor, eu gostaria de excluir o enunciado de um exercício para que possa apagar aqueles que não serão mais usados.
- Como professor, eu gostaria de ter acesso ao histórico de tentativas de um determinado aluno em um exercício para que possa avaliar a sua evolução no conteúdo ensinado.
- Como professor, eu gostaria de enviar um *feedback* a um aluno específico sobre um determinado exercício para que ele possa ter um retorno de como está sendo seu processo de aprendizagem.
- Como professor, eu gostaria de fazer uma cópia do enunciado de um exercício para que, depois, possa editá-lo e criar, por exemplo, um exercício com o mesmo enunciado, porém usando outro SGBD.
- Como professor, eu gostaria de fazer *backup* dos bancos de dados dos alunos para ter uma cópia de segurança ou ainda para fazer uma avaliação detalhada do que criaram.
- Como professor, eu gostaria de importar um banco de dados criado em outro sistema ou até mesmo um *backup* gerado pelo próprio sistema e disponibilizá-lo para os alunos para que possa recuperar dados caso necessário ou usá-lo em um exercício para os alunos.
- Como professor, eu gostaria de visualizar uma lista de quais alunos conseguiram terminar quais exercícios sobre linguagem SQL para que possa ter uma avaliação da turma como um todo.
- Como professor, eu gostaria de configurar o *plugin* de linguagem SQL para Moodle de forma fácil, informando configurações como usuário e senha para cada SGBD suportado pelo sistema para que possa utilizá-lo rapidamente com os alunos.

Como será usado o Moodle como base para o desenvolvimento do sistema de software, os requisitos ligados ao cadastro de alunos, professores e turmas não foram especificados, pois não precisarão ser implementados.

4.3.3 Tarefa três

Para entender parcialmente as configurações críticas de hardware e a integração com outros sistemas do software a ser desenvolvido foi instalada a plataforma Moodle. Essa instalação foi feita em um servidor compartilhado com o sistema operacional Linux Debian. A versão da plataforma Moodle instalada é 1.9.10. Para isso, foram usados o PHP 5.2.17 e o MySQL 5.0.51a.

Essa versão de Moodle já vem com diversos módulos por padrão e cada um deles representa uma pasta no servidor. Através dessa tarefa foi possível entender parcialmente como é a arquitetura do Moodle e como integrar um novo módulo a ele. Além disso, pode-se perceber que é possível usar um servidor compartilhado para o sistema de software proposto.

4.3.4 Resultados da terceira iteração

Através da *checklist* do modelo SEMAT foi feita a análise dos estados atuais dos alfas escolhidos para serem trabalhados durante essa iteração. Abaixo será detalhado como foram atingidos cada um dos estados alvo desses alfas.

4.3.4.1 *Stakeholders*

O objetivo dessa iteração era atingir o estado de envolvidos nesse alfa, o que foi plenamente atingido. Os representantes dos alunos deram assistência à equipe de acordo com suas responsabilidades nas tarefas executadas nessa iteração, fornecendo *feedback* e participando da tomada de decisão em tempo hábil.

À medida que as decisões foram sendo tomadas, os representantes dos alunos imediatamente foram comunicando as mais relevantes para o grupo que representam.

4.3.4.2 Oportunidade

Para o alfa oportunidade, o estado alvo era o de valor estabelecido, sendo que ele foi plenamente atingido. Segundo a *checklist* do SEMAT, o valor da oportunidade deve ser quantificado em termos absolutos ou em retorno ou economia por período de tempo (por exemplo, por ano). Como é um sistema de software voltado para educação, não há como medir diretamente o retorno de sua implantação. Espera-se que ele apoie e facilite o processo de ensino-aprendizagem da linguagem SQL, visando uma melhor colocação no mercado de trabalho dos alunos da instituição.

O impacto da solução sobre os *stakeholders* e seu valor são compreendidos. Para os alunos e professores, o processo de ensino-aprendizagem da linguagem SQL será facilitado através de seu uso. Já para os coordenadores e direção terão uma solução diferenciada para oferecer a seus alunos, além de proporcionarem condições motivadoras de trabalho para sua equipe de professores.

Na tarefa um da iteração, foi desenvolvida uma listagem de critérios de sucesso e de resultados desejados pela qual o sistema de software será julgado.

4.3.4.3 Requisitos

Os requisitos deveriam estar coerentes ao final da terceira iteração, o que foi plenamente atingido. Os requisitos foram capturados e compartilhados com a equipe e os *stakeholders* através dos cartões de história transcritos na tarefa dois. A origem dos requisitos é clara através da identificação dos *stakeholders* nesses cartões. Os requisitos descritos nas histórias comunicam as características essenciais do sistema a ser entregue e, através deles, os cenários de uso mais importantes para o sistema podem ser explicados. O impacto de implementar os requisitos é conhecido como foi analisado no item do alfa oportunidade. Também está clara a lógica por trás dos requisitos. A desenvolvedora do projeto em questão entendeu e concordou com o que deve entregar.

Também se deve identificar requisitos conflitantes e atendê-los. Isso foi sendo feito através de acordos durante o próprio jogo do planejamento. Um conflito que ocorreu foi no caso do *feedback* dos erros em um exercício. Os alunos gostariam que, quando errassem, o sistema lhes mostrasse direto qual seria a *query* correta. Porém, os professores achavam que pedagogicamente não seria a melhor opção. Assim, foram pensadas alternativas que atendessem as expectativas de ambos os *stakeholders*: dicas de sintaxe dos comandos SQL, destaque em outra cor da parte errada da *query* e *feedbacks* dos erros em língua portuguesa.

A prioridade dos requisitos é clara. Primeiramente, devem ser implementados os requisitos de cadastro de enunciados dos exercícios e os de resolução dos exercícios. Depois, os demais devem ser priorizados ao longo das iterações de acordo com o jogo do planejamento com as histórias restantes.

4.3.4.4 Sistema de software

No caso do alfa sistema de software, o estado alvo era de demonstrável. Esse estado não foi atingido, pois como os requisitos ainda estavam sendo levantados, não foi possível iniciar a implementação. Com isso, não foi possível ter:

- As principais características da arquitetura demonstradas;
- O uso do sistema e uma medição de seu desempenho;
- Uma demonstração das configurações críticas de hardware;
- Uma demonstração das interfaces críticas;
- Uma demonstração da integração com outros sistemas existentes e
- Um acordo entre os *stakeholders* relevantes de que a arquitetura demonstrada é apropriada.

A única tarefa relacionada ao alfa sistema de software executada foi a instalação de uma plataforma Moodle, o que demonstra parcialmente as configurações críticas de hardware e a integração com outros sistemas. O sistema de software a ser

desenvolvido será um *plugin* para o Moodle, logo a integração com outros sistemas em questão é essa.

4.4 Quarta iteração

Os estados iniciais dessa iteração são os atingidos na iteração anterior: *stakeholders* (envolvidos), oportunidade (valor estabelecido) e requisitos (coerentes). Os estados alvo para os alfas são: *stakeholders* (em acordo), oportunidade (viável) e requisitos (aceitável). O estado de demonstrável do alfa sistema de software não foi plenamente atingido anteriormente, continuando a ser o estado alvo dessa iteração. Ao final dessa iteração, os representantes dos *stakeholders* devem estar em acordo que a solução pode ser produzida rapidamente e barata o suficiente para contemplar a oportunidade com sucesso e os requisitos devem descrever um sistema que é aceitável para os *stakeholders*.

A partir da *checklist* de cada um dos estados foram pensadas as tarefas necessárias para se atingir o estado de cada um dos alfas. As tarefas estão mostradas na Tabela 8.

Tabela 8: *Backlog* da iteração 4

Alfas	**Tarefas**
Stakeholders	Tarefa 2: Jogo do planejamento para definição da primeira iteração de desenvolvimento
Oportunidade	Tarefa 3: Análise de riscos associados com a solução Tarefa 4: Análise de custos da solução
Requisitos	
Sistema de Software	Tarefa 1: Desenvolvimento de um *plugin* de testes para o Moodle

A descrição de como cada tarefa foi cumprida é feita nos tópicos seguintes.

4.4.1 Tarefa um

Nessa tarefa foi desenvolvido um *plugin* de testes para o Moodle. A ideia foi explorar o funcionamento da arquitetura modular do Moodle, quais arquivos devem ser desenvolvidos, a estrutura de pastas, como acessar o banco de dados, além de como ativar um novo módulo no Moodle.

A conclusão dessa tarefa era um pré-requisito para executar a tarefa dois, pois a desenvolvedora conseguiu ter uma noção mais realista dos desafios na implementação de um *plugin* para o Moodle e esse conhecimento ajudará na estimativa do tempo a ser gasto para implementar uma determinada história.

4.4.2 Tarefa dois

Para construir o *backlog* de iteração, escolhendo assim as histórias que serão implementadas na primeira iteração de desenvolvimento do sistema de software, foi feito novamente o jogo do planejamento com os *stakeholders* desenvolvedores, professores e alunos.

Nesse segundo jogo do planejamento foi feita uma priorização das histórias escritas na iteração anterior e que estavam no *backlog* de produto. Depois, foram feitas estimativas de quanto tempo demoraria a implementar cada história, o que foi anotado nos cartões. Tendo em vista que cada iteração tem a duração de uma semana e as prioridades das histórias, foram selecionadas quatro histórias a serem desenvolvidas durante a quinta iteração (a primeira de desenvolvimento):

- Como professor, eu gostaria de inserir o enunciado de um novo exercício para os alunos, podendo escolher sobre qual comando SQL versa, qual o nível de dificuldade, em qual SGBD dos suportados pelo sistema será executado e

qual a forma de pontuação, para que os alunos possam treinar os comandos ensinados em aula.

- Como aluno, eu gostaria de visualizar uma lista de exercícios sobre a linguagem SQL para que possa treinar os comandos aprendidos em aula.
- Como aluno, eu gostaria de selecionar um exercício da lista para poder resolvê-lo para que possa treinar os comandos aprendidos em sala de aula.
- Como aluno, eu gostaria de digitar uma *query* SQL como solução de um determinado exercício e depois executá-la em um SGBD conhecido para testar meus conhecimentos na linguagem SQL.

As histórias devem ser desenvolvidas nessa ordem, uma vez que são interdependentes entre si.

4.4.3 Tarefa três

Para ter sucesso no desenvolvimento de qualquer sistema de software é necessário conhecer os riscos envolvidos e que eles sejam aceitáveis e gerenciáveis. Por isso, foi feito uma listagem com os principais riscos associados ao sistema em questão:

- Introdução de características interessantes, porém dispensáveis, durante os jogos de planejamento, traduzindo-se em histórias, ou também pelos desenvolvedores, durante a implementação;
- Aceitação descontrolada de novas histórias durante uma iteração, alterando o cronograma da iteração;
- Atrasos no cronograma das iterações devido a previsões excessivamente otimistas;
- Falta de patrocínio para o projeto;
- Falta de participação efetiva dos *stakeholders* durante o desenvolvimento do sistema de software;

- Atritos entre desenvolvedores e os demais *stakeholders*, causados por expectativas irreais destes e otimismo sem fundamento daqueles e
- Mudança de tecnologias e ferramentas de desenvolvimento no meio do projeto.

No momento de planejamento de cada iteração deve-se atentar a esses riscos, sempre atualizando essa lista e tomando atitudes para mitigá-los.

4.4.4 Tarefa quatro

Deve ser feita uma análise de custos indicativos da solução para verificar sua viabilidade. Como se optou desenvolver por iterações o software há uma dificuldade associada a essa análise, pois não se consegue prever com exatidão quantas iterações serão necessárias para desenvolver o sistema de software até o ponto de aceitação pelos *stakeholders*.

O que pode ser analisado é o que ocorreu até o momento. Foram executadas quatro iterações, sendo que cada uma delas com a duração de uma semana com dedicação de 20 horas por parte da desenvolvedora. Portanto, os custos da solução incluem a mão de obra da desenvolvedora, além de um servidor *web* compartilhado e de infraestrutura (custos com banda larga, eletricidade).

4.4.5 Resultado da quarta iteração

Foi feita a análise dos estados atuais dos alfas escolhidos para serem trabalhados durante essa iteração através da *checklist* do modelo SEMAT. Abaixo será detalhado como foram atingidos cada um dos estados alvo desses alfas.

4.4.5.1 *Stakeholders*

O estado alvo para esse alfa era o de em acordo. O estado foi plenamente atingido, uma vez que os representantes dos *stakeholders* fizeram um acordo quanto as suas expectativas mínimas para a próxima implantação do novo sistema (que ocorrerá após a primeira iteração de desenvolvimento), estão felizes com a sua participação no trabalho, além de concordarem que sua contribuição é valorizada pela equipe, que são tratados com respeito e com a forma como as suas prioridades e perspectivas diferentes estão sendo equilibradas para fornecer uma direção clara para a equipe.

Por outro lado, a desenvolvedora concorda que sua contribuição é valorizada pelos representantes dos *stakeholders* e que é também tratada com respeito. Pode-se perceber isso durante o jogo do planejamento que envolveu alunos, professores e a desenvolvedora. Houve um ambiente de colaboração entre todos os participantes, que participaram de bom grado, além de sentirem úteis no processo.

4.4.5.2 Oportunidade

Para o alfa oportunidade, o estado alvo era o de viável, que foi plenamente atingido. A solução está delineada através das histórias levantadas na iteração anterior e revisadas nessa. As razões para o desenvolvimento de uma solução baseada em software são compreendidas pela desenvolvedora, uma vez que ela também representa os *stakeholders* professores. As indicações são de que a solução pode ser desenvolvida e implantada dentro das restrições listadas na iteração anterior. Foi desenvolvida uma listagem com os riscos associados à solução e estes são aceitáveis e gerenciáveis.

Foi feita uma análise simplificada em relação aos custos indicativos da solução, já que o desenvolvimento de software por iterações não permite uma análise mais precisa. Porém, pode-se afirmar que os custos indicativos da solução são menores do que o valor esperado da oportunidade e está claro que a oportunidade é viável.

4.4.5.3 Requisitos

Ao final dessa iteração, os requisitos deveriam ser considerados aceitáveis. Esse estado foi plenamente atingido através da tarefa dois. Durante o jogo do planejamento, pode-se verificar que os *stakeholders* acham que os requisitos descrevem uma solução aceitável e que o valor fornecido pela implementação dos requisitos é claro, além de estarem claras quais partes da oportunidade estão satisfeitas por cada requisito.

A forma como os requisitos foram descritos em cartões de história permite que eles sejam testáveis após sua implementação. A taxa de mudanças para os requisitos acordados está sob controle devido à prática do replanejamento constante e do jogo do planejamento a cada iteração.

4.4.5.4 Sistema de software

O estado alvo para o alfa sistema de software era o de demonstrável. Esse estado foi parcialmente atingido, pois como os requisitos ainda não haviam sido priorizados e nem tinham sido feitas as estimativas de tempo para sua implementação, não foi possível iniciar a implementação efetiva do *plugin*.

A única tarefa relacionada ao alfa sistema de software executada foi a criação de um *plugin* de teste para o Moodle, o que demonstra parcialmente as configurações críticas de hardware, a integração com outros sistemas e as principais características da arquitetura. Os *stakeholders* relevantes (desenvolvedora) estão de acordo que a arquitetura demonstrada é apropriada. Porém, como não foi desenvolvido um módulo com uma das histórias levantadas, não foi possível demonstrar as interfaces críticas nem fazer uso e medição do desempenho.

5 AVALIAÇÃO E ANÁLISE DO USO DO SEMAT

Nesse item será apresentada uma análise feita a partir do estudo de caso aplicando o modelo SEMAT para o desenvolvimento de software para *e-learning* de banco de dados.

De forma sintética, o SEMAT possui sete alfas, que representam o conjunto mínimo de características presentes em qualquer esforço de engenharia de software. Cada um desses alfas possuem estados que representam a evolução do esforço naquela característica. Para cada estado de um alfa, existe uma *checklist* associada. As *checklists* são úteis na verificação se os estados dos alfas foram ou não alcançados em uma iteração. Além disso, elas também são úteis no momento do planejamento da iteração. Nesse momento, são escolhidos quais serão os estados alvo e, a partir das *checklists*, podem-se planejar quais as tarefas devem ser feitas para atingi-los ao fim da iteração.

Há uma clara interdependência entre os alfas, o que pode ser visto pelas referências feitas nas *checklists* de um estado e outros alfas. Por exemplo, em um item da *checklist* de um estado do alfa oportunidade é descrito sobre identificar os *stakeholders*, que é outro alfa. Esse recurso é interessante para criar uma visão global do esforço de engenharia de software e o que um alfa impacta e influencia em outro. Logo, as *checklists* são um mecanismo eficaz de controle da evolução do esforço de engenharia de software. Porém, apesar disso, alguns itens das *checklists* não apresentam diferenças claras entre si, sendo até redundantes ou não expressando claramente a diferença entre dois estados distintos.

Apesar das *checklists* evidenciarem se um estado do alfa foi alcançado ou não, essa análise é subjetiva e depende da visão do gerente do projeto, de sua equipe e dos *stakeholders* envolvidos. Portanto, está sujeita a vários fatores humanos, como a experiência e a maturidade da equipe no uso de modelos de engenharia de software, o nível de conhecimento do modelo SEMAT, o entrosamento da equipe e a própria interpretação do que significa cada item da *checklist*.

Jacobson et al. (2012) sugere o uso de cartões para representar os estados dos alfas. Nesses cartões, são mostrados: o nome do alfa, o nome do estado, seu número dentro de todo o conjunto de estados e um pequeno resumo da checklist.

Nesse estudo de caso, o método usado para determinar o estado atual do esforço foi o de *walkthrough*, fazendo a utilização dos cartões para isso. O uso dos cartões se mostrou interessante, pois ajudou a determinar as próximas tarefas durante o planejamento da iteração e também podiam ser consultados durante sua execução, ajudando a ter sempre claro o objetivo a ser alcançado. Além disso, os cartões serviram para avaliar se os estados alvos foram alcançados ao final da iteração.

Apesar do SEMAT possuir *checklists* dos estados dos alfas e a recomendação do uso de cartões para representá-los, ele não especifica como alcançar cada estado, ficando a cargo da equipe e dos *stakeholders* planejarem como fazer isso, quais serão as tarefas necessárias e quais serão as práticas usadas. Isso é interessante, pois permite uma flexibilidade para a adoção de práticas já usadas anteriormente pela equipe e também para a evolução do conjunto particular de práticas selecionadas pela equipe ao longo do tempo. Porém, isso pode ser problemático para grupos sem uma experiência prévia com algum modelo de engenharia de software.

No estudo de caso, optou-se por usar as práticas dos modelos ágeis programação extrema e scrum, como os cartões de história, o jogo do planejamento e os *backlogs*. Essas escolhas se devem à experiência prévia da autora nessas práticas e por serem práticas leves que não exigem a construção de documentos formais. Antecipou-se, porém, que isso poderia restringir a análise do modelo SEMAT, uma vez que o uso de práticas distintas de outros modelos poderia acarretar resultados diferentes no desenvolvimento do estudo de caso.

Também não foram explorados alguns conceitos do modelo SEMAT como os espaços de atividades e as competências, pois estes foram definidos posteriormente e ainda não há explicações ou exemplos de como utilizá-los na literatura existente.

Quanto às tarefas executadas durante as iterações, há uma observação importante a se fazer em relação aos questionários aplicados a alunos sobre as dificuldades encontradas no aprendizado de alguns tópicos de banco de dados. Podem-se

perceber algumas incoerências e leves desvios nas respostas. Alunos de turmas para as quais ainda não havia sido abordado determinado tópico pontuaram um nível de dificuldade em seu aprendizado quando na verdade deveriam escolher a alternativa que aquele tópico não havia sido ensinado. Também se percebeu na pergunta aberta do questionário algumas respostas com reclamações e sugestões sobre a instituição onde a pesquisa ocorreu e sobre suas disciplinas de banco de dados. Isso mostra que alguns alunos não possuem ainda maturidade suficiente para distinguir entre o que era pertinente à pesquisa realizada ou à instituição. Mas isso também se deve, em parte, pelo fato da pesquisadora estar vinculada à instituição de ensino como professora da disciplina de banco de dados.

Através do uso do SEMAT no estudo de caso, pode-se concluir que ele é um modelo de gestão de projetos de software. Ele pode ser considerado como uma camada de controle, podendo ser considerado acima dos demais modelos apresentados na fundamentação teórica desse trabalho, como é mostrada na Figura 17. Com isso, ele permite uma maior visibilidade do que está ocorrendo no projeto e um controle de qualidade durante todo o processo. Além disso, com essa camada de controle no projeto, é possível aplicar as práticas leves de modelos ágeis (como a programação extrema e scrum) para projetos maiores e com maior criticidade.

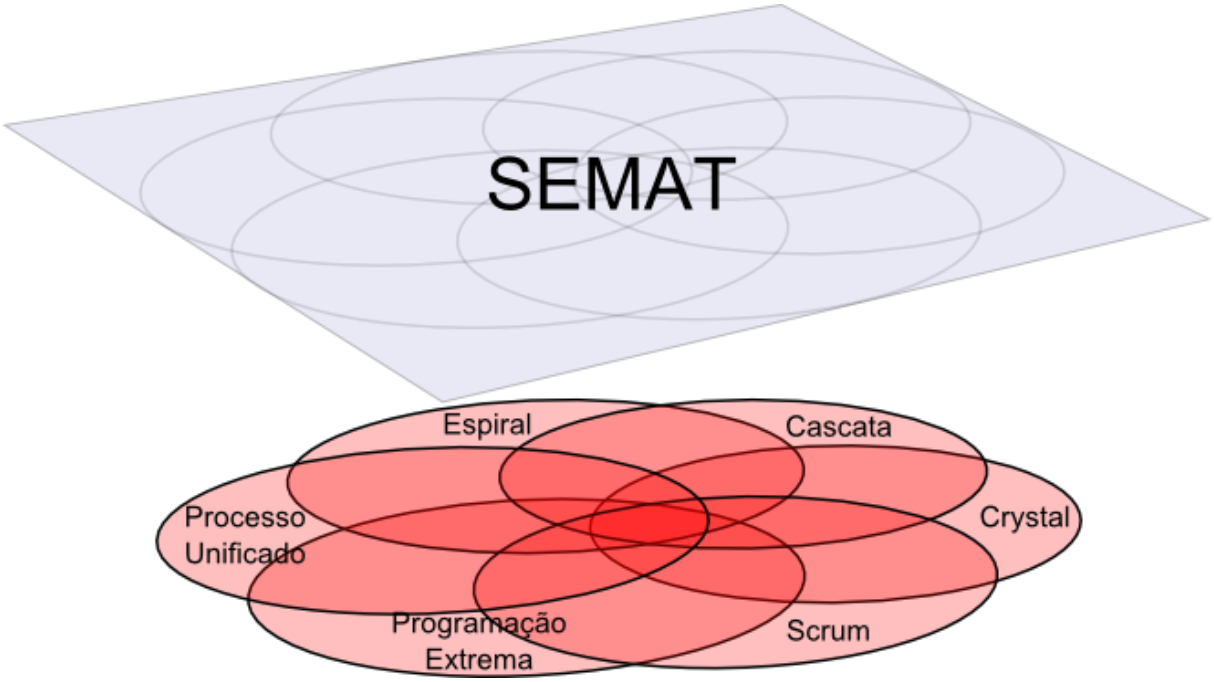

Figura 17: Camada de controle em projetos de engenharia de software

6 CONSIDERAÇÕES FINAIS

Para a conclusão do estudo de caso houve diversos desafios. Primeiramente, o modelo SEMAT ainda está em construção. A primeira versão do modelo foi disponibilizada em fevereiro de 2012 e, após isso, foram feitas algumas revisões que especificaram melhor algumas de suas características, sendo a última de novembro de 2012. Isso dificultou em partes a execução do estudo de caso, pois o modelo foi sendo modificado ao longo do tempo e algumas dúvidas foram sendo esclarecidas com as novas revisões.

Outra dificuldade é a de não haver muitos exemplos na literatura ou casos reais de uso do SEMAT, justamente pelo fato de ser um modelo em desenvolvimento e pouco conhecido. A referência que foi usada nesse trabalho é o livro que ainda está em fase de rascunho com previsão para ser lançado no início de 2013 (Jacobson et al., 2012). Por outro lado, esse desafio foi um dos maiores motivadores para o desenvolvimento deste trabalho, que poderá servir de base para outros projetos usando o modelo SEMAT no futuro.

Uma crítica que se pode fazer é quanto ao exemplo do livro sobre o SEMAT (Jacobson et al., 2012). Esse exemplo pretende mostrar como aplicar o modelo na prática, porém, considera apenas alguns alfas; os demais não são usados sem que os autores justifiquem o porquê. Um questionamento que fica é porque foram usados aqueles determinados alfas no exemplo. Outra dúvida é, sendo os alfas um conjunto mínimo de características de qualquer esforço de engenharia de software, essenciais segundo o próprio título do livro, porque apenas um subgrupo destes foi usado e não todo o conjunto.

No estudo de caso apresentado também foram usados apenas alguns alfas a exemplo do livro. Não foram usados os alfas que estão dentro da área de interesse da equipe, pois o foco desse trabalho é o levantamento de requisitos e também por não haver uma equipe completa de desenvolvimento, impossibilitando uma análise mais aprofundada nas questões de equipe e modo de trabalho. Porém, pode-se intuir que esta seja uma restrição do estudo de caso, uma vez que as tarefas

relacionadas com os alfas da equipe podem influenciar ainda que indiretamente nas demais.

Outra restrição identificada no estudo de caso foram os fatores humanos. A autora que fez a gestão do projeto apresentado possui uma pequena experiência em gestão de projetos de engenharia de software e não possuía nenhum conhecimento anterior no uso do SEMAT. Apesar disso, possuía experiência em desenvolvimento de software e nas práticas utilizadas da programação extrema e do scrum. Esses fatores podem também ter influenciado nos resultados obtidos.

O resultado obtido também é influenciado pelo fato do trabalho ter sido desenvolvido em ambiente acadêmico e para uma área de aplicação especifica, sendo que não é possível generalizar que determinadas práticas e decisões de projeto possam ser aplicadas a quaisquer contextos e outras áreas obtendo os mesmos resultados.

Por fim, com o estudo de caso, foi possível verificar que o modelo SEMAT é aplicável para a concepção e o levantamento de requisitos de um software para *e-learning*. Além disso, foram discutidas as restrições e adaptações necessárias para essa área de aplicação específica nesse trabalho. Também, como resultado do estudo de caso, foi obtida a especificação de um software para *e-learning* para ensino de banco de dados.

Como trabalho futuro, vislumbra-se a aplicação de todos os alfas do modelo SEMAT no desenvolvimento completo de um sistema de software, em especial para a análise crítica daqueles referentes à área de interesse da equipe. Também seria interessante a comparação desse estudo de caso com outros que usem práticas de diferentes modelos de engenharia de software para a execução das tarefas de cada iteração.

REFERÊNCIAS

ACKOFF, R. L. **Creating the corporate future: plan or be planned for**. Nova York: John Wiley and Sons, 1981, 297p. ISBN: 9780471090090.

AMERICAN ENGINEERS' COUNCIL FOR PROFESSIONAL DEVELOPMENT. **Canons of ethics for engineers**. Nova York: Engineers' Council for Professional Development, 1947.

BASSI, D. L. F.; **Experiências com desenvolvimento ágil**. São Paulo, 2008. Dissertação (Mestrado) - Instituto de Matemática e Estatística, Universidade de São Paulo.

BECK, K.; ANDRES, C. **Extreme Programming Explained: Embrace Change** (2nd Edition). Stoughton: Addison-Wesley Professional, 2004. ISBN 0321278658.

BECK, K. et al. **Manifesto for Agile Software Development**. 2001. Disponível em: <http://agilemanifesto.org/>. Acesso em: 01 set 2012.

BERTALANFFY, L. von. **General System Theory: Foundations, Development, Applications**. New York: George Braziller, Revised Edition, 1976, 295 p., fifteenth paperback printing, 2006. ISBN: 978-0-8076-0453-3.

BOLT, A.; MAZUR, G. H. **Jurassic QFD: Integrating Service and Product Quality Function Deployment**. In: The Eleventh Symposium on Quality Function Deployment. June 14-15, 1999, Novi, Michigan, USA.

BRUSILOVSKY, P.; SOSNOVSKY, S; YUDELSON, M. V.; LEE, D. H.; ZADOROZHNY, V.; ZHOU, X. **An Open Integrated Exploratorium for Database Courses**. In: Proceedings of the 13th annual conference on Innovation and technology in computer science education. June 30 - July 2, 2008, Madrid, Spain.

BRUSILOVSKY, P.; SOSNOVSKY, S; YUDELSON, M. V.; LEE, D. H.; ZADOROZHNY, V.; ZHOU, X. **Learning SQL Programming with Interactive Tools: From Integration to Personalization**. In: ACM Transactions on Computing Education, Vol. 9, No. 4, Article 19, Pub. date: January 2010.

CHAPANIS, A. **Human Factors in Systems Engineering.** New York: John Wiley & Sons, 1996. 352p. ISBN: 0471137820.

CHECKLAND, P. **Systems Thinking, Systems Practice**, includes a 30-years retrospective, John Wiley & Sons, New York, 1999.

CHECKLAND, P. **Soft Systems Methodology: A Thirty Year Retrospective**. System Research and Behavioral Science, v. 17, p. S11-S58, 2000.

COCKBURN, A. **Agile Software Development**. Boston: Addison-Wesley, 2002. 278p. ISBN 978-0201699692.

COHN, M. **Agile estimating and planning**. Prentice Hall, 2006.

HAREZLAK, K.; WERNER, A. **E-learning database course with usage of interactive database querying**. In: Internet – Technical Development and Applications Series, AISC, vol. 64, p. 81-89, 2009. ISBN: 978-3-642-05018-3.

HAREZLAK, K.; WERNER, A. **Database Access and Management with the Use of the Moodle Platform**. In: Human - Computer Systems Interaction, AISC 98, Part I, p. 49–65, 2012. Berlin: Springer.

HEUSER, C. A. **Projeto de Banco de Dados**. Porto Alegre: Sagra Luzzattto, 2004, 236 p. ISBN: 8524105909.

HITCHINS, D. K. **Systems Engineering: A 21st Century Systems Methodology**. Chichester: John Wiley & Sons, 2008. 528p. ISBN 9780470058565.

HITCHINS, D. K. **Basic Models for System Thinking**. Disponível em: <http://www.hitchins.net/SysMods.html>. Acesso em: 15 de agosto de 2012.

HORTON, W. **E-Learning by Design**. San Francisco: Pfeiffer, 2006. 640p. ISBN 9780787984250.

HSIANG-JUI, K.; HUI-LIEN, T. **A web-based tool to enhance teaching/learning database normalization**. In: Proceedings of the 2006 Southern Association for Information Systems Conference. March 11 - 12, 2006, Florida, USA.

IEEE, **IEEE Standard Glossary of Software Engineering Terminology**. IEEE Standard 610.12-1990, New York, 1990. 84p.

INEP - Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira. **Censo da educação básica: 2011 – resumo técnico**, 2012. Disponível em: < http://download.inep.gov.br/educacao_basica/censo_escolar/resumos_tecnicos/resumo_tecnico_censo_educacao_basica_2011.pdf >. Acesso em: 04 fev. 2013.

JACOBSON, I.; NG, P.; MCMAHON, P.; SPENCE, I.; LIDMAN, S. **The Essence of Software Engineering: Applying the SEMAT Kernel (Rough Cuts)**. Addison-Wesley Professional, 2012, 250p. ISBN 9780321885951

KENNY, C.; PAHL, C. **Automated Tutoring for a Database Skills Training Environment**. In: Technical Symposium on Computer Science Education. February 23–27, 2005, St. Louis, Missouri, USA.

KOTONYA, G.; SOMMERVILLE, I. **Requirements Engineering: Processes and Techniques**. London: J. Wiley, 1998, 294p. ISBN 9780471972082.

KOURI, M. G.; **Definição de requisitos para um sistema de monitoramento de veículos no transporte rodoviário de cargas**. São Paulo, 2007. Dissertação (Mestrado) - Escola Politécnica, Universidade de São Paulo.

LARMAN, C.; BASILI, V. R. **Iterative and incremental development: A brief history**. Computer, v. 36, n. 6, p. 47–56, 2003. ISSN 0018-9162.

LEE, S. M.; PRITCHETT, A. R. **Predicting Interations Between Agents in Agent-Based Modeling and Simulation of Sociotechnical Systems**, IEEE Transactions on systems, man and cybernetics – part A: Systems and humans, vol. 38, n. 6, 2008.

LOCAWEB. **Blog da Locaweb**. Disponível em: <http://blog.locaweb.com.br/metodologias-ageis/programacao-extrema-extreme-programming-ou-simplesmente-xp/>. Acesso em: 12 dez. 2011.

MACIASZEK, L. A. **Requirements Analysis and System Design: Developing Information Systems with UML**. Harlow: Addison-Wesley, 2001, 378p. ISBN 978-0201709445.

MARTINAZZO, A. A. G.; **Considerações sobre desenvolvimento colaborativo de software para aprendizagem em plataformas móveis**. São Paulo, 2011. Dissertação (Mestrado) - Escola Politécnica, Universidade de São Paulo.

MATÉ, J. L.; SILVA, A. **Requirements engineering for sociotechnical systems**. London: Information Science Publishing, 2005, 373p. ISBN 9781591405078.

MITROVIC, A.; HAUSLER, K. **An Intelligent SQL Tutor on the Web**. International Journal of Artificial Intelligence in Education, v. 13, p. 171–195, 2003. ISSN 1560-4292.

MOODLE. **Moodle 2.0 language packs**. Disponível em: <http://download.moodle.org/langpack/2.0/>. Acesso em: 21 jan. 2012.

MOODLE. **Installing Moodle**. Disponível em: <http://docs.moodle.org/24/en/Installing_Moodle>. Acesso em: 24 dez. 2012.

MOSS, K. **Database Teaching tools**. In: IEEE EDUCON Education Engineering 2010 – The Future of Global Learning Engineering Education. April 14 - 16, 2010, Madrid, Spain.

NEMETH, C. P. **Human Factors Methods for Design: Making Systems Human-centered**. Boca Raton: CRC Press. 2004. 392p. ISBN: 0415297982.

NG, P.; MAGEE, M. **Light Weight Application Lifecycle Management Using State-Cards**, 2010. Disponível em: <http://www.agilejournal.com/articles/columns/column-articles/3325-light-weight-application-lifecycle-management-using-state-cards>. Acesso em: 04 set. 2012.

PAHL, C.; BARRETT, R.; KENNY, C. **Supporting Active Database Learning and Training through Interactive Multimedia**. In: Conference on Innovation and Technology in Computer Science Education. June 28 - 30, 2004, Leeds, United Kingdom.

PISA - Programme for International Student Assessment. **PISA Country Profiles**. Disponível em: <http://pisacountry.acer.edu.au/>. Acesso em: 10 dez. 2010.

PRESSMAN, R. S. **Engenharia de Software**. São Paulo: McGraw-Hill, 2006, 720 p. ISBN: 8586804576.

ROSSON, M. B.; CARROLL, J. M. **Usability Engineering: Scenario-Based Development of Human Computer Interaction**. San Francisco: Morgan Kaufmann, 2001. 448p. ISBN 1558607129.

SAKAI. **The Sakai Community**. Disponível em: <http://sakaiproject.org/community-home>. Acesso em: 21 jan. 2012.

SANCHES, F. L. L.; **Engenharia de Sistemas em Redes Eusociais**. São Paulo, 2010. Dissertação (Mestrado) - Escola Politécnica, Universidade de São Paulo.

SANDOM, C.; **Human Factors for Engineers**. London: Institution of Electrical Engineers. 2004. 361p. ISBN: 0863413293.

SATO, D. T.; **Uso eficaz de métricas em métodos ágeis de desenvolvimento de software**. São Paulo, 2007. Dissertação (Mestrado) - Instituto de Matemática e Estatística, Universidade de São Paulo.

SHARP, H.; ROBINSON, H. **Collaboration and co-ordination in mature eXtreme programming teams**. International Journal of Human-Computer Studies, 2007. Disponível em <http://dx.doi.org/doi:10.1016/j.ijhcs.2007.10.004 >.

SEMAT – Software Engineering Method and Theory. **Call for Action**. Disponível em: <http://semat.org/?page_id=2>. Acesso em: 03 set. 2012.

SEMAT - Software Engineering Method and Theory. **Essence – Kernel and Language for Software Engineering Methods**, 2012a. Disponível em: <http://semat.org/wp-content/uploads/2012/02/12-08-15.pdf>. Acesso em: 03 set. 2012.

SIMONETTE, M. J.; **Engenharia de Sistemas em Sistemas Sociotécnicos**. São Paulo, 2010. Dissertação (Mestrado) - Escola Politécnica, Universidade de São Paulo.

SOARES, J. O. P.; **Especificação de Métodos de Desenvolvimento de Sistemas - Aplicação a Sistemas de Tempo Real**. São Paulo, 1986. Dissertação (Mestrado) - Escola Politécnica, Universidade de São Paulo.

SOMMERVILLE, I. **Software Engineering**. Boston: Addison-Wiley, 2007, 840 p. ISBN: 9780321313799.

SWEBOK. **Guide to the Software Engineering Body of Knowledge (SWEBOK)**. Disponível em: <http://www.computer.org/portal/web/swebok/html/ch1>. Acesso em: 28 dez. 2011 .

TORI, R. **Educação sem Distância: As Tecnologias Interativas na Redução de Distâncias em Ensino e Aprendizagem**. São Paulo: Editora SENAC São Paulo e Escola do Futuro da USP, 2010, 256 p. ISBN: 9788573599213.

TUPAROVA, D.; TUPAROV, G. **Didactical Issues of E-learning: Problems and Future Trends**. International Conference on Computer Systems and Technologies – CompSysTech, v. IV. 12-1-6, 2005.

ZANDI, I. **Science and engineering in the age of systems**, presented at What is System Engineering?, Sept 19 2000. International Council on System Engineering (INCOSE). Disponível em: <http://www.incose.org/delvalley/Zandi_Paper.pdf>. Acesso em: 16 nov. 2011.

REFERÊNCIAS CONSULTADAS

DELL'AQUILA, C.; LEFONS, E.; TANGORRA, F. **Design of an e-learning Environment for Teaching Databases and Information Systems**. In: 5th WSEAS / IASME International Conference on ENGINEERING EDUCATION. July 22 - 24, 2008, Heraklion, Greece.

MORENO, L.; IGLESIAS, A. M.; CASTRO, E.; MARTINEZ, P. **Using Accessible Digital Resources for Teaching Database Design: Towards an inclusive distance learning proposal**. In: ItiCSE'08. June 30 – July 2, 2008, Madrid, Spain.

SHIHONG, H.; **Capturing the essence of software engineering - An academia perspective of SEMAT**. In: JARAMILLO, C. M. Z. et al (org). Software Engineering: Methods, Modeling and Teaching. Bogotá: Universidad de Medellín, 2011, 326 p. ISBN: 9789588692326.

VENANCIO, V. **SEEALL: sistema interativo de apoio ao planejamento e acompanhamento de aprendizagem baseado em competências**. São Paulo, 2011. Dissertação (Mestrado) - Escola Politécnica, Universidade de São Paulo.

APÊNDICES

APÊNDICE A - Questionário de pesquisa de perfil de usuário

Sexo: [ ] Feminino [ ] Masculino	
Idade:	Turma:

1. Desde que idade você usa **computadores**? [ ] não uso/não lembro. [ ] desde ___ anos.

2. Com que frequência você usa a **Internet** nesses locais?

	Nunca	De vez em quando	1 a 3 vezes por semana	Todos os dias
Em casa	[ ]	[ ]	[ ]	[ ]
Na escola	[ ]	[ ]	[ ]	[ ]
No trabalho	[ ]	[ ]	[ ]	[ ]
Na lan-house	[ ]	[ ]	[ ]	[ ]
Em espaços públicos (telecentros, bibliotecas)	[ ]	[ ]	[ ]	[ ]
Outros. Quais? _____	[ ]	[ ]	[ ]	[ ]

3. Com que frequência você utiliza os aparelhos abaixo para acessar a **Internet**?

	Nunca	De vez em quando	1 a 3 vezes por semana	Todos os dias
Computador	[ ]	[ ]	[ ]	[ ]
Notebook / Laptop	[ ]	[ ]	[ ]	[ ]
Celular/Smartphone/ Palmtop	[ ]	[ ]	[ ]	[ ]
Outros. Quais? _____	[ ]	[ ]	[ ]	[ ]

4. Com que frequência você acessa esses serviços na **Internet**?

	Nunca	De vez em quando	1 a 3 vezes por semana	Todos os dias
E-mail	[ ]	[ ]	[ ]	[ ]
Jogos	[ ]	[ ]	[ ]	[ ]
Blogs/Portais/Notícias	[ ]	[ ]	[ ]	[ ]
Fóruns	[ ]	[ ]	[ ]	[ ]
Cursos a distância	[ ]	[ ]	[ ]	[ ]
M.I. (MSN, Gtalk ...)	[ ]	[ ]	[ ]	[ ]
Sites de relacionamento (Orkut, Facebook ...)	[ ]	[ ]	[ ]	[ ]
Vídeos (Youtube, Blip.tv ...)	[ ]	[ ]	[ ]	[ ]
Outros. Quais?_____	[ ]	[ ]	[ ]	[ ]

Dê sua opinião sobre as afirmações feitas abaixo.

5. Indique o nível de dificuldade para entender o tópico de **abordagem entidade-relacionamento** (entidades, relacionamentos, atributos, cardinalidades)

[ ] Fácil	[ ] Normal	[ ] Difícil	[ ] Muito Difícil	[ ] Não foi ensinado/ Não me lembro

6. Indique o nível de dificuldade para entender o tópico de **abordagem relacional** (tabelas, colunas, chaves primária e estrangeira)

[ ] Fácil	[ ] Normal	[ ] Difícil	[ ] Muito Difícil	[ ] Não foi ensinado/ Não me lembro

7. Indique o nível de dificuldade para entender o tópico de **transformação entre modelos** (transformar o diagrama entidade-relacionamento em tabelas)

[ ] Fácil	[ ] Normal	[ ] Difícil	[ ] Muito Difícil	[ ] Não foi ensinado/ Não me lembro

8. Indique o nível de dificuldade para entender o tópico de **normalização** (1FN, 2FN, 3FN)

[ ] Fácil	[ ] Normal	[ ] Difícil	[ ] Muito Difícil	[ ] Não foi ensinado/ Não me lembro

9. Indique o nível de dificuldade para entender o tópico de **comandos SQL** (SELECT, INSERT, UPDATE, etc.)

[ ] Fácil	[ ] Normal	[ ] Difícil	[ ] Muito Difícil	[ ] Não foi ensinado/ Não me lembro

10. A ideia de um software/site para o ensino de banco de dados é interessante.

[ ] Discordo Totalmente	[ ] Discordo	[ ] Neutro	[ ] Concordo	[ ] Concordo Totalmente

11. Se tivesse que escolher um dos tópicos, em qual deles seria mais útil existir um software para ajudar no seu entendimento?

[ ] abordagem entidade-relacionamento	[ ] abordagem relacional	[ ] transformação entre modelos	[ ] normalização	[ ] comandos SQL

12. Você já fez algum curso a distância? [ ] Sim [ ] Não

13. Se sim, qual(is)?_____

14. Há mais alguma coisa que você ache importante citar sobre seus interesses ou experiências na área de banco de dados? Se sim, descreva brevemente.

_____

_____

APÊNDICE B - Carta de informação e termo de consentimento à instituição

CARTA DE INFORMAÇÃO À INSTITUIÇÃO

Esta pesquisa tem como objetivo principal estudar a adequação de metodologias de engenharia de software para o levantamento de requisitos para software educacional na área de banco de dados. Os dados para o estudo serão coletados através da aplicação de um questionário de perfil de usuário, no qual serão levantados alguns dados demográficos, sobre o uso de computadores e internet e sobre a experiência educacional com a temática de banco de dados.

Para tanto, solicitamos a autorização desta instituição para a aplicação de instrumentos de coleta de dados. Os indivíduos não serão obrigados a participar da pesquisa, podendo desistir a qualquer momento. Quaisquer dúvidas que existirem agora ou depois poderão ser livremente esclarecidas, bastando entrar em contato com a pesquisadora no telefone abaixo mencionado. De acordo com estes termos, favor assinar abaixo. Uma cópia ficará com a instituição e outra com a pesquisadora. Obrigada.

Nathalia Sautchuk Patrício Mestranda do Programa de Pós-Graduação em Engenharia Elétrica (PPGEE/USP) (11) 99678-4238	Diretor José Wanderlei Lua da Silva Escola Estadual Técnica Raposo Tavares (11) 3782-5782/(11) 3782-5529

TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO

Pelo presente instrumento, o Sr. José Wanderlei Lua da Silva, diretor da instituição, após a leitura da CARTA DE INFORMAÇÃO À INSTITUIÇÃO, ciente dos procedimentos propostos, não restando quaisquer dúvidas a respeito do lido e do explicado, firma seu CONSENTIMENTO LIVRE E ESCLARECIDO de concordância quanto à realização da pesquisa.

São Paulo, ___ de _____ de 20_____.

_____
Diretor José Wanderlei Lua da Silva

APÊNDICE C - Carta de informação e termo de consentimento ao sujeito de pesquisa

CARTA DE INFORMAÇÃO AO SUJEITO DE PESQUISA

O presente trabalho se propõe a estudar a adequação de metodologias de engenharia de software para o levantamento de requisitos para software educacional na área de banco de dados. Os dados para o estudo serão coletados através da aplicação de um questionário de perfil de usuário, no qual serão levantados alguns dados demográficos, sobre o uso de computadores e internet e sobre a experiência educacional com a temática de banco de dados.

Os instrumentos de avaliação serão aplicados pela pesquisadora Nathalia Sautchuk Patrício nas dependências da Escola Estadual Técnica Raposo Tavares. Este material será posteriormente analisado e será garantido sigilo absoluto sobre as questões respondidas pelos alunos e também sobre o nome dos participantes. A divulgação do trabalho terá finalidade acadêmica, esperando contribuir para um maior conhecimento do tema estudado.

Os dados coletados serão utilizados na dissertação de mestrado de Nathalia Sautchuk Patrício, aluna do Programa de Pós-Graduação em Engenharia Elétrica (PPGEE) da Escola Politécnica da Universidade de São Paulo (USP).

Agradeço sua participação!

TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO

Eu, o(a) aluno(a) _____, estudante da Escola Estadual Técnica Raposo Tavares, após leitura da CARTA DE INFORMAÇÃO AO SUJEITO DA PESQUISA, firmo meu CONSENTIMENTO LIVRE E ESCLARECIDO de concordância em participar da pesquisa proposta.

São Paulo, ____ de _____ de 20_____.

_____
Assinatura do aluno(a)