Alexandre Antonino Gonçalves Martinazzo

Considerações sobre desenvolvimento colaborativo de software para aprendizagem em plataformas móveis

São Paulo - SP, Brasil

2011

Alexandre Antonino Gonçalves Martinazzo

Considerações sobre desenvolvimento colaborativo de software para aprendizagem em plataformas móveis

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção do título de Mestre em Engenharia Elétrica junto ao Departamento de Engenharia de Sistemas Eletrônicos

Orientadora: Professora Dra. Roseli de Deus Lopes

Departamento de Engenharia de Sistemas Eletrônicos
Escola Politécnica
Universidade de São Paulo

São Paulo - SP, Brasil

2011

DEDICATÓRIA

Dedico este trabalho a meus pais, meu irmão e meus avós, por tudo que fizeram por mim nestes anos.

AGRADECIMENTOS

Agradeço à Professora Dra. Roseli de Deus Lopes, pelo estímulo na contínua melhoria de minha pesquisa.

Agradeço a meus pais, Fatima e Alberto, e meu irmão, Gabriel, pela compreensão e incentivo absolutamente decisivos para a conclusão desta etapa, aos meus avós por me sempre me fazerem buscar as melhores possibilidades e não desistir de meus objetivos; e à Shari Joseph pelo imenso afeto.

Agradeço aos meus amigos e amigas do Laboratório de Sistemas Integráveis, Ana Grasielle Côrrea, Gilda Assis, Irene Ficheman, Leandro Biazon, Marcelo Archanjo, Nathalia Sautchuk Patrício, Ramona Straube, Valkiria Venancio, cujas sugestões e críticas foram fundamentais para o desenvolvimento deste trabalho e também a Adriana Depieri, Cássia Gabriela, Elena Saggio, Fábio Durand, Johny Ho, Márcia Almeida por todo o apoio ao longo dos anos em que nos conhecemos.

Agradeço também aos membros de minha família *kung fu* por me proporcionarem uma nova visão de mundo, à Nina Antonioli pelos questionamentos e amizade. Aos amigos que estiveram próximos, meu muito obrigado; sei que minha dedicação não esteve à altura de vocês.

Agradeço, por fim, aos que direta ou indiretamente contribuíram com este trabalho.

RESUMO

A aplicação de dispositivos eletrônicos móveis na Educação tem ficado cada vez mais intensa na última década. Projetos como UCA (*Um Computador por Aluno*), elaborado pelo Governo Federal Brasileiro, OLPC (*One Laptop per Child*), conduzido por uma organização sem fins lucrativos de mesmo nome, e M-Learning, de universidades europeias, são exemplos de larga escala deste fenômeno. Os impactos educacionais do uso destes dispositivos são estudados nestes e em outros projetos relacionados, havendo diversas indicações de como alcançar de resultados positivos. Não existem, entretando, modelos de Engenharia de Software voltados à produção dos aplicativos usados neste contexto. Visando atender esta demanda, este texto analisa as particularidades no projeto de ferramentas para estas plataformas móveis, com mais interesse no desenvolvimento colaborativo das comunidades de *software* livre e na realidade brasileira. O desenvolvimento de um aplicativo de desenho para o projeto OLPC foi usado como estudo de caso para esta pesquisa. Este aplicativo foi criado usando o método da Programação Extrema por uma equipe de pesquisadores liderada pelo autor e atualmente conta com colaboração da comunidade de *software* livre. A partir desta experiência, foram estendidos dois modelos de Engenharia de Software: a Programação Extrema e a Engenharia sócio-cognitiva. Estas extensões foram elaboradas a fim de a dar apoio a uma equipe presencial (funcionando de acordo com um destes 2 métodos) interagindo com uma comunidade de *software* livre.

Palavras-chaves: Engenharia de Software. Software livre. Desenvolvimento colaborativo. UCA. OLPC. Aprendizagem Móvel.

ABSTRACT

The application of mobile electronic devices in Education has been increasing since the last decade. Projects such as UCA (*Um Computador por Aluno*), formulated by the Brazilian Federal Government, OLPC (One Laptop per Child), conducted by a nonprofit organization of the same name, and M-Learning, organized by European universities, are large-scale examples of that phenomenon. The educational impacts of those devices have been reported in those projects and in related ones; and they also indicate how to achieve positive results. However there are no Software Engineering models focused on producing the kind of applications used in this context. Thence this text analyzes the design particularities of these tools for mobile platforms, with a closer look to the collaborative development in free software communities and the Brazilian reality. The development of a drawing tool for the OLPC project was used as a study case for this research. This application was created using the Extreme Programming model in a team of researchers led by the author and it is currently supported by the OLPC community. Based in that experience, two Software Engineering models have been extended: Extreme Programming and Socio-cognitive Engineering. These extensions were developed in order to support a colocated team (working according to one of these two methods) interacting with a free software community.

Keywords: Software Engineering. Free Software. Collaborative development. UCA. OLPC. Mobile Learning.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

De acordo com o paradigma Construtivista (NAISMITH et al., 2004), os aprendizes devem desempenhar um papel ativo no processo de aprendizagem, pois eles são os responsáveis pela construção do próprio conhecimento. A aprendizagem pode ser vista como uma mudança de estado resultante de experiências passadas e da interação com outros (SIEMENS, 2005). Esse paradigma é fortemente baseado na ideia de "aprender fazendo", assim os aprendizes devem criar continuamente para expressar sua visão. A colaboração também é apontada como um componente importante do processo de aprendizagem, pois pode-se aprender da interação com professores ou com colegas (SHARPLES, 2000).

A colaboração não é importante apenas para a Educação, ela também pode aumentar a produtividade das pessoas. Diversos aplicativos focados em trabalho colaborativo tem surgido nos últimos anos. Este é um fenômeno bastante comum na *Web* 2.0. Aplicações *Web* podem ser usadas para escrever documentos, gerenciar projetos, compartilhar arquivos, entre outros. Algumas destas aplicações permitem que pessoas trabalhem simultaneamente em computadores distintos, fazendo com que a distância não seja mais um impedimento para trabalhar em conjunto. Esse tipo de situação só tem sido atingida por meio da tecnologia.

Tecnologias móveis, como *laptops*, PDAs e telefones celulares permitem novos arranjos para a aprendizagem, diminuindo as restrições de espaços físicos, como uma sala de aula (BULL et al., 2005). A mobilidade física proporcionada por estes dispositivos criou uma nova área de estudos, conhecida como Aprendizagem Móvel (*Mobile Learning*). Esta área esta focada no aprendizado em diferentes espaços e contextos além do aprendizado utilizando tecnologias móveis, sempre considerando o ponto de vista do aprendiz. As tecnologias móveis ampliam as possibilidades de colaboração (SYVÄNEN et al., 2005), uma vez que os dispositivos podem ser transportados pelos aprendizes, tornando a tecnologia disponível independentemente do momento e do lugar.

Em 2007, o governo federal brasileiro lançou um projeto para implantar o uso de netbooks nas escolas públicas, o projeto Um Computador por Aluno (UCA). Uma das metas do projeto é distribuir um netbook para cada um dos alunos de escolas públicas do país (BRASIL, 2010). O projeto UCA está bastante inspirado na iniciativa promovida pela organização não-governamental OLPC (*One Laptop per Child*), que visa produzir um laptop barato o suficiente para toda criança no mundo ter acesso a um.

Tanto o projeto UCA quanto o projeto OLPC são projetos com foco em Educação, portanto, assumem a tecnologia como uma ferramenta importante para a aprendizagem. Desta forma, a introdução da tecnologia neste contexto inclui também algumas mudanças do ponto de vista das escolas. No caso particular do projeto UCA, o Ministério da Educação (MEC) oferece programas de formação às equipes das escolas participantes do projeto, bem como canais de suporte *online*.

Existem diversos modelos de Engenharia de Software aplicados para o desenvolvimento de aplicativos interativos, e não é diferente para as atividades educacionais. Muitos dos produtos desenvolvidos notoriamente conhecidos na comunidade científica de tecnologia para Educação estão ligados a alguma universidade ou instituto de pesquisa, como o Logo e o Scratch, ligados ao *Massachussets Institute of Technology* (RESNICK et al., 2009) e o projeto M-Learning, ligado à universidades europeias (KUKULSKA-HULME et al., 2009). Estas instituições podem adotar métodos convencionais da Engenharia de Software ou modelos específicos. Uma tendência nesta área diz respeito à descentralização dos processos de desenvolvimento, em que equipes de desenvolvedores interagem de diferentes partes do mundo através de ferramentas de colaboração online.

1.1 Problema

Existem diversas plataformas móveis disponíveis para uso em atividades colaborativas no contexto educacional. Alguns esforços já foram feitos no sentido de elaborar um conjunto de boas práticas para elaboração de atividades (SHARPLES, 2000; SHARPLES; CORLETT; WESTMANCOTT, 2002). Este conjunto de práticas tem o intuito de fazer o desenvolvedor (que, em geral, não tem familiaridade com o campo de aplicação do *software* ou de uma determinada tecnologia) projetar um aplicativo ou um dispositivo de maior qualidade e impacto. Estes estudos, entretanto, foram feitos considerando dispositivos específicos, mas possuem uma base bastante sólida no que diz respeito à Educação.

Nas comunidades de *software* livre, dificilmente se encontram recomendações que entram no âmbito da área de uso do *software*, ou das tecnologias envolvidas, pois há uma tendência de haver poucos especialistas com competências diferentes da programação (BARCELLINI et al., 2008; YEATS, 2006). Portanto, há uma lacuna no que se refere ao desenvolvimento embasado de aplicativos e tecnologias móveis focadas em colaboração na Educação.

A comunidade científica de Engenharia de *Software* aponta para a tendência de ambientes de desenvolvimento distribuídos e utilização de componentes de terceiros na produção de software. Estes aspectos, entretanto, não estão discutidos sob o contexto do desenvolvimento de aplicativos educacionais. Observando o cenário do projeto UCA, que requer inovações tanto em Engenharia tanto quanto em Educação, é necessário entender quais as necessidades geradas pelas particularidades das plataformas e quais as necessidades do público-alvo, já que **não existem diretrizes de desenvolvimento colaborativo de software** nestes projetos.

A pergunta norteadora da pesquisa é: *o que desenvolvedores de software precisam saber para criar um novo aplicativo (ou portar um aplicativo já existente) no contexto dos projetos UCA e OLPC?*

1.2 Hipótese

É possível criar diretrizes capazes de dar subsídios para o desenvolvimento de novos aplicativos para plataformas móveis ou para a reengenharia de aplicativos já existentes visando sua adequação para estas plataformas, dentro do contexto dos projetos UCA e OLPC.

1.3 Objetivos

O objetivo deste trabalho é sugerir extensões no processo de desenvolvimento colaborativo de aplicativos para plataformas móveis, de forma a *aplicar os fundamentos* teóricos da Aprendizagem Móvel e *fornecer diretrizes nos moldes da Engenharia de Software*, tanto para o projeto de novos aplicativos quanto para a reengenharia de aplicativos já disponíveis.

1.3.1 Objetivos específicos

Os objetivos específicos desta pesquisa são:

- Identificar o contexto e as caraterísticas das plataformas móveis disponíveis na Educação Brasileira;
- Avaliar outros trabalhos que apresentem recomendações ou métodos para o desenvolvimento de software para Educação e para plataformas móveis;
- Acompanhar e relatar o desenvolvimento de aplicativos para plataformas móveis;
- Desenvolver um conjunto de diretrizes para desenvolvimento de aplicativos voltados para aprendizagem com foco nestas plataformas.

1.4 Justificativa

As considerações sobre desenvolvimento de aplicativos são importantes porque surgem novas plataformas aplicadas em contextos novos (como o projeto UCA e o projeto OLPC). Pode-se dizer também que o perfil do aprendiz é novo, mas este não é um problema do escopo deste trabalho; assume-se que esta variável é tratada no nível pedagógico, daí buscar o suporte na teoria de Aprendizagem Móvel.

Há diversas plataformas eletrônicas sendo usadas na Educação, como celulares e *laptops*. Embora estas não sejam inéditas no cotidiano, internacionalmente há poucos casos de uso massivo aplicados na Educação, tanto na Educação básica quanto na Educação superior. Estas plataformas têm, em comum, o fato de proverem mais mobilidade aos usuários. Este aspecto traz desafios do ponto de vista de desenvolvimento de *software* e de atividades a serem desenvolvidas nos espaços de aprendizagem. Algumas plataformas também trazem inovações do ponto de vista de *hardware*, como dispositivos de entrada exclusivos ou acesso à novos protocolos de comunicação.

No contexto da Educação básica, há, particularmente, uma nova geração de usuários de computador, chamada de nativos digitais. Autores argumentam que esta geração tem necessidades de aprendizado diferentes (RESNICK, 2002; VAVOULA; SHARPLES, 2002; RESNICK et al., 2009), por serem nascidos depois das tecnologias digitais serem amplamente disponíveis.

No panorama brasileiro, há ainda a iniciativa de implantar a utilização massiva de tecnologias em escolas através do programa Um Computador por Aluno (UCA). Neste programa, estudantes das escolas públicas brasileiras recebem *laptops* de baixo custo como uma ferramenta para ser usada dentro e fora das salas de aula. Internacionalmente, o projeto conhecido como *One Laptop per Child* (OLPC), originado no *Massachussets Institute of Technology* (MIT), tem o objetivo de distribuir computadores de baixo custo para crianças de países em desenvolvimento.

Durante alguns anos, Universidades europeias desenvolveram um projeto chamado M-Learning. Este projeto estudou os impactos do uso de dispositivos eletrônicos portáteis no contexto escolar, dando maior ênfase aos estudantes (KUKULSKA-HULME et al., 2009).

Embora os estudos do projeto M-Learning tenham sido majoritariamente baseados em dispositivos compatíveis com o tamanho da mão (*hand-held devices*), há diversas semelhanças com as caraterísticas dos dispositivos usados nos projetos UCA e OLPC. Muitas das publicações relatando os impactos do projeto trazem grande fundamentação teórica em termos de paradigmas de aprendizagem, além de dados relativos ao uso de longo prazo das tecnologias móveis (NAISMITH et al., 2004). Desse modo, há bastante a ser aproveitado das experiências do projeto M-Learning.

1.5 Organização do trabalho

O **capítulo 2** apresenta os principais conceitos relacionados ao tema desta pesquisa, tratando de aspectos de aprendizagem e sua relação com ferramentas digitais, da teoria de Aprendizagem Móvel e de ambientes computacionais focados em plataformas móveis. O capítulo também discute alguns modelos para o desenvolvimento de aplicativos interativos.

O **capítulo 3** é um relato da experiência do autor no desenvolvimento colaborativo de aplicativos para o projeto OLPC e também de sua participação no projeto UCA. O capítulo conta brevemente o histórico dos projetos e depois descreve a organização da comunidade à luz dos conceitos apresentados no capítulo anterior e fecha narrando o desenvolvimento da plataforma de desenho para o ambiente Sugar do projeto OLPC.

Uma proposta de modelagem da interação entre especialistas e comunidades de *software* livre é feita no **capítulo 4**. O capítulo inicia-se com algumas considerações a respeito do ponto de partida da modelagem, partindo depois para a modelagem segundo métodos de Engenharia de *Software* já conhecidos. As conclusões, as contribuições desta pesquisa

para a comunidade e alguns trabalhos futuros são apresentados no **capítulo 5**.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os fundamentos teóricos e os trabalhos relacionados ao tema desta pesquisa.

2.1 Paradigmas de Aprendizagem

Existem diversas discussões e fóruns acerca de como as pessoas aprendem. Nesta pesquisa, o entendimento de como se dá a aprendizagem serve de apoio para desenvolvimento das atividades de pesquisa. O entendimento das condições às quais estão submetidos os usuários de tecnologias digitais (sejam elas *hardware* ou *software*) são os aspectos de grande interesse em termos de aprendizagem. Estas condições estão relacionadas principalmente ao uso de ferramentas digitais para fins de autoria e colaboração, conforme Papert (1999).

Do ponto de vista da Pedagogia, a autoria é o processo através do qual os estudantes usam meios diversos para produzir conteúdos. Nos ambientes com apoio de tecnologias computadorizadas, as ferramentas digitais de autoria permitem aos estudantes expressarem suas ideias. Para Naismith et al. (2004), as atividades de aprendizagem podem ser classificadas em seis categorias: Behavioristas, Construtivistas, Situadas, Colaborativas, Informais e duradouras, atividades de suporte ao ensino e à aprendizagem. Os autores consideram que as atividades podem ser classificadas em mais de uma categoria por vez, ou seja, as categorias não são exclusivas.

- *Paradigma behaviorista*: a aprendizagem é o resultado de um processo de estímulo e reação. O processo toma forma através do reforço positivo ou negativo de cada resposta. Ao aplicar esta classificação às tecnologias da educação, a apresentação do problema é o estímulo e a solução proposta pelo aprendiz é a reação nas situações com a presença do computador.

- *Paradigma construtivista*: esta abordagem define a aprendizagem como um processo ativo. Assim, as pessoas são vistas como construtores da informação, de forma que o conhecimento atual e o passado são usados para dar forma a novos conceitos. Portanto, as experiências passadas são importantes para o entendimento de novas situações. Num ambiente construtivista, os estudantes devem ser estimulados a descobrir novos conceitos. Os autores também consideram que simulações participativas são um bom exemplo de ambientes apoiados por computador cuja base está na teoria construtivista.

- *Paradigma de aprendizagem situada*: neste paradigma, aprender é parte de um processo de interação social. É necessário que os aprendizes estejam em contato com contextos e culturas autênticos (que podem ser deliberados ou não-intencionais) e também participem de uma comunidade de prática. Uma comunidade de prática é um grupo dedicado a fazer e/ou melhorar algo através da interação com outros. Os instrutores trabalhando de acordo com este paradigma devem prover situações nas quais os aprendizes tenham contato com problemas reais mesmo antes de ter entendimento completo dele. Naismith et al. (2004) consideram que tecnologias móveis podem potencializar o contexto por serem levadas fisicamente a diferentes locais.

- *Paradigma colaborativo*: a aprendizagem também é entendida como uma parte do processo de participação social; assim, a interação é um componente fundamental deste paradigma. Aprender é uma experiência recíproca que pode ser descrita como uma conversação entre agentes de aprendizagem (como estudantes e professores, ou até mesmo dispositivos tecnológicos); os aprendizes devem interagir com outros agentes para dividir seu entendimento. Este paradigma inclui a Aprendizagem Colaborativa Apoiada por Computador (em inglês *Computer Supported Collaborative Learning* - CSCL); o papel da tecnologia é dar suporte a estas interações e aumentar as possibilidades de comunicação.

- *Paradigma de aprendizagem informal*: aprender não é uma atividade restrita às salas de aula, ou seja, as situações de aprendizagem não tem um lugar específico para acontecer. Sob este ponto de vista, a aprendizagem é frequentemente informal, especialmente entre adultos. O papel das tecnologias é permitir que as pessoas aprendam o tempo todo e em qualquer lugar, dando assistência em situações intencionais e não-intencionais de aprendizagem. Esta teoria é bastante estudada em situações de aprendizagem no campo profissional (ERAUT, 2000).

- *Atividades de suporte ao ensino e à aprendizagem*: estas atividades estão relacionadas à administração e gerenciamento em sala de aula, além de revisão e avaliação.

Analisando os paradigmas construtivista, situado, colaborativo e informal, nota-se que eles têm alguns aspectos em comum. Todos estes têm uma abordagem mais centrada no aprendiz e dão grande importância à interação entre os estudantes. Desta forma, o projeto das tecnologias devem manter também uma relação próxima ao usuário, justamente para manter o caráter *pessoal*. Além disso, para dar apoio à interação entre aprendizes é desejável que estas tecnologias tenham capacidade de se *comunicar em rede*.

2.1.1 Autoria, colaboração e ferramentas digitais

A autoria é um componente fundamental das atividades Construtivistas. Papert afirma que parte do processo de aprendizagem é sobre a coleta de informações, lendo livros, ouvindo os professores ou visitando páginas *Web*. Outra parte do aprendizado tem a ver com "fazer coisas, realizar e construir coisas" (PAPERT, 1999); ao fazer isso, os alunos estão construindo seus conhecimentos. Segundo o autor, no contexto escolar, os instrutores/professores devem estimular os alunos a descobrir os princípios abordados.

A proposta Construtivista representa uma mudança na escola tradicional. É uma mudança de atitude que transforma os alunos de receptores de informações em "construtores ativos de conhecimento", se equipados com as ferramentas apropriadas (NAISMITH et al., 2004). Papert usa a palavra Construcionismo para se referir à ideia de "aprender fazendo", ou seja, quando os alunos desempenham um papel ativo no processo de aprendizagem (PAPERT, 1999). Esta é uma razão relevante para a produção de ferramentas de autoria digital.

A capacidade de criar coisas usando computadores é um tema importante para a sociedade do futuro. Resnick considera que os computadores são a ferramenta de criação mais poderosa inventada (RESNICK, 2002). As pessoas têm contato diário com as tecnologias digitais em ambientes de escritório e também em atividades informais. Os custos decrescentes de computadores os tornam disponíveis para um público mais amplo, reduzindo a desigualdade digital em termos de acesso à tecnologia (RESNICK, 2002).

Mas o acesso em si não garante a fluência digital necessária para enfrentar alguns dos desafios do futuro. Educação contínua (ou seja, durante a vida toda) e auto-aperfeiçoamento são consideradas fundamentais para melhorar uma sociedade baseada no conhecimento

(MAGALHÃES; KNIGHT; COSTA, 2009). Além disso, fluência digital não se limita apenas sobre como usar o computador, mas saber *expressar-se* usando um (RESNICK, 2002): é usar a *criatividade* aliada a um pensamento sistemático através de uma ferramenta computacional (RESNICK et al., 2009).

O projeto *Computer Clubhouse* (desenvolvido no MIT) tem como objetivo abordar o problema da fluência digital para os jovens. Ao frequentar os clubes, as pessoas são incentivadas a criar coisas diversas usando computadores, como jogos, simulações, música ou páginas *Web*. Este projeto estimula a criação de confiança para fazer mais jovens como aprendizes (RESNICK, 2002, 2003).

A *aprendizagem colaborativa* ocorre quando um aprendiz interage com outro agente de aprendizagem para troca de experiências de aprendizagem (LIPPONEN, 2002). Sharples, Corlett e Westmancott (2002) argumentam que a aprendizagem bem-sucedida é fruto de um processo construtivo, onde a conversação toma um lugar central. Sharples define a conversação como a comunicação entre os sistemas de conhecimento (aprendizes, professores ou as tecnologias propriamente ditas) sobre o que se sabe. Sob este ponto de vista, a importância da colaboração no processo de aprendizagem é questionar e ser questionado sobre um de conceitos e interpretações. A colaboração pode ser definida como a "co-construção de conhecimento e compromisso mútuo" dos seus pares, configurando uma "forma especial de interação" (LIPPONEN, 2002).

Dillenbourg (1999) vê semelhanças entre sistemas cognitivos individuais e sistemas cognitivos coletivos. Ele não considera que as pessoas "aprendem com a colaboração". Uma única pessoa não aprende simplesmente por ser um indivíduo; da mesma forma, pessoas em um sistema cognitivo coletivo não aprendem com o simples fato de formar um grupo. Dillenbourg destaca que as atividades realizadas pelos aprendizes (como leitura) são as verdadeiras responsáveis por desencadear os "mecanismos de aprendizagem" (como a indução e dedução). Quando em grupo, os aprendizes ainda podem executar algumas das atividades individuais, também desencadeando os mecanismos de aprendizagem. Mas a interação com os outros possibilita novas atividades (como a explicação e discussão), ativando novos mecanismos de aprendizagem (tais como extração de conhecimento, interiorização e apropriação).

A aprendizagem colaborativa apoiada por computador (*Computer-supported collaborative learning* - CSCL) é uma área do conhecimento que estuda como "a tecnologia pode melhorar a interação entre colegas e trabalhar em grupos, e como a colaboração e facilitar

o compartilhamento de tecnologia e distribuição de conhecimentos e experiências entre os membros da comunidade" (LIPPONEN, 2002). Desta forma, Lipponen constitui CSCL como uma situação de colaboração, e desencadeia os mecanismos de aprendizagem descritos anteriormente, uma vez que pode ser visto como um contrato entre os aprendizes, ou entre colegas e o professor (DILLENBOURG, 1999). Dillenbourg também destaca que esta interação particular observada na colaboração é esperada sempre em situações coletivas, mas não há nenhuma garantia da ativação dos mecanismos correspondentes.

2.2 Aprendizagem Móvel

As tecnologias móveis permitem aos professores e aprendizes criar diferentes espaços de aprendizagem, que não são limitados aos contextos de aprendizagem formal. Do ponto vista dos aprendizes, o aprendizado pode ser classificado como móvel em três diferentes dimensões (VAVOULA; SHARPLES, 2002):

- com relação ao contexto: a aprendizagem pode ocorrer em diferentes contextos; pode-se aprender a partir de uma demanda de trabalho ou de lazer;
- com relação aos espaços: a aprendizagem não está restrita a um único espaço, ela pode ocorrer em casa, no escritório de trabalho ou em parques temáticos;
- com relação ao tempo: não existe horário pré-determinado para aprender; a aprendizagem pode acontecer em diferentes períodos do dia, não importando ser um fim de semana, dia útil ou feriado.

A Aprendizagem Móvel também se concentra na aprendizagem *utilizando tecnologias móveis*; fazer as tecnologias estarem disponíveis a qualquer hora e lugar pode aumentar as possibilidades de aprendizagem. Autoria e colaboração são compatíveis com as práticas da Aprendizagem Móvel (NAISMITH et al., 2004). Ao considerar a mobilidade física dos aprendizes, podemos enriquecer as maneiras de realizar as atividades de aprendizagem. As experiências de aprendizagem podem ganhar qualidade quando se permite estudar em lugares onde antes não seria possível (BULL et al., 2005).

Uma classificação das tecnologias através de dois eixos ortogonais (pessoal-compartilhado, portátil-estático) foi fornecido por Naismith et al. (2004), esta classificação é mostrada na figura 2.1.

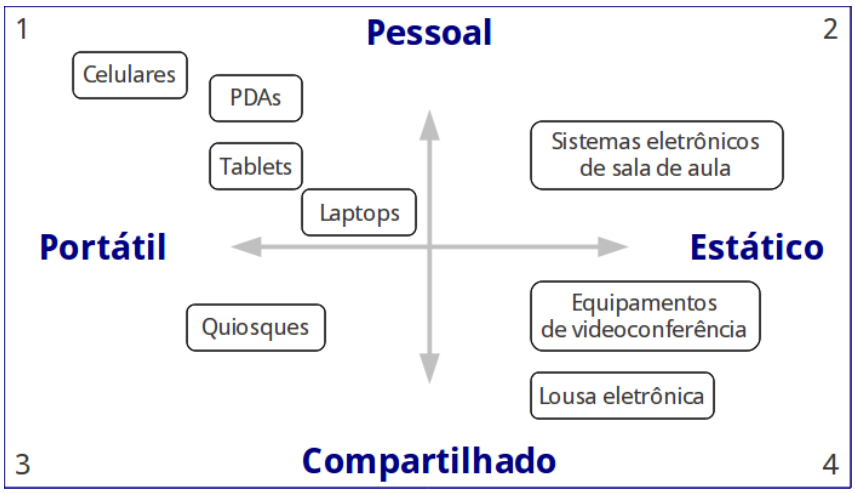

Figura 2.1: Classificação de tecnologias móveis (adaptado de (NAISMITH et al., 2004)).

Esta classificação ajuda a identificar como a mobilidade das plataformas, a mobilidade dos aprendizes e o compartilhamento de dispositivos afetam as experiências de aprendizagem. O quadrante 1 mostra os dispositivos pessoais e portáteis, como telefones celulares, *laptops* e videogames portáteis. Estes são os dispositivos mais comumente associados ao termo "tecnologias móveis". Os dispositivos exemplificados normalmente têm também a capacidade de operar em rede, permitindo a comunicação entre dispositivos e compartilhamento de informações. Embora esses dispositivos portáteis tenham caráter fortemente pessoal (em termos físicos), os dados presentes neles podem seguir uma tendência diferente, sendo comum haver compartilhamento (NAISMITH et al., 2004).

O segundo quadrante mostra tecnologias pessoais estáticas, tais como sistemas de resposta em sala de aula (os alunos podem responder a perguntas de escolha múltipla utilizando estes dispositivos). A interação continua pessoal, já que o dispositivo está alocado a um único usuário; mas a tecnologia não poderia ser tirada da sala de aula, por isso é considerada estática.

O terceiro quadrante representa as tecnologias estáticas capazes de proporcionar experiências de aprendizagem para os alunos que se deslocam, pois "ser movido fisicamente de um lugar para outro não é a única maneira para as tecnologias móveis serem portáteis" (NAISMITH et al., 2004). Exposições em museus interativos são um exemplo para o quadrante 3, pois eles fornecem o acesso à informação a um público em mudança. Assim, a portabilidade refere-se aos aprendizes e não à tecnologia. Ainda assim, estes dispositivos são normalmente utilizados por mais de uma pessoa no momento; logo, sua classificação fica no conjunto das tecnologias compartilhadas.

O quarto quadrante apresenta dispositivos com fortes características de compartilhamento por seus tamanhos maiores. Um exemplo seria uma instalação de vídeo-conferência, que pode ser usado por muitas pessoas ao mesmo tempo mas dificilmente seria movida. Naismith et al. (2004) consideram as tecnologias móveis como aquelas incluídas nos quadrantes 1-3 e as do quadrante 4 que não estão no extremo do eixo estático. Estas tecnologias foram incluídas apenas para dar uma ideia da completude da classificação.

Ao tratar da interação com a tecnologia, Sharples, Corlett e Westmancott (2002) consideram que recursos de aprendizagem móvel, "deveriam, idealmente, se encaixar perfeitamente nesse padrão complexo de oportunidades de aprendizagem e recursos". Por isso, há algumas questões-chave para os desenvolvedores de tecnologia e educadores sobre o planejamento e projeto bem sucedido de ambientes de aprendizagem móvel (NAISMITH et al., 2004):

- contexto: a coleta e utilização de informações contextuais podem não combinar com o desejo de anonimato e privacidade dos aprendizes,
- mobilidade: a capacidade de ligar as atividades da escola ao mundo proporciona aos aprendizes a capacidade de "escapar" da sala de aula e participar de atividades não propostas por qualquer professor ou previstas no currículo.
- aprendizagem ao longo do tempo: instrumentos eficazes são necessárias para o registo, a organização e recuperação de experiências de aprendizagem.
- informalidade: os aprendizes podem abandonar o uso de determinadas tecnologias se lhes parecer que sua socialização está comprometida.
- propriedade: os aprendizes querem possuir e controlar sua tecnologia pessoal, mas isso representa um desafio quando estas tecnologias são trazidas para a sala de aula.

O contexto é uma das questões mais importantes, já que as tecnologias móveis são intrinsecamente mais orientadas pelas condições de uso do que pelas orientações dadas em salas de aula. O contexto, neste caso, pode envolver a trajetória e a motivação do aprendiz, bem como os recursos disponíveis no momento (SHARPLES; CORLETT; WESTMANCOTT, 2002). As tecnologias móveis são primariamente computadores, mas as características descritas anteriormente sugerem que elas suportam atividades diferentes das feitas atualmente em computadores *desktop* (geralmente compartilhados e estáticos).

A mobilidade de alunos e dispositivos incentiva novas práticas, além de dar condições para o surgimento de novas interações entre os aprendizes e o ambiente (NAISMITH et al., 2004). Estas tecnologias também podem levar os aprendizes a se conectar a novos contextos, "ajudando a formar pontes entre a aprendizagem formal e informal" (KUKULSKA-HULME et al., 2009).

Existem muitas iniciativas de Aprendizagem Móvel em todo o mundo. O projeto M-Learning durou 33 meses e objetivava propor e avaliar uma arquitetura para a aprendizagem móvel. O projeto ENLACE concebeu e implementou uma infra-estrutura técnica para apoiar atividades de aprendizagem colaborativa dentro e fora da escola. O projeto "Mistério no Museu" (*Mystery at the Museum*) usava dispositivos móveis para aumentar o engajamento em atividades do museu através de uma abordagem baseada em jogo; Kukulska-Hulme et al. (2009) fornecem mais detalhes de algumas destas iniciativas.

2.2.1 Tecnologias hand-held para aprendizagem

No contexto da Aprendizagem Móvel no projeto M-Learning, alguns autores centralizaram suas publicações na descrição dos sistemas que compunham os experimentos realizados. Dentro desta realidade, optou-se por desenvolver aplicações em aprendizagem para dispositivos *hand-held* (ou seja, dispositivos portáteis do tamanho de uma mão). Assim, Sharples (2000) analisou os requisitos para o projeto de tecnologias adequadas a suportar atividades de aprendizagem de longo prazo; as variáveis são as seguintes:

- portabilidade: disponível sempre que um aprendiz necessita;
- uso individual: é capaz de apoiar a aprendizagem pessoal e adaptável às habilidades pessoais e necessidades;
- não-intrusivo: a tecnologia não deve se intrometer na situação;
- disponibilidade: estar sempre pronta para proporcionar a comunicação com alunos e professores;
- capacidade de adaptação: a tecnologia deve ser sensível ao contexto e evoluir de acordo com o aluno conhecimento;
- persistência: as produções do aprendiz devem estar disponíveis, apesar de mudanças na tecnologia;

- utilidade: adequado para as necessidades diárias de comunicação, de referência, trabalho e aprendizagem;
- intuitivo: fácil de usar, mesmo sem nenhuma experiência anterior.

O estudo de Sharples (2000) está focado em dispositivos *hand-held*, mas a abordagem de coleta de requisitos serve para outros dispositivos. A maioria do requisitos apresentados são compatíveis com outras plataformas que não apenas as abordadas. Os requisitos apresentados, com exceção da não-intrusividade, são adequados para plataformas móveis (como as apresentadas no quadrante 1 da figura 2.1) com capacidade de se conectar à Internet. Evidentemente, a adequação da interface é uma questão bastante importante, daí a importância de se quebrar a metáfora do uso do computador, ainda muito ligada ao ambiente de escritório (OLPC, 2007; SHARPLES; CORLETT; WESTMANCOTT, 2002).

O trabalho de Sharples, Corlett e Westmancott (2002) é baseado no método da engenharia sócio-cognitiva (vide seção 2.4), procurando relacionar as interações entre pessoas e tecnologias computacionais.

2.3 Ambientes computacionais para plataformas móveis

Novos dispositivos eletrônicos pessoais têm surgido nos últimos anos, bem como novas formas das pessoas interagirem com seus dispositivos. Essas mudanças se dão tanto nas formas disponíveis para entrada de dados (como a popularização de telas sensíveis ao toque) e na interfaces gráficas dos dispositivos. Esta última representa um fator bastante importante na aceitação dos diferentes dispositivos, uma vez que um bom projeto de interação (refletido também numa boa interface) determina o uso eficiente e agradável do aparelho.

O iPhone, desenvolvido pela Apple, é um exemplo de dispositivo móvel de muito sucesso no mercado mundial, tanto pelos recursos apresentados quanto pela interface gráfica. Os desenvolvedores desta plataforma têm diversos documentos de apoio para projeto de interface nesta plataforma.

As seções a seguir apresentam algumas iniciativas de *software* livre de criação de novos ambientes computacionais focados em romper os paradigmas de interface.

2.3.1 O ambiente Sugar

O projeto *One Laptop per Child* (OLPC) tem por objetivo criar um laptop barato o suficiente para ser distribuído para todas as crianças do mundo. Isto permitiria a criação de oportunidades para as "crianças mais pobres do mundo, proporcionando a cada criança um *laptop* robusto, baixo custo, baixa potência e conectado; com conteúdo e *software* projetado para aprendizagem colaborativa, alegre e autônoma" (OLPC, 2007). Por tratar-se de um projeto de educação, esta é, claramente, uma iniciativa Aprendizagem Móvel.

O projeto OLPC começou no final de 2005 e alavancou o desenvolvimento dos *netbooks* de hoje (BAJARIN, 2008). A iniciativa da OLPC é provavelmente o primeiro esforço em grande escala para a concepção deste novo tipo de laptop, visando baixo custo e baixo consumo de energia (WARSCHAUER; AMES, 2010). Outro destaque desta iniciativa é ser totalmente baseada em soluções livres (*free and open source software*).

O laptop da OLPC é chamado XO e interface gráfica foi chamada de Sugar. O hardware foi projetado para ser muito mais barato que outros *laptops* e funcionar mesmo sob condições adversas dos ambientes, como ter umidade sobre o teclado ou suportar quedas. O ambiente Sugar é baseado na teoria Construcionista, assim o ambiente Sugar é composto principalmente de ferramentas de criação colaborativa (MARTINAZZO et al., 2008; SUGAR LABS,). Os aplicativos são chamados de "atividades" na interface do Sugar, e elas usam verbos em vez de substantivos como nomes dos aplicativos. Por exemplo, "Escrever" para o editor de texto, "Pintar" para o aplicativo de desenho e "Gravar" ao se referir à atividade que acessa câmera e microfone. O foco destas atividades é justamente o de proporcionar a experiência de "aprender a aprender" colaborativamente; daí a grande facilidade para se engajar em atividades colaborativas. As bibliotecas de colaboração do ambiente gráfico permitem compartilhar ou entrar em atividades compartilhadas com um único clique, obedecendo o princípio de projeto de ser fácil de começar a usar e ter várias possibilidades de trabalho (em inglês, usa-se a expressão "*low floor, high ceiling*") (OLPC, 2007; RESNICK et al., 2009).

O hardware do XO trouxe algumas inovações, tais como a implementação da chamada rede Mesh (baseada no rascunho do padrão IEEE 802.11s), baixo consumo de energia e uma tela que pode ser lida mesmo sob a sol luz. Como apontado por Syvänen et al. (2005), os locais nos quais se pode aprender não são controlados e conhecidos, por isso pode afetar a disponibilidade e a qualidade de ferramentas e serviços online. Esta pode ser uma restrição para o compartilhamento de informações e colaboração mediada por

computador, embora a mobilidade do dispositivo possa encorajar os aprendizes a criar seus próprios locais de aprendizagem.

Depois de alguns anos a OLPC teve duas frentes abrigadas dentro da mesma instituição: a iniciativa de *hardware* e a do ambiente Sugar. O desenvolvimento do *hardware* é tarefa da OLPC, enquanto foi criada uma nova fundação para o desenvolvimento do Sugar, chamada de Sugar Labs.

As especificações de hardware do XO combinadas com as diretrizes de interface Sugar fez os desenvolvedores de aplicativos repensarem algumas decisões de *design*, como arquitetura e interface. As atividades do Sugar são *software* livre, alguns deles foram construídos a partir do zero, mas a maioria delas são na verdade aplicações desktop reprojetadas para se adequar às exigências do ambiente Sugar. Isto é principalmente devido à colaboração e apoio à adaptação ao sistema *Journal*. O sistema *Journal* é uma inovação proposta no projeto OLPC, que permitem aos usuários o acesso a arquivos criados sem se referir ao sistema de arquivos diretamente. Em vez disso, não se lida com arquivos, mas com objetos e pessoas. O *Journal* registra cada interação do usuário com o laptop, mantendo um histórico das coisas que a criança fez e as atividades que ela tenha participado (MARTINAZZO et al., 2008; OLPC, 2007).

O ambiente Sugar também utiliza uma nova metáfora para organizar a interface gráfica e visualização de informações. A metáfora do Desktop não é apropriada para ambientes de aprendizagem (SHARPLES; CORLETT; WESTMANCOTT, 2002) e outras metáforas e imagens do sistema devem ser criadas para tornar a interação mais reconhecível. O ambiente Sugar é baseado na metáfora de proximidade (*zoom*), para refletir a ideia de colaboração (OLPC, 2007). A figura 2.2 mostra a metáfora da proximidade proposta no ambiente Sugar, enquanto a figura 2.4 mostra como é a interface da atividade Escrever. A proximidade tem quatro níveis: **Vizinhança** (*Neighborhood*), **Grupo** (*Group*), **Casa** (*Home*) e **Atividade** (*Activity*), onde o primeiro é o nível mais afastado e o último o mais próximo.

A **Vizinhança** mostra os outros laptops XO conectados à rede e as atividade das quais eles participam. O nível **Grupo** representa os amigos, colegas de classe e outros grupos dos quais o aprendiz faça parte. O nível **Casa** permite trocar entre as atividades executadas e verificar o status da rede e do *laptop* em si, por exemplo. Além disso, é desta visão que se pode iniciar as atividades no ambiente Sugar através da moldura mostrada na figura 2.3; esta é a visão que mais lembra um ambiente *desktop* tradicional.

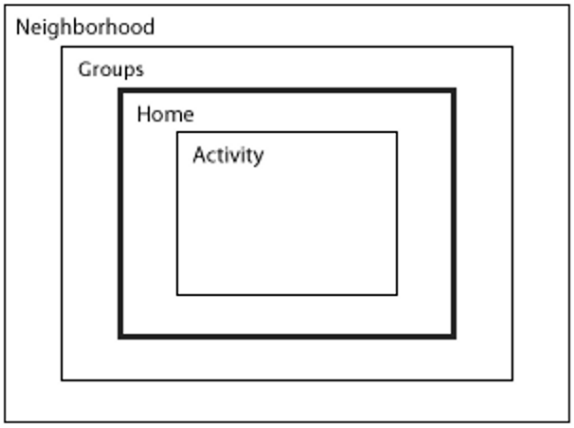

Figura 2.2: Metáfora de proximidade do Sugar (retirado de OLPC (2007)).

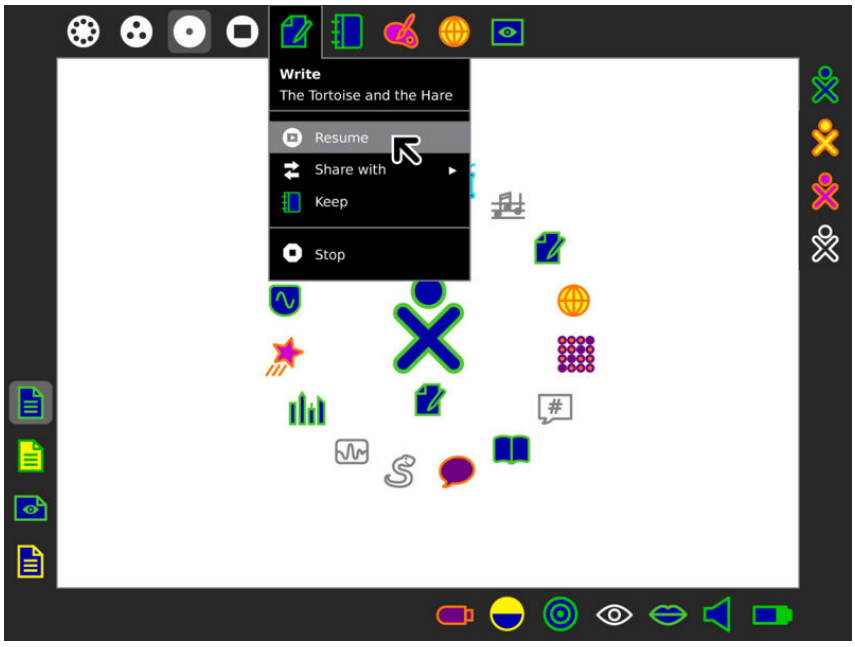

Figura 2.3: A moldura (*frame*) do ambiente Sugar (retirado de OLPC (2007)).

O topo do *frame* apresenta a lista dos lugares disponíveis na interface, mostrando os níveis de proximidade descritos anteriormente. Além disso, a moldura permite trocar entre os aplicativos em execução. Quando o usuário está visualizando uma atividade, a moldura não fica visível e só é mostrada ao colocar o ponteiro nos cantos da tela.

O nível da **Atividade** é o último nível e mostra a atividade executada no momento; a figura 2.4 exemplifica este nível com o editor de texto. Quando uma atividade é mostrada, toda a tela é preenchida, ou seja, os aplicativos são executados em tela cheia. Esta ruptura das interfaces gráficas é observada também em outros ambientes voltados para *netbooks*, como MeeGo e Android (vide seção 2.3.3).

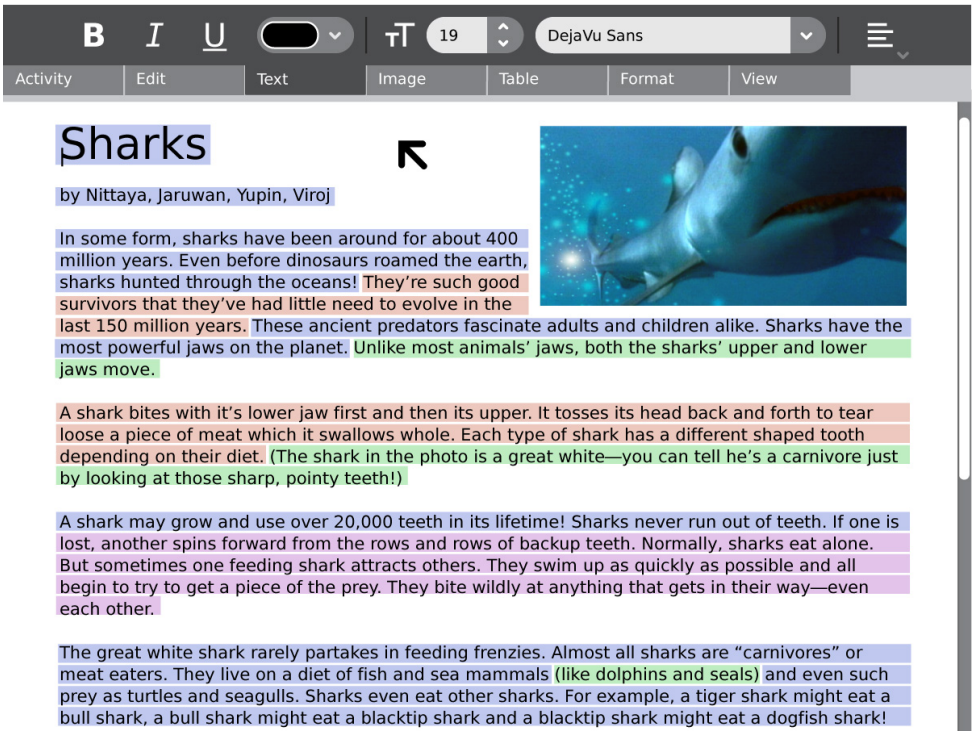

Figura 2.4: A atividade Escrever (retirado de OLPC (2007)).

A arquitetura do ambiente Sugar fornece um conjunto de artefatos para criar *software* colaborativo baseada nas bibliotecas DBUS e Telepathy. Esta combinação (também chamada de D-Tubes) fornece um mecanismo de *Remote Procedure Call* (RPC) com suporte a eventos através da rede Mesh. A biblioteca DBUS cuida do processo de comunicação inter-processos enquanto a biblioteca Telepahty lida com passagem de mensagens através de *firewalls* e cuida do gerenciamento dos contatos. As características da biblioteca Telepahty são responsáveis pelas visões de Vizinhança e Grupo na metáfora de zoom do Sugar. A abstração fornecida pelo D-Tubes facilita a integração entre as atividades XO e outras aplicações de GNU/Linux, pois a biblioteca está disponível para outras plataformas.

O ambiente de programação criado para o Sugar permite a criação de *software* de colaboração descentralizada, ou seja, o modelo cliente-servidor não é necessário. O projeto OLPC é todo baseado na hipótese um-para-um, em outras palavras, o XO é um dispositivo pessoal. A capacidade da rede compatível com o padrão IEEE 802.11s torna esses dispositivos mais poderosos, os alunos podem experimentar a colaboração mesmo sem conexão à Internet.

A figura 2.5 mostra como está organizado o sistema em uma visão de camadas. O ambiente está baseado numa estrutura GNU/Linux usada em outras distribuições do mesmo gênero, mas tem o grande diferencial de prover uma interface de programação (API - *Application Programming Interface*) de acordo com a inovações da sua interface gráfica e com a construção de aplicativos colaborativos.

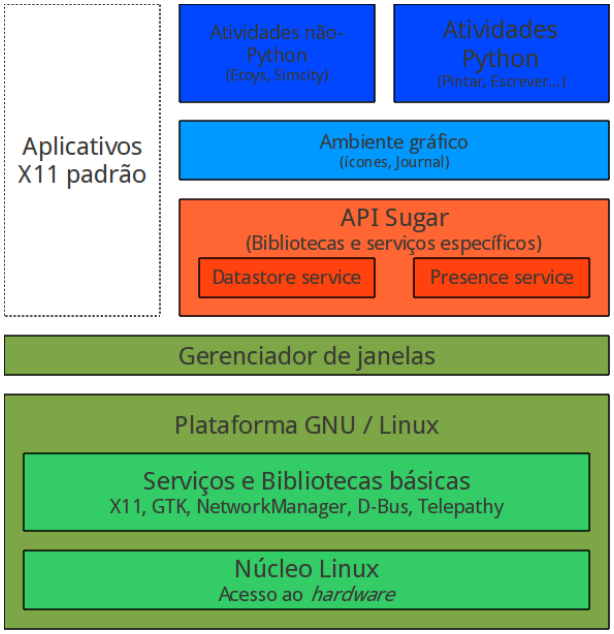

Figura 2.5: Arquitetura em camadas do ambiente Sugar proposto pela OLPC (adaptado de `http://wiki.sugarlabs.org/go/Development_Team/Architecture`).

O ambiente Sugar traz o *Journal* como um recurso único para gerir a produção. Não usar a abstração de arquivos e pastas pode ser uma grande vantagem para os aprendizes mais jovens. O *Journal* foi projetado para assemelhar-se a memória humana (ordenando as produções de acordo com a data e frequência de uso) (OLPC, 2007) e com a organização de conteúdos por etiquetas (*tags*), já difundida na Internet em *blogs*, *microblogs* e outros. Tal como portais e serviços de Internet, o Journal usa *tags* e mini-descrições para as produções armazenadas. Esta já é uma maneira consagrada por muitos portais de conteúdo, seja de imagem, texto ou multimídia, como nos serviços de *blog wordpress* e

blogspot, no serviço de compartilhamento de *links* *D*elicious, ou como no Youtube.

2.3.2 O ambiente Classmate Metasys

O ambiente Classmate Metasys é uma distribuição GNU/Linux criada para rodar em *laptops* Classmate da Intel. Sua interface gráfica é baseada no ambiente KDE (*K Desktop Environment*) com adição de alguns aplicativos voltados para o contexto educacional. O ambiente é desenvolvido pela empresa brasileira International Syst S/A [1] e foi adotado na primeira compra de *laptops* do projeto UCA.

O modelo proposto pela empresa é de integrar os sistemas dos computadores disponíveis para professores e alunos e o servidor da escola, integrando as informações e dados dos computadores pessoais nos servidores. Se o usuário puder levar o computador para sua casa e tiver conexão com a Internet lá, seus dados também seriam sincronizados de lá. Sendo baseado no ambiente KDE sem modificações drásticas de sua interface, o ambiente Classmate Metasys possui funcionalidades disponíveis em sistemas *desktop* convencionais, como listado a seguir:

- navegador de Internet;
- suíte de escritório;
- *software* de mensagem instantânea;
- *software* de comunicação de voz pela Internet;
- editor de foto;
- *software* para visualização de *webcam*;
- atualizações automáticas feitas com a autorização do usuário;
- jogos educacionais.

Alguns aplicativos específicos foram desenvolvidos com foco no contexto de uso do projeto UCA, ou seja, com a presença de um servidor na escola, com a preocupação de possível furto de máquinas e eventuais aplicações em sala de aula. Alguns destes aplicativos rodam no servidor da escola e alguns nos computadores pessoais; a lista destes segue:

[1] vide `http://www.syst.com.br/`

- *software* para gerenciamento em sala de aula; uma aplicação que permite professores e alunos interagirem entre si. É possível enviar comandos do computador do professor para o aluno e vice-versa, além de permitir ao professor a aplicação de testes e monitorar as atividades propostas em aula;

- *software* de sincronização de arquivos entre os *laptops* e o servidor da escola;

- *software* para leitura da caneta digital, um equipamento distribuído com *laptops* Classmate. Esta caneta, aliada ao aplicativo específico, permite reconhecimento de caracteres e reproduz os traços na tela do *laptop* (INTERNATIONAL SYST, 2009);

- sistema de controle antifurto com uso de certificados digitais de autorização com prazos configuráveis para expirar;

- sistema de controle de acesso a programas e páginas na Internet; o aplicativo também pode ser configurado para armazenar um histórico das informações do computador (como páginas acessadas, arquivos modificados e programas abertos). Ao conectar-se na rede da escola, o controle de acesso é definido pela configuração estabelecida pelo professor ou aquela válida para a escola como um todo, dependendo da situação. Em casa, a política de restrições é estabelecida pelos pais.

O ambiente ainda permite o compartilhamento de arquivos através de rede Mesh criada entre computadores Classmate não conectados a uma rede sem fio, embora esta funcionalidade não esteja disponível em todos as versões de Classmate. Alguns dos pontos apresentados acima fazem parte das exigências colocadas na licitação da compra dos computadores do projeto; para maiores detalhes, veja o anexo A. Versões mais novas do *laptop* usam processadores Intel Atom e têm tela sensível ao toque (*touch screen*), que também são suportadas por este sistema operacional (INTERNATIONAL SYST, 2009).

A figura 2.6 mostra uma captura de tela do ambiente Metasys. Como é possível verificar na figura, o ambiente não rompe com a metáfora de escritório predominante nos computadores atualmente, mesmo tendo incluído um conjunto de aplicativos e jogos educacionais, ferramentas para sincronização de arquivos e controle de acesso.

A empresa International Syst também mantém uma loja *online* para venda e distribuição de aplicativos para o ambiente Metasys e acessórios, além de manter canais virtuais de treinamento.

Figura 2.6: Visão geral do ambiente Metasys (retirado de `http://www.metasys.com.br/`).

2.3.3 MeeGo

Historicamente, os computadores surgiram para fazer processamento de grandes volumes de dados e cálculos complexos. O paradigma de interface dos ambientes convencionais de computadores *desktop* são voltados para trabalhadores de escritório, também por motivação histórica. De forma a se adequar ao contexto da Aprendizagem criativa apoiada por computador, o ambiente Sugar foi uma das iniciativas que buscou romper com o paradigma de interface vigente nos computadores *Desktop* e, depois de sua implementação, outras iniciativas surgiram no sentido de inovar as interfaces gráficas.

Da mesma forma que o Sugar, o ambiente MeeGo está organizado em camadas acima do núcleo Linux. Uma característica deste ambiente é a interface baseada na comunicação por rede (LINUX FOUNDATION, 2010). No ambiente Sugar, o paradigma de interface gráfica foi pensado na colaboração. Desta forma, muitos aplicativos, bibliotecas e recursos disponíveis nas diversas distribuições GNU/Linux estão disponíveis para este ambiente gráfico. Entre as bibliotecas disponíveis, estão o D-Bus (biblioteca para comunicação entre processos) e o Telepathy (biblioteca para uso de protocolos de comunicação diversos), que juntas permitem a construção de aplicativos colaborativos tal qual o Sugar.

O projeto MeeGo é uma iniciativa de *software* livre resultante da junção do projeto Moblin, liderado pela Intel, e o projeto Maemo, da Nokia, em uma única atividade *open source*. O MeeGo integra a experiência e as competências dos dois ecossistemas de desenvolvimento significativo, versado em comunicações e tecnologias de computação. O projeto Meego coloca esses dois pilares como a base técnica para as plataformas de próxima

geração e usos no espaço móvel e plataformas de dispositivos (LINUX FOUNDATION, 2010). Algumas características do desenvolvimento desta plataforma são:

- otimizações de desempenho e melhoria de características que permitam aplicações ricas graficamente e computacionalmente, além do desenvolvimento orientado a serviços conectados;
- não comprometer normas da internet para prover as melhores experiências web;
- interface de usuário Fácil de usar, flexível e poderosa / ambiente de desenvolvimento de aplicativo baseado em Qt;
- organização do projeto *open source* gerenciado pela Linux Foundation;
- núcleo Linux otimizado para o tamanho e as capacidades das plataformas de dimensões reduzidas e dispositivos móveis, mas oferecendo ampla compatibilidade com aplicativos projetados para Linux.

Atualmente, o ambiente Meego é desenvolvido para plataformas como *netbooks*, dispositivos eletrônicos portáteis, dispositivos de *infotainment* em veículos, TVs ligadas à rede, e os telefones *smartphone*, mas ainda não é amplamente utilizado comercialmente. Todas estas plataformas têm características comuns em comunicações, aplicações e serviços de Internet em um formato portátil ou pequeno (LINUX FOUNDATION, 2010). O projeto Meego deverá expandir o suporte a novas plataformas como novos recursos são incorporados e os fatores de forma nova surgindo no mercado.

O ambiente Meego usa uma arquitetura Unix mais tradicional com base no servidor gráfico X11 e a biblioteca gráfica Qt 4.6. Esta combinação foi concebida justamente para atrair mais desenvolvedores habituados a outros ambiente GNU/Linux já disponíveis para plataformas *desktop* (VAUGHAN-NICHOLS, 2010). A figura 2.7 mostra a arquitetura do sistemas MeeGo como uma pilha de *software*.

Olhando a arquitetura em camadas, é possível ver como se procura uma camada de abstração única para os serviços disponíveis no ambiente. Além disso, a interface busca um nível de abstração diferenciado, pois a implementação da interface é dependente do *hardware*: o mesmo sistema operacional poderia ser instalado em celulares ou em *neetbooks*. O ambiente Meego é construído sobre kernel Linux e usa DeviceKit e udev para trabalhar com dispositivos de *hardware*, assim, suporta uma gUPnP (um framework

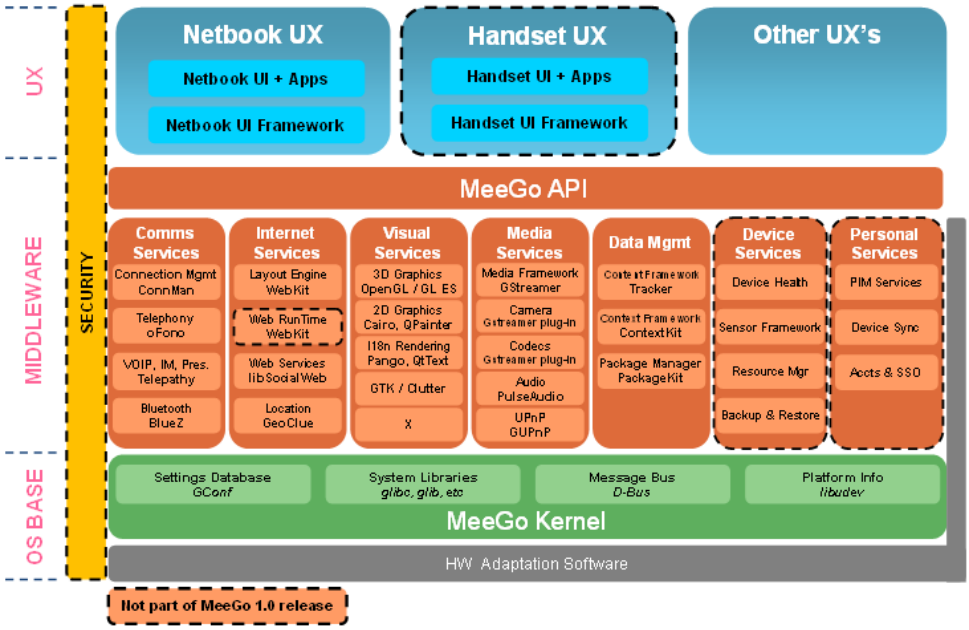

Figura 2.7: Visão geral da arquitetura do ambiente MeeGo (retirado de Linux Foundation (2010)).

universal de Plug and play). Para conectividade de voz e dados, Meego usa o gerenciador de conexões Connman, Ofono na pilha de telefonia, e a biblioteca BlueZ Bluetooth.

A distribuição Metasys Meego está disponível para computadores Intel Classmate, tornando-o um candidato para entrar nas próximas distribuições do projeto UCA, uma vez que estes *laptops* já se encontram distribuídos nas escolas participantes do projeto. Como a figura 2.8 permite ver, a interface possui uma metáfora orientada a conectividade, um aspecto bastante salientado nos fóruns de educação e informática.

2.4 Modelos de desenvolvimento de software

Esta seção aborda alguns dos paradigmas vigentes para o desenvolvimento de *software* em aprendizagem. As práticas das comunidades de *software* livre também são estudadas por estas estarem amplamente ligadas à aplicação de tecnologias na Educação. A pesquisa teve interesse particular em verificar quais as relações entre os processos presenciais e os processos colaborativos no desenvolvimento de *software*.

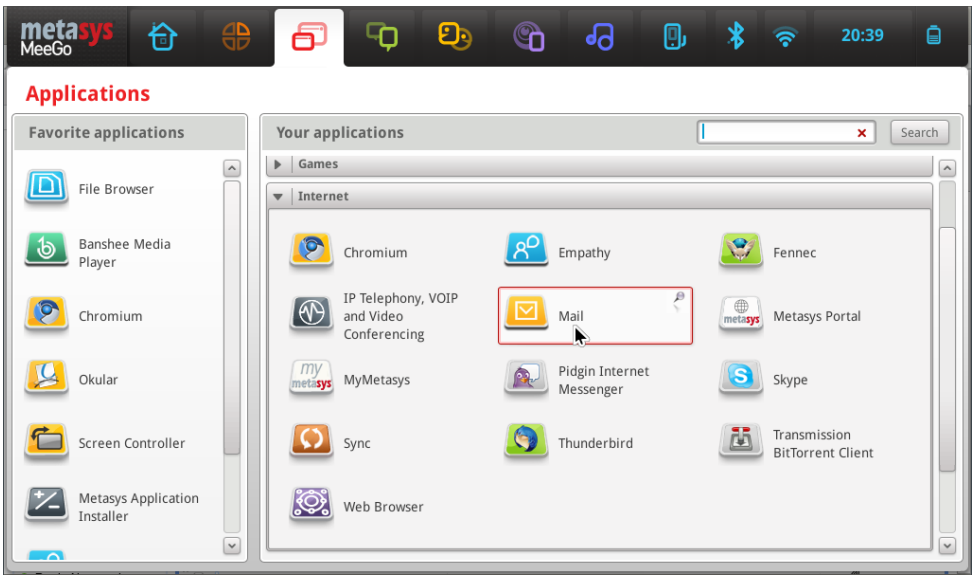

Figura 2.8: Visão da interface do ambiente MeeGo (retirado de `http://www.metasys.com.br/`).

2.4.1 Engenharia sócio-cognitiva

O desenvolvimento iterativo é uma abordagem para o desenvolvimento de *software* baseada no uso de ciclos. De maneira geral, o desenvolvimento de um aplicativo começa com as etapas de planejamento de um projeto, passando pelas etapas de desenvolvimento e lançamento do produto. Quando o produto é lançado, podem haver de testes de produtos e usuários, e estes resultados são eventualmente usados para o próximo ciclo de lançamento.

Neste método, os desenvolvedores devem estar cientes do fato dos produtos e aplicativos desenvolvidos não serem finalizados em uma rodada. Ao invés de tentar prever todos os possíveis problemas e necessidades dos usuários, uma série de iterações permite refinar e melhorar gradativamente o produto, de forma a adequar o produto às necessidades observadas. Uma das principais vantagens do desenvolvimento iterativo é permitir respostas mais rápidas a problemas e necessidades em constante mudança, pois as ações de mudança (como reconstrução, reversão e refinamentos) são parte intrínseca do processo (LARMAN, 2003).

O projeto de tecnologias interativas têm recebido grande atenção da comunidade científica. As estratégias costumam considerar o usuário como um dos componentes mais importantes do sistema, tornando o ser humano mais importante que o sistema em si. Uma das possíveis abordagens de projeto é considerar a tecnologia em seu contexto de uso. Procurando estender a proposta de Norman para o termo "Engenharia Cognitiva",

Sharples et al. (2002) procuraram analisar o contexto no qual a tecnologia está inserida e como as pessoas interagem com ela a fim de projetá-la. Trata-se, portanto, de um sistema sócio-técnico (SHARPLES; CORLETT; WESTMANCOTT, 2002). O método desenvolvido nesta abordagem chama-se Engenharia sócio-cognitiva, e está pautada na experiência prévia dos autores em projetos de Aprendizagem Móvel.

O principal objetivo da Engenharia sócio-cognitiva é justamente analisar as complexas interações entre pessoas e sistemas computacionais e transformar esta análise em sistemas usáveis e úteis (SHARPLES; CORLETT; WESTMANCOTT, 2002). O método desenvolvido tem sido usado ao longo de projetos ligados ao uso de tecnologia como ferramenta de apoio ao trabalho e à aprendizagem ao longo de mais de 25 anos (SHARPLES et al., 2002; SHARPLES, 2000).

A proposição parte do princípio de que as tecnologias devem ser projetadas numa abordagem centrada no usuário. Sharples et al. (2002, p. 311) consideram os usuários como uma boa fonte de informações para o projeto da tecnologia, além de poderem ser parceiros neste processo. Entrevistas com usuários seriam capazes de elucidar problemas a respeito das condições atuais de trabalho ou diferenças de ponto de vista de usuários de diferentes perfis.

A figura 2.9 mostra uma visão geral do processo proposto na Engenharia sócio-cognitiva, mostrando sua divisão em 2 etapas principais: a primeira estabelece as condições gerais de projeto e como as pessoas interagem e trabalham com suas ferramentas atuais; a segunda projeta de fato a nova tecnologia.

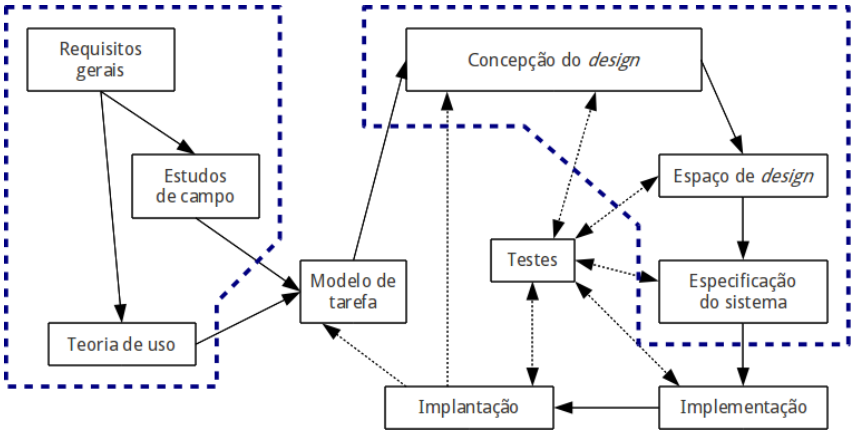

Figura 2.9: Processo de desenvolvimento da Engenharia sócio-cognitiva.

O método propõe que o desenvolvimento comece com os requisitos gerais, definindo

logo no início qual o tipo de atividade a ser trabalhada com a tecnologia em projeto e quais as principais restrições de desenvolvimento, como o prazo. O produto do estudo do contexto aliado a um embasamento teórico sobre os processos cognitivos e sociais é um modelo de tarefas. O primeiro modelo de tarefa deve sintetizar como as pessoas envolvidas agem dentro do contexto atual e como elas interagem entre si, além de mapear os contextos (SHARPLES; CORLETT; WESTMANCOTT, 2002). Este modelo também deverá representar como as pessoas externalizam seu trabalho (como notas e diagramas), as regras envolvidas no exercício das atividades e a terminologia geral. O ciclo de desenvolvimento pode, então, ser resumido da seguinte forma:

1. Levantamento geral de requisitos e estudos de campo;
2. Modelagem de tarefa;
3. Design e especificação;
4. Implementação do sistema;
5. Implantação no local de uso.

As etapas 3 a 5 são repetidas indefinidamente; ficando a critério da equipe de desenvolvimento quantas vezes o ciclo será percorrido. O estabelecimento do primeiro modelo de tarefas faz a ligação entre a primeira e a segunda fase do processo da Engenharia sócio-cognitiva. A partir daí, da-se um processo iterativo que tem o particularidade de incluir **testes em todas as etapas**, podendo inclusive alterar o modelo de tarefa previamente estabelecido, conforme ilustrado na 2.9. Embora o ciclo seja baseado num processo **iterativo** convencional, como os descritos por Larman e Basili (2003), o seu diferencial está em dar para análise de fatores cognitivos e organizacionais a mesma ênfase dada nas especificações da tarefa e do *software* (SHARPLES et al., 2002; SHARPLES; CORLETT; WESTMANCOTT, 2002).

A saída do ciclo (na etapa 5) é uma tecnologia com recomendações de uso, projetadas para serem adequadas ao contexto atual de interação com a tecnologia e com as práticas de trabalho correntes. Ao implementar esta tecnologia no seu local de uso para os atores envolvidos no processo, o sistema sócio-técnico anterior será *transformado*. Esta transformação cria novas atividades a serem suportadas e novos problemas a tratar, essencialmente pelo fato de as práticas mudarem com a introdução do sistema (SHARPLES; CORLETT; WESTMANCOTT, 2002).

2.4.2 Métodos ágeis

Os métodos ágeis de desenvolvimento de *software* surgiram a fim de superar alguns problemas dos métodos tradicionais da Engenharia de *Software*. Muitas organizações têm um histórico na adoção de métodos mais com processos extremamente burocráticos ao invés de adotar processos cujo foco está na qualidade do *software* desenvolvido (GOLDMAN et al., 2004). Em muitos projetos analisados na literatura, há mudanças de requisito, escopo e tecnologia sobre as quais as equipes de desenvolvimento não tem controle ou o planejamento estabelecido não tem condições de dar conta (HIGHSMITH; COCKBURN, 2001). Nos anos de 1990, surgiram modelos de desenvolvimento que preferem eliminar uma etapa de especificação muito detalhada, criando abordagens com especificações iniciais mais soltas e análise evolutiva através do projeto (LARMAN; BASILI, 2003).

Por anos, o modelo em cascata foi criticado pela sua falta de flexibilidade, de forma que os modelos ágeis buscaram prover alternativas a um modelo considerado "pesado" (LARMAN; BASILI, 2003; GOLDMAN et al., 2004). Uma abordagem capaz de superar este problema é, justamente, preparar uma equipe para as mudanças inevitáveis de um projeto de *software*. Ao invés de tentar reduzir os custos buscando prever todas as mudanças possíveis no decorrer do desenvolvimento, os métodos ágeis assumem que a melhor estratégia é estar preparado para responder às mudanças, trazendo assim redução nos custos (HIGHSMITH; COCKBURN, 2001). Williams e Cockburn (2003) afirmam que os métodos ágeis partem do princípio que o desenvolvimento de *software* deve estar inspirado nos processos empíricos de Engenharia, ou seja, devem haver ciclos curtos de **inspeção e adaptação** com **realimentação** rápida no processo.

Vários são os modelos de desenvolvimento agrupados sob o nome de métodos ágeis, como Scrum, Crystal, Programação Extrema (*Extreme Programming* - XP), *Feature-Driven Development* (FDD), Programação Pragmática, entre outros. Todos eles têm em comum a característica de operação em ciclos curtos, motivo pelo qual Larman e Basili (2003) os posiciona como derivados de métodos iterativos e incrementais. A fim de criar um consenso a respeito de discordâncias dos modelos vigentes de Engenharia de *Software*, representantes de diversos métodos, ligados à indústria de *software* ou à academia, criaram o **Manifesto Ágil** (BECK et al., 2001) em 2001. O manifesto defende os seguintes princípios:

- indivíduos e interações são mais importantes que processos e ferramentas;

- *software* em funcionamento é mais valioso que documentação abrangente;
- colaboração com o cliente tem mais valor que negociação de contratos;
- responder a mudanças é melhor que seguir um plano.

Seguir estes princípios é considerado fundamental para o sucesso em projetos de *software* para os defensores dos modelos ágeis (GOLDMAN et al., 2004). As regras destes métodos são consideradas generativas, ou seja, são apenas um conjunto mínimo de diretrizes a serem seguidos sempre, de forma que a equipe deve se basear nelas para tomar decisões (HIGHSMITH; COCKBURN, 2001). Muitos consideram os métodos ágeis como os modelos de desenvolvimento mais adequados para projetos com restrições de tempo e com muitas mudanças de requisito (GOLDMAN et al., 2004).

Embora os métodos em si tenham diferenças quanto a procedimentos, todos eles criticam abordagem com documentação em excesso e etapas bloqueadas entre si, além de apoiar a repetição das etapas de desenvolvimento (LARMAN; BASILI, 2003). Os ciclos de desenvolvimento (também tratados como *iterações*), entretanto, não devem estar amplamente espaçados entre si; embora cada método tenha a sua especificidade, a duração típica de cada iteração fica entre 2 e 6 semanas (HIGHSMITH; COCKBURN, 2001). Williams e Cockburn (2003), Larman e Basili (2003) afirmam que os métodos ágeis não defendem princípios inéditos na história da Engenharia de *Software*, a novidade é apenas juntar os princípios sob arcabouços (já que há mais de 1 método ágil) e a divulgação mais veemente destes princípios. Mesmo assim, as condições adequadas de funcionamento dos métodos ágeis são conhecidas; sendo que estes métodos têm sua aplicação notoriamente bem-sucedida restrita a *sistemas não-críticos* com *requisitos em constante mudança* e a equipes relativamente *pequenas*, altamente *competentes* e com alocação *presencial* (WILLIAMS; COCKBURN, 2003).

2.4.2.1 Programação Extrema (XP)

Tratando de maneira simplificada, a maior caraterística dos métodos ágeis é ter ciclos de desenvolvimento curtos no tempo (WILLIAMS; COCKBURN, 2003). Assim sendo, a Programação Extrema pode ser considerada um **método de desenvolvimento iterativo** com um horizonte de planejamento curto, com ciclos de desenvolvimento de 1 ou 2 semanas, tipicamente (LAYMAN et al., 2006). Beck (1999) faz uma comparação bastante

resumida entre os modelos cascata, iterativo simples (como o modelo em espiral) e a proposta da programação extrema, mostrada na figura 2.10.

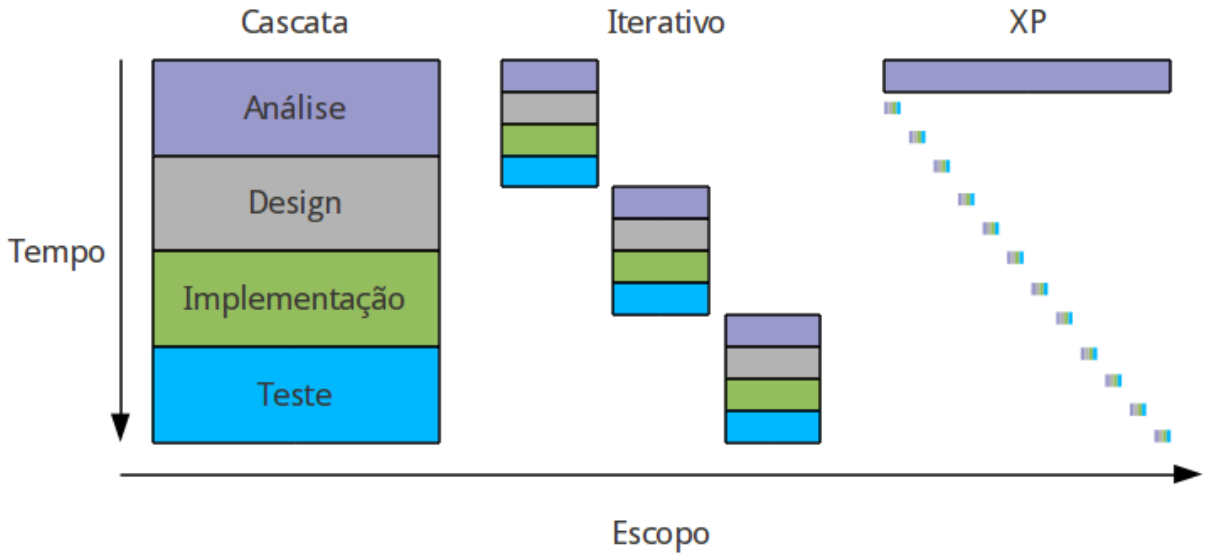

Figura 2.10: Comparação entre os ciclos de desenvolvimento dos modelos cascata, espiral e XP (adaptado de Beck (1999)).

A figura 2.10 ilustra a distribuição das etapas em diferentes modelos de desenvolvimento, comparando os eixos escopo (*scope*) e tempo (*time*). A proposta em modelos iterativos é, justamente, encurtar as etapas de planejamento, análise, *design*, implementação e teste, executando-as repetidamente. XP, por outro lado, leva esta característica ao extremo, usando espaços da ordem de dias para chegar à etapa de implementação. Beck (1999) também assinala para a redução dos custos de desenvolvimento ao executar todos estes passos um pouco por vez, fazendo a programação extrema ser bastante adequada para situações com constantes mudanças no projeto.

A programação extrema não costuma ser descrita como passos sequenciais como é praxe em modelos de Engenharia de *Software*, ao invés disso, Beck e Andres (2004) colocam a programação extrema como uma composição de **valores**, **princípios** e **práticas**. Os valores são um conjunto de critérios abstratos para a equipe balizar seus julgamentos e decisões; os valores listados originalmente são comunicação, simplicidade, realimentação (o termo original em inglês é *feedback*), coragem e respeito, aos quais podem ser adicionados outros valores de acordo com a equipe. As práticas representam ações relacionadas ao cotidiano dos desenvolvedores, enquanto os princípios são um conjunto de diretrizes específicas para ajustar as práticas aos valores. Nem sempre os valores podem ser diretamente traduzidos em práticas, daí a necessidade da existência dos princípios e sua

aplicação é justificada nos momentos em que as práticas propostas não servem a uma situação específica (SATO, 2007). Os princípios de XP são:

- humanidade;
- economia;
- benefício mútuo;
- auto semelhança;
- melhoria;
- diversidade;
- reflexão;
- fluxo;
- oportunidade;
- redundância;
- falha;
- qualidade;
- passos pequenos;
- responsabilidade aceita.

Na figura 2.11, apresenta-se a relação entre os valores, os princípios e as práticas da programação extrema proposta pela AgilCoop [2], ficando mais evidente que nem todos os valores têm tradução direta em práticas do cotidiano.

Embora as práticas estejam diretamente ligadas ao dia-a-dia da desenvolvimento de *software*, sua aplicação ou não depende de cada situação. Uma mudança no contexto do projeto pode acarretar numa troca de prática, mas os valores não necessariamente precisam mudar para nesta situação (BECK; ANDRES, 2004). Beck e Andres propõem que as práticas sejam escolhidas de acordo com cada equipe, ou seja, cada equipe tem a liberdade de adaptar o método a si própria, optando pelo conjunto de práticas que

[2] vide `www.agilcoop.org.br`.

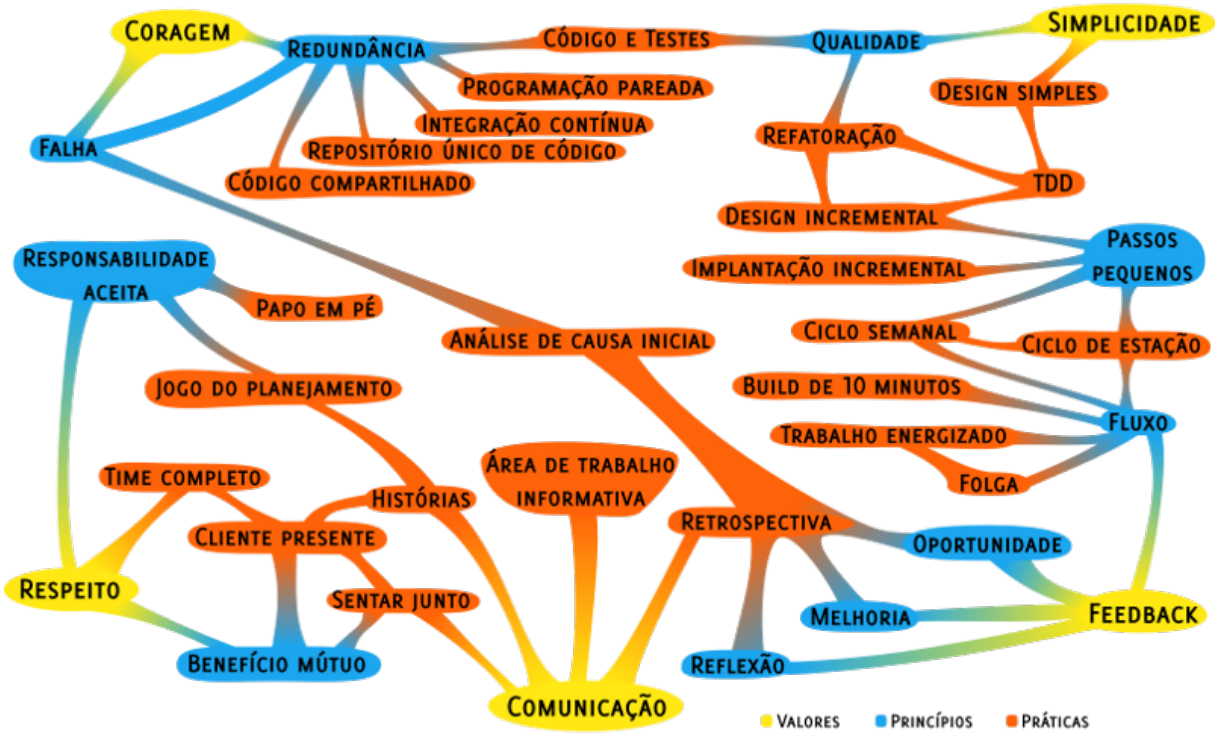

Figura 2.11: Relação entre os valores, os princípios e as práticas da Programação Extrema (adaptado de `http://www.agilcoop.org.br/`).

mais fizer sentido. Além disso, as práticas são divididas em duas categorias, `primárias` e `corolárias`. As primárias são as práticas mais simples e devem produzir resultados imediatos mesmo quando aplicadas individualmente; já as corolárias são intrinsecamente mais complexas e os resultados de sua adoção dependem do domínio das práticas primárias. As práticas primárias são detalhadas a seguir:

- sentar junto: o espaço de trabalho deve ser amplo e aberto;
- time completo: a equipe deve ter os profissionais com as competências necessárias para o desenvolvimento;
- área de trabalho informativa: o espaço onde ficam os desenvolvedores deve expor informações que permitam um observador entender o estado do projeto;
- trabalho energizado: o ritmo de trabalho deve seguir um ritmo sustentável, de forma que horas-extras sejam a exceção e não a regra;
- programação pareada: os programadores trabalham sempre em pares;

- histórias: o cliente do projeto descreve as funcionalidades para o sistema em *cartões de história*, dando prioridade a eles, e os desenvolvedores, por sua vez, devem estimar qual o esforço necessário para implementar um cartão tão logo ele seja criado;

- ciclo semanal: as atividades devem ser planejadas semanalmente, sendo necessário refletir sobre resultados anteriores, priorizar as histórias da semana e quebrar histórias em tarefas;

- ciclo trimestral (ou de estação): o plano trimestral está associado ao lançamento de uma versão do sistema (também designada por *release*). As histórias do trimestre são agrupadas por um tema (observado do ponto de vista do cliente) e a equipe de desenvolvedores deve identificar gargalos de implementação e reservar algum tempo para reparos;

- folga: o planejamento deve incluir tarefas de baixa prioridade para o caso de haver atrasos. Este tempo deve existir por conta do caráter subjetivo das estimativas feitas para as histórias;

- *build* de 10 minutos: um sistema capaz de ser compilado e testado *automaticamente* e *em pouco tempo* vai ser compilado e testado com mais frequência, aumentando as oportunidades de se detectarem problemas;

- integração contínua: as alterações produzidas por uma dupla de desenvolvedores sempre deve estar disponível para o restante da equipe no menor tempo possível e com garantia de funcionamento (preferencialmente em não mais do que algumas horas);

- testar antes de escrever código: os testes automatizados devem ser escritos antes de se programar uma nova funcionalidade;

- *design* incremental: ao manter a arquitetura mais simples possível, a equipe minimiza os custos para se adaptar a mudanças. O *design* incremental busca sempre manter baixa a complexidade e alta a flexibilidade para as adaptações necessárias.

Segundo Beck (1999), a dinâmica proposta pela programação extrema é a seguinte:

1. cliente (auxiliado pelos desenvolvedores) escolhe as funcionalidades (tratadas de *histórias* no jargão deste modelo) para a próxima iteração,

2. desenvolvedores transformam as histórias escolhidas em tarefas,
3. desenvolvedores escrevem testes automatizados para provar que uma tarefa foi cumprida,
4. trabalhando em pares, os desenvolvedores devem escrever código que passam nos testes,
5. desenvolvedores evoluem o *design* de forma a mantê-lo o mais simples possível.

Conforme apresentado anteriormente, a programação extrema não possui uma etapa longa a exaustiva de coleta de requisitos ou produção de documentação de *design* antes do início da implementação, pois entende-se que a comunicação constante entre os participantes supre estas necessidades (BECK; ANDRES, 2004). Uma característica marcante das equipes trabalhando segundo a programação extrema é a necessidade de vias para comunicação informal, mesmo assim, Layman et al. (2006) indica que, dentro de certas condições, o modelo de desenvolvimento pode ser usado por equipes distribuídas (o modelo de desenvolvimento distribuído é tratado na seção 2.4.3.2). O estudo conduzido por Layman et al. relata que o sucesso está em estabelecer condições favoráveis de comunicação entre as pessoas envolvidas. Os fatores de sucesso são 4:

- ter um **representante do cliente** permanente para tornar a tomada de decisões mais eficiente e a definição dos requisitos mais clara. É importante que este representante possa tomar decisões de maneira autônoma com relação a funcionalidades e escopo, estar interessado no projeto e sempre estar prontamente acessível;
- ter um **representante da outra equipe** fisicamente presente no dia-a-dia da equipe local para ajudar a estabelecer um canal de comunicação; o distanciamento entre o gerenciamento e os desenvolvedores gera a necessidade deste papel cujo propósito é trabalhar próximo às duas equipes. É preferível que esta pessoa saiba se comunicar em todos os idiomas envolvidos, pois ela atuará como **ponte** entre as equipes;
- **ciclos de comunicação rápidos e assíncronos** têm impacto positivo no estabelecimento de um ambiente de desenvolvimento focado e de uma boa relação de confiança e compromisso. No caso da comunicação face-a-face não ser aplicável, uma lista de *emails* tem grande chance de prover respostas úteis, rápidas e conclusivas;

- prover **visibilidade às informações de processo e produto** para os membros da equipe ajuda a melhorar o processo de controle e a efetividade do planejamento. Para este fim, o uso diário de ferramentas *online* de gerenciamento de projeto é essencial, pois tornam o registro e monitoramento do estado do projeto disponível em praticamente qualquer lugar.

2.4.3 Desenvolvimento colaborativo e distribuído

É cada vez mais comum o surgimento de membros de equipes trabalhando nos mesmos projetos de forma geograficamente distribuída em diversos setores do mundo corporativo, inclusive na indústria de *software*. Muitos fatores contribuem para impulsionar esta prática, mas podem ser destacados o fato de os profissionais habilidosos poderem estar em diferentes locais ou até mesmo questões de custo (PRIKLADNICKI; AUDY; SHULL, 2010). Embora a prática do desenvolvimento distribuído de *software* tenha ganho bastante destaque na última década, as empresas que optaram pela sua adoção enfrentam desafios de engenharia, como gerenciar custos e prazos. Os desafios também são de ordem social, política e cultural, influindo nos modos de conceber, projetar e testar estes aplicativos (PRIKLADNICKI; AUDY, 2010). Os modelos de referência para estes processos ainda são pouco desenvolvidos (PRIKLADNICKI; AUDY; SHULL, 2010).

Em termos de relacionamento entre empresas, são duas as estratégias mais comuns: *offshore outsourcing* (uma empresa externa sendo contratada para desenvolver *software* em outro país) e *internal offshoring* (uma filial da mesma empresa desenvolve o *software* em outro país). Os estudos disponíveis na área de desenvolvimento distribuído ainda não são extensivos, há poucos modelos de referência para o nível de projeto e os existentes nem sempre foram testados extensivamente na prática.

A maior parte dos estudos, entretanto, não aborda aspectos técnicos, mas sim possíveis estratégias de negócio para auxiliar decisões em nível corporativo (PRIKLADNICKI; AUDY, 2010). Desta forma, parece haver um sentimento comum de direcionar os modelos vindouros para a seleção das estratégias pertinentes a um projeto, como: **avaliar** a capacidade dos indivíduos envolvidos, **estender os modelos de maturidade** de desenvolvimento de *software* (como o CMMI) e **criar um conjunto de boas práticas** de Engenharia de *Software* específico para este campo.

Gutwin, Penner e Schneider (2004) aproximam o **desenvolvimento colaborativo** (ou

seja, aquele praticado por *software* livre) do desenvolvimento distribuído de *software* praticado na indústria, pois inúmeros programadores trabalham em diversos lugares do mundo desenvolvendo *software* em conjunto, mantendo a consciência dos outros desenvolvedores e seu trabalho. Além disso, Scacchi (2007) considera a estratégia colaborativa uma maneira de construir, entregar e sustentar sistemas de *software* de grande porte de maneira global. Mesmo com as dificuldades associadas à distância e à virtualidade do relacionamento entre os desenvolvedores, há diversos projetos de *software* livre bem-sucedidos. Uma das dificuldades no trabalho distribuído em comparação com o trabalho presencial é que muitas vezes as informações implicitamente disponíveis para equipes presenciais não estão disponíveis para membros remotos (GUTWIN; PENNER; SCHNEIDER, 2004).

2.4.3.1 Estrutura de projetos de software livre

Os projetos de *software* livre são comumente chamados de comunidades (ou, mais especificamente, de comunidades online) por duas características: organização em torno de um objetivo comum e respeito aos princípios e às regras dos projetos de *software* livre (BARCELLINI et al., 2008). Para a *Free Software Foundation* (FSF), um *software* pode ser considerado como *software* livre se o usuário tiver a liberdade de rodar, copiar, estudar, distribuir, modificar e aperfeiçoar o *software* [3]. Além disso, estas comunidades são identificadas como meritocráticas, tendo sua hierarquia organizada segundo habilidades técnicas. Esta hierarquia tem reflexo no acesso ao código-fonte, pois, enquanto todos têm acesso à leitura destes arquivos, apenas alguns desenvolvedores enviam alterações para o repositório central. Seguindo a nomenclatura apresentada anteriormente, o privilégio o envio de alterações cabe ao líder de projeto, aos administradores e aos desenvolvedores. A contribuição dos usuários se dá através de testes, ideias de funcionalidades, documentação e tradução (BARCELLINI et al., 2008; SCACCHI, 2007).

O artigo de Gutwin, Penner e Schneider (2004) explora como se dá a colaboração entre os membros de comunidades de *software* livre. O artigo aponta alguns mecanismos pelos quais os participantes se mantém informados do que acontece na comunidade: listas de discussão, canais de chat, logs do repositório de código. O envolvimento dos desenvolvedores nas comunidades de *software* livre têm algumas peculiaridades apontadas por alguns estudos. Muitas vezes, os desenvolvedores são usuários dos próprios aplicativos, fazendo o termo "usuário final" ganhar uma conotação diferente da usual (BARCELLINI et al.,

[3] vide `http://www.gnu.org/philosophy/free-sw.html`

2008). A motivação dos participantes muitas vezes está ligada à "fama" proporcionada pela visibilidade do projeto; mas também tem a ver com a possibilidades de melhorar suas habilidades e competências técnicas e por poderem exercer diferentes papéis dentro da comunidade (GUTWIN; PENNER; SCHNEIDER, 2004; SCACCHI, 2007).

A organização das comunidades é fortemente baseada em escala da habilidade (muitas vezes descrita como *skill-based meritocracy*) e os papéis mais importantes rendem mais fama ao desenvolvedor. É bastante comum também um núcleo pequeno ser responsável pelo maior parte do desenvolvimento, controlando a arquitetura e direcionando o calendário de desenvolvimento (faz o chamado *roadmap*). Este núcleo pequeno (ou membros dele) muitas vezes está ligado a alguma empresa ou universidade, mas mesmo assim não há a noção de autoridade dentro da comunidade (SCACCHI, 2007). Estudos sobre o tema indicam forte utilização de canais online para comunicação e registro, como wikis, listas de email, fóruns e redes sociais (GUTWIN; PENNER; SCHNEIDER, 2004; BARCELLINI et al., 2008). Scacchi (2007) também aponta para a eventual substituição de recursos formais da Engenharia de *Software* tradicional (como documentos de especificação de requisitos e diagramas UML) por recursos chamados de informais. Estes recursos são quase sempre recursos Web e têm caráter descritivo, sendo geralmente narrativos; via de regra estes informalismos são armazenados nos canais online citados anteriormente.

Scacchi (2007) argumenta que são necessários alguns recursos sócio-técnicos para o funcionamento efetivo de um projeto de *software* livre. Entre estes recursos estão os pessoais, as crenças, as informalidades, o reconhecimento e as habilidades. Os conflitos têm um caráter diferente da Engenharia de *Software* tradicional pelo caráter distribuído. Um dos desafios de gerenciar e administrar equipes envolvidas em projetos de *software* livre ou em desenvolvimento distribuído de *software* é justamente contornar conflitos, seja usando ferramentas eletrônicas, seja através de recursos organizacionais (SCACCHI, 2007; CASEY, 2010). Os conflitos são referidos, neste caso, como decisões de arquitetura, alocação de recursos humanos ou financeiros e discordâncias técnicas. A principal diferença, entretanto, está na noção de autoridade dentro das comunidades de *software* livre: a maioria dos desenvolvedores participa de forma voluntária, tornando os sistemas de controle de versões e atribuição de tarefas (como um *issue/bug tracker*) ferramentas muito importantes do gerenciamento de conflitos. Além disso, as comunidades apostam no planejamento das versões de lançamento ao público e discussão online como modos de evitar conflitos.

A organização destas equipes pode ser vista como a de uma equipe virtual, trazendo um lógica centralizada para um grupo distribuído. A leitura de Scacchi (2007) e Casey (2010) evidencia diversas semelhanças entre os processos de gerenciamento de equipes virtuais, mesmo quando os contextos são distintos. Tal qual apontado por Gutwin, Penner e Schneider (2004), Casey (2010) indica a existência do impacto da distância entre as pessoas, barreiras de idioma, diferenças culturais e de fuso-horário como fatores negativos para o gerenciamento global das equipes virtuais. Neste sentido, Casey (2010) lista quatro fatores como variáveis-chave para o sucesso do desenvolvimento de projetos distribuídos: **comunicação**, **cooperação**, **coordenação** e **visibilidade**. Assim sendo, os esforços para mitigar os impactos negativos nestas variáveis têm muito a ver com os enunciados para as comunidades de *software* livre:

coordenação: coordenar uma equipe virtual exige planejamento realístico e avaliação de riscos, como no gerenciamento de uma equipe presencial, além da divisão do trabalho segundo critérios técnicos e experiência dos membros. Estas exigências para para coordenação são apontadas da mesma forma por Gutwin, Penner e Schneider (2004) ao tratar das comunidades de *software* livre;

visibilidade: é preciso fazer os membros das equipes virtuais terem acesso ao que outros membros não-locais estão fazendo e qual seu valor para o projeto. Por isso, articular os papéis e as responsabilidades claramente (membros das equipes devem conhecer os requisitos do seu desenvolvimento bem como seus prazos) e criar estruturas e calendários efetivos para as equipes conhecerem o trabalho das outras têm muita importância. A visibilidade é ressaltado em Gutwin, Penner e Schneider (2004) e Scacchi (2007) para o desenvolvimento colaborativo;

comunicação: adotar um vocabulário comum mínimo e escolher ferramentas de comunicação uniformes a todos os membros da(s) equipe(s) é um pilar necessário para o trânsito das informações. As comunidades de *software* livre em geral adotam listas de discussão e canais de *chat* para este fim (GUTWIN; PENNER; SCHNEIDER, 2004; SCACCHI, 2007);

cooperação: Casey (2010) chama a atenção para a importância do gerente da equipe encontrar maneiras de estimular o senso de equipe de seus membros, ou seja, a colaboração entre seus membros. Desta forma, o autor leva a crer que esta é uma situação a ser resolvida caso a caso, fazendo um paralelo com as observações

> de Gutwin, Penner e Schneider (2004) no sentido de que cada comunidade busca a melhor estratégia para si, buscando uma harmonia entre seus membros. Ainda assim, é sabido que grande parte dos desenvolvedores participa voluntariamente, cooperando por motivação ideológica, pelo desafio técnico ou pela "fama" (BARCELLINI et al., 2008; SCACCHI, 2007).

Um ponto particular dos projetos de *software* livre é sua evolução. Na Engenharia de *Software* tradicional, não se sabe ao certo qual é a interdependência usuário-desenvolvedor, mas os estudos sobre desenvolvimento de *software* livre indicam que usuários e desenvolvedores tendem a evoluir de maneira mutuamente dependente (SCACCHI, 2007). Esta evolução depende, entretanto, da formação de uma massa crítica para a continuidade sustentável do projeto, tipicamente ligada ao número de desenvolvedores envolvidos.

2.4.3.2 O processo de design no software livre

O trabalho de Barcellini et al. (2008) faz uma análise das características sócio-cognitivas das comunidades de *software* livre nas atividades de *design* dos projetos. O processo de *design* destas comunidades é tipicamente distribuído geograficamente, comunitário e assíncrono. Por outro lado, as características sociais destes projetos apontam para a presença de diferentes papéis, como líder, administrador e desenvolvedor. Os demais participantes podem ser chamados de usuários, embora o termo não se refira necessariamente ao "usuário final", já que a participação de profissionais da computação é bastante comum. A organização hierárquica da comunidade pode evoluir em função da interação entre os participantes.

Uma importante diferença entre o processo tradicional de *design* preconizado na Engenharia de *Software* e o processo das comunidades de *software* livre é justamente a assincronia. Ao analisar os processos de *design* colaborativo, Barcellini et al. (2008) indicam 3 possibilidades de interação: face-a-face (como em encontros presenciais e reuniões), mediada e síncrona (como em vídeo-conferência) e mediada e assíncrona (como em *email* ou fóruns *online*). Como os meios de comunicação mais usados em comunidades de *software* livre são justamente as listas de discussão, seu processo é classificado como mediado e assíncrono, mas pode assumir um caráter quase síncrono em algumas situações (em decisões de *design* consideradas críticas, as discussões entre os membros da comunidade tendem a ficar menos esparsas temporalmente, por exemplo).

Barcellini et al. (2008) também destacam a diferença entre as atividades típicas do *design* colaborativo presencial e aquele usado nas comunidades de *software* livre. Estas atividades podem ser de 3 tipos: **geração e avaliação** (atividades relacionadas ao processo de enunciar o problema e propor e avaliar alternativas para sua solução), **clarificação** (atividades nas quais os participantes constroem uma representação coletiva do *design*) e **gerenciamento** (atividades relacionadas à coordenação de pessoas e recursos e coordenação de discussões). Para o *design* presencial, as atividades mais importantes são as de geração e avaliação e as de clarificação; já para as comunidades de *software* livre, as atividades de *geração e avaliação* são mais importantes que as de clarificação, ficando esta última mais a cargo do líder do projeto. Esta afirmação tem relação com o fato de o líder assumir maior importância nas decisões de arquitetura e desempate de votações nas comunidades de *software* livre (GUTWIN; PENNER; SCHNEIDER, 2004; BARCELLINI et al., 2008).

O processo de *design* nas comunidades de *software* livre também é visto como descontínuo por sua característica distribuída, justamente por apresentar a barreira da distância, limites profissionais e organizacionais e assincronia temporal. Barcellini et al. (2008) também apontam para a tendência da indústria de *software* passar a adotar métodos das comunidades de *software* livre, pois cada vez mais o desenvolvimento se dá de maneira global, sendo, consequentemente, distribuído. Além disso, Liu (2005), Hawthorne e Perry (2005) indicam a necessidade de se melhorar a formação em Engenharia de Software, **incluindo no ensino** práticas relacionadas a participar e gerenciar equipes de desenvolvimento geograficamente distantes, com caraterísticas culturais distintas e fuso horário influenciando as horas de trabalho.

Analogamente aos interessados em *software* livre, existem também pessoas interessadas na aplicação destes conceitos na construção de circuitos. A iniciativa do *Open Source Hardware* vai na linha de fornecer orientações de como projetar *hardware* aberto. Atualmente, a definição exata do que se pode chamar de *hardware* livre ou não encontra-se em discussão [4]; mas os pontos principais da proposta tratam apenas de aspectos técnicos, como:

1. documentação, ou seja, uma pessoa deve ser capaz de desenvolver o mesmo produto usando as orientações disponíveis;

[4] há um rascunho disponível em `http://blog.makezine.com/archive/2010/07/open_source_hardware_-_oshw_draft_d.html`

2. *software*, isto é, quais os aplicativos de computador necessários para reproduzir o projeto;

3. licenciamento; permissões e proibições do uso do presente projeto e atribuições relacionadas a trabalhos derivados.

Tanto no *design* de *software* livre quanto no de *hardware* livre, as recomendações e métodos relacionados ficam isoladas no campo técnico, não trazendo a questão do uso da tecnologia para dentro do projeto. Entretanto, alguns projetos de *software* livre têm um número de usuários muito grande, configurando uma massa crítica capaz de passar a interferir nas decisões de projeto (SCACCHI, 2007), mais comumente nas decisões relacionadas à interface. Projetos como o do navegador Firefox podem ter diversos rascunhos de mudança na interface gráfica submetidos à votação antes da implementação. Dessa forma, é possível notar como o próprio usuário é capaz de interferir no projeto quando há algum tipo de insatisfação.

Por outro lado, a participação maciça de usuários com pouco conhecimento em programação em comunidades de *software* livre não é uma regra, fazendo os projetos terem uma abordagem de *design* muito técnica (YEATS, 2006). O trabalho de YEATS pesquisou o papel dos usuários finais (tratados por *end-users*) em comunidades de *software* livre. O estudo indica uma certa "preferência" por usuários que sejam capazes de dar contribuições de arquitetura ou escrevendo código, os usuários incapazes de contribuir nestas frentes teriam o papel de pagar para desenvolvedores implementarem seus desejos.

> I found that both Raymond and Stallman demonstrated a bias toward highly technological users that left more typical end users out of the community of developers who benefit from the open-source development process.
> (. . .)
> Less technologically sophisticated users are relegated to the status of second-class citizens—people who must pay developers to code for them because they can't meaningfully contribute to the open-source projects.
> (. . .)
> The results of the survey demonstrated that developers generally do not pursue the kind of usability evaluation techniques that utilize the expertise and knowledge of a usability professional. Instead, developers rely on the opinions of the people around them (other developers, family members, friends) to evaluate their interfaces (YEATS, 2006, p. 190).

Segundo a pesquisa, ao se preocuparem com usabilidade, os desenvolvedores não utilizam as técnicas consagradas pelos profissionais da área, avaliando as interfaces com

pessoas próximas e outros desenvolvedores. Ainda assim, Yeats também vê o projeto do navegador Firefox como uma exceção à menor importância dada aos usuários finais nas comunidades. Na opinião do autor, o motivo de um usuário típico adotar ou não um *software* livre nada tem a ver com fato de o aplicativo em si ser livre ou de código aberto, este é um aspecto que atrai **desenvolvedores**.

Esta abordagem que pouco leva em consideração o usuário final pode ser vista como uma barreira para a popularização do *software* livre, pois fatores mais visíveis para o usuário como desempenho e aparência têm mais importância que princípios de *software* livre. Porém, projetos agregadores de aplicativos, como o do ambiente Sugar ou do ambiente GNOME [5] e até mesmo o ambiente Mac OS X [6], costumam usar o artifício das recomendações de interface (em inglês, *Human Interface Guidelines*) para atacar o problema da usabilidade. Em geral, estes documentos possuem uma série de orientações para padronizar a aparência geral dos aplicativos do ambiente gráfico e para garantir o cumprimento mínimo dos requisitos de interação (OLPC, 2007). Esta é uma maneira de manter a aparência geral do sistema e fazer os diferentes aplicativos se comportarem de forma similar. Como consequência, um usuário que já saiba usar algum aplicativo do mesmo ambiente, aprenderia a lidar com um novo aplicativo mais rapidamente pela semelhança de comportamento e aparência. Além disso, seguir os padrões estabelecidos na própria comunidade facilita o processo de documentação e de tradução, uma vez que os outros membros da comunidade já estarão familiarizados com a interface. É comum o artifício da recomendações de interface estarem permeadas por ideias norteadoras dos projetos a que servem.

2.5 Observações finais

Este capítulo apresenta os principais paradigmas de aprendizagem relacionados ao uso de tecnologias móveis na Educação. Outros trabalhos relacionados ao tema de diretrizes de projeto de aplicativos para dispositivos móveis foram apresentados de maneira geral. Há uma conexão natural entre aprendizagem vista como uma atividade de contexto e tecnologias móveis e pessoais, por isso, tem sido factível prover ferramentas capazes de apoiar a aprendizagem em qualquer momento e em qualquer lugar (SHARPLES; CORLETT;

[5] vide `http://gnome.org/`

[6] vide `http://developer.apple.com/library/mac/documentation/UserExperience/Conceptual/AppleHIGuidelines/`

WESTMANCOTT, 2002).

Alguns modelos de desenvolvimento de *software* também foram apresentados; um destes modelos em particular, a Engenharia sócio-cognitiva, tem origem e aplicação em projetos de Aprendizagem Móvel. O diferencial da Engenharia sócio-cognitiva é a abordagem de projeto centrada no usuário, embora ela seja basicamente um método iterativo e incremental, como os métodos ágeis. Esta base iterativa aproxima os modelos de desenvolvimento, mas sua aplicação não tem a mesma difusão que outros métodos iterativos. O maior problema deste método parece ser a complexidade, pois requer a participação de profissionais bastante especializados, especialmente por requerer estudos no campo de aplicação. Por outro lado, a Programação Extrema é bastante popular entre programadores e outros profissionais da área, mas carece de diretrizes específicas para a aplicação em projetos de Aprendizagem Móvel. Além disso, nenhum destes modelos de Engenharia de Software possui diretrizes específicas para sua aplicação em contextos com equipes distribuídas, sejam elas comunidades de *software* livre, sejam elas equipes de empresas trabalhando em *offshoring* ou *outsourcing*.

3 RELATO DE EXPERIÊNCIA

Este capítulo relata a experiência do autor como participante da comunidade de desenvolvedores do projeto OLPC e como um dos responsáveis técnicos na implantação da primeira fase do projeto UCA. Primeiramente, relata-se o histórico da introdução da plataforma XO no Brasil, com a participação do governo Federal e institutos de pesquisa na criação do projeto UCA. Depois, relata-se a organização do projeto OLPC como uma comunidade de *software* livre e seus principais canais de comunicação. A participação do autor no desenvolvimento do aplicativo de desenho da plataforma é descrita em seguida, bem como a interação com a comunidade de outros desenvolvedores do mundo.

Neste relato, o autor participou desta comunidade como desenvolvedor e como observador. Por isso, o método adotado para a observação do funcionamento assemelha-se ao de Gutwin, Penner e Schneider (2004), Barcellini et al. (2008), mas tem a caraterística implícita do observador fazer parte do fenômeno observado. Assim, as informações aqui contidas vêm justamente da leitura das listas de discussão da comunidade (principalmente as destinadas aos desenvolvedores do projeto *sugar-devel*[1] e *devel*[2]) ao longo do tempo de participação, dos anúncios gerais feitos pela comunidade (disponibilizados através das wikis [3]) e pelo contato direto com outros desenvolvedores. Além disso, o fato de o autor ter participado como desenvolvedor permitiu conhecer mais profundamente os mecanismos de correção de falhas e o processo de controle de qualidade.

3.1 Histórico dos projetos OLPC e UCA

Em 2005, no Fórum Econômico Mundial em Davos na Suíça, Nicholas Negroponte, que na época era professor do *Massachussets Institute of Technology* (MIT), apresentou

[1] vide `http://lists.sugarlabs.org/listinfo/sugar-devel`.
[2] vide `http://lists.laptop.org/listinfo/devel`.
[3] em `http://wiki.sugarlabs.org/` e `http://wiki.laptop.org/`.

ao mundo uma ideia que fazia pouco sentido na época: criar um *laptop* de 100 dólares. Neste momento, Negroponte e outros professores do MIT fundaram a organização OLPC (*One Laptop per Child*), uma entidade sem fins lucrativos cuja meta é distribuir *laptops* para todas as crianças do mundo (OLPC, 2007). Apesar do espanto causado à plateia na ocasião da apresentação, Negroponte veio ao Brasil no mesmo ano acompanhado de Seymour Papert (também professor do MIT) e Mary Lou Jenpsen (especialista em tecnologias LCD) para expor a ideia em detalhes ao presidente Luís Inácio Lula da Silva. A partir desta reunião, o presidente instituiu um grupo interministerial para avaliar a ideia e verificar sua factibilidade (BRASIL, 2010).

Com este grupo, institutos brasileiros especializados foram convidados a debater a viabilidade técnica e pedagógica da implantação de computadores portáteis para todos os alunos das escolas públicas brasileiras. As avaliações davam conta de que seria necessário ter um planejamento pedagógico fortemente aliado ao planejamento tecnológico (CORRÊA et al., 2006). Três instituições foram chamadas para dar seus pareceres sobre a proposta:

LSI EPUSP: Laboratório de Sistemas Integráveis da Escola Politécnica da Universidade de São Paulo;

CenPRA: Centro de Pesquisa Renato Archer, ligado ao Ministério de Ciência e Tecnologia (MCT);

CERTI: Fundação Centros de Referência em Tecnologias Inovadoras, ligada à Universidade Federal de Santa Catarina.

A organização OLPC parece ter sido a primeira iniciativa de larga escala a viabilizar a produção de *laptops* de baixo custo e baixo consumo de energia, dando origem ao que aos *netbooks*. Aliada à esta inovação produtiva, a organização também estava permeada por influências ideológicas, provenientes das comunidades de *software* livre, e pedagógicas, fundamentadas na teoria Construcionista de Papert (1999). Pela estimativa inicial, Negroponte esperava distribuir mais de 100 milhões de *laptops* pelo mundo, mas havia chegado até cerca de 1,5 milhão até meados de 2010: apenas dois países decidiram comprar o suficiente para promover a proporção de 1:1 entre máquinas e crianças (WARSCHAUER; AMES, 2010).

Scacchi (2007) descreve o *software* livre como um movimento social, num sentido de atividade coletiva e organizada permeada por questões ideológicas. Observando o impacto

causado pela iniciativa OLPC, nota-se a pertinência desta descrição, pois o surgimento da organização tem muito a ver com as convicções típicas do *software* livre no tocante à liberdade e transparência, aliadas à crença de Papert de que um computador individual é essencial para a construção do conhecimento na sociedade atual (PAPERT, 1999; WARSCHAUER; AMES, 2010). Assim, o objetivo de distribuir *laptops* para crianças de todo o mundo é a concretização da visão de que um computador conectado é uma poderosa ferramenta para a aprendizagem (OLPC, 2007).

Este conjunto de inovações técnico-pedagógicas representou, então, um desafio para o grupo responsável pela avaliação proposta ao governo brasileiro, pois não havia nenhuma iniciativa similar no mundo. É importante ressaltar que o projeto M-Learning é quase contemporâneo aos fatos descritos e algumas de suas avaliações já estivam disponíveis para a comunidade científica (BULL et al., 2005; SYVÄNEN et al., 2005; NAISMITH et al., 2004), além de haver um vasto conhecimento sobre aplicação de computadores na Educação e para inclusão digital (CORRÊA et al., 2006). Tal como narrado em outras experiências (SHARPLES, 2000), avaliar o uso de tecnologias inovadoras ou utilizadas num contexto inovador (como *netbooks*, no caso do projeto OLPC) com tamanha escala visando a saturação era desafiador também por conta da realidade brasileira, onde a disponibilização de computadores conectados para professores e alunos configura uma oportunidade para inclusão social e digital (CORRÊA et al., 2006).

A conclusão dos estudos pelas instituições de pesquisa culminou com a criação do projeto UCA (Um Computador por Aluno), que "tem como objetivo ser um projeto Educacional utilizando tecnologia, inclusão digital e adensamento da cadeia produtiva comercial no Brasil" (BRASIL, 2010). Com início oficial em 2007, 5 escolas foram selecionadas como parte dos experimentos pedagógicos do projeto, nas cidades de São Paulo (SP), Porto Alegre (RS), Palmas (TO), Piraí (RJ) e Brasília (DF). Os experimentos tinham o objetivo de gerar modelos para administração das escolas, para infraestrutura necessária e para uso pedagógico das máquinas, visando a expansão do projeto.

Nesta primeira fase, foram também avaliados outros modelos de *laptop* de baixo custo disponíveis detectados durante os estudos. Além do *laptop* XO da OLPC, foram avaliados os modelos Classmate, fabricado pela Intel, e Mobilis, produzido pela empresa Encore. As escolas de Porto Alegre e de São Paulo foram equipadas com XO; as escolas de Piraí e Palmas receberam Classmates; e a escola de Brasília recebeu Mobilis. O número de máquinas variou de escola para escola.

A segunda fase do projeto consiste na expansão dos experimentos para aproximadamente 300 escolas de todo o país, aplicando o conhecimento adquirido na primeira fase do projeto. Para este fim, o governo federal fez uma chamada pública para compra de 150.000 *laptops* educacionais (termo usado na própria chamada). O pregão nº 107/2008 foi vencido pelo consórcio CCE/DIGIBRAS/METASYS depois de uma série de contestações judiciais. Esta fase inclui a distribuição dos *laptops* para alunos e professores das escolas contempladas (que são tanto municipais quanto estaduais), infraestrutura para acesso à Internet e capacitação de gestores e professores no uso das tecnologias (BRASIL, 2010). Outra característica desta fase é a criação dos municípios UCA Total, nos quais todas as escolas são atendidas pelo projeto. Estes municípios permitem entender melhor as condições a que estarão submetidos os estudantes e professores quando o projeto for expandido para todas as escolas do país.

O projeto OLPC e o projeto UCA estão, portanto, muito próximos um do outro em **objetivos** a cumprir e **instrumentos** para realizá-los. O governo brasileiro resolveu trilhar um caminho mais voltado a gerar concorrência entre diferentes fornecedores de hardware, enquanto a OLPC criou suas próprias soluções para *software* e *hardware* capazes de cumprir os requisitos pedagógicos. Outros governos do mundo, entre eles Uruguai, Peru e Uganda, também têm projetos em parceria com a OLPC para implantar distribuição de *laptops* de baixo custo para seus estudantes. Warschauer e Ames (2010) consideram a decisão de desenvolver a solução própria de *software* e de *hardware* um dos fatores negativos da iniciativa da organização não-governamental. Os autores também consideram de extrema importância a adaptação à realidade local no tocante à resolução de problemas e à estratégia pedagógica adotada para implantar o uso das máquinas. Neste sentido, o modelo de formação do UCA parece mais adequado, pois provê formação distribuída nas escolas participantes (BRASIL, 2010), mas resta saber como isso continua no longo prazo.

3.2 Organização da comunidade OLPC

O projeto OLPC é, na verdade, composto por três projetos interdependentes: de *software* livre, de *hardware* livre e de Educação. No início, o projeto de *hardware* teve de ser desenvolvido do zero, tendo como principal requisito o baixo consumo de energia. Este ponto foi trabalhado de forma tanto fazer a bateria durar mais quanto maximizar sua vida útil. O projeto educacional está fortemente baseado na teoria Construcionista proposta por Papert (1999), que usa criatividade e a exploração como porta de entrada

para construção do conhecimento (RESNICK, 2002). Há uma equipe da organização centrada em capacitar os professores das escolas que recebem os *laptops*, bem como recursos de educação à distância para dar suporte à esta capacitação.

O projeto de software está organizado em torno do ambiente Sugar, especialmente concebido para a plataforma *laptop* XO. Além deste ambiente, merece atenção especial da comunidade de *software* organizada no projeto o desenvolvimento voltado para a implementação da rede *Mesh* (baseada no esboço IEEE 802.11s) (CARRANO et al., 2007). As redes em malha receberam este destaque justamente por terem sido usadas em projetos de inclusão digital em vários países pelo mundo, evitando o custo do cabeamento (ABELÉM et al., 2007).

Dentro do projeto de desenvolvimento do ambiente Sugar, existem diversos aplicativos de usuário em implementação; no contexto do ambiente, estes aplicativos são chamados de atividades. Para distribuição das chamadas atividades, foi adotada um estratégia difundida entre projetos de *software* livre: a construção de um canal (portal *web*) para *download*. Portais semelhantes já tinham sido feito anteriormente para distribuição de *softwares* complementares para o navegador Firefox, por exemplo. Esta mesma tática é comum na distribuição e compra de aplicativos para celulares, onde o canal inclui a função de cobrar do usuário o valor de cada aplicativo.

Os tópicos a seguir descrevem detalhes relacionados ao funcionamento do projeto de *software*, o ambiente Sugar, detalhando a dinâmica de trabalho da equipe virtual e das atividades de *design* da comunidade.

3.2.1 Equipe

As equipes na comunidade OLPC/Sugar Labs estão organizadas em torno de projetos de aplicativos, ou seja, cada *software* tem um equipe distinta com um repositório de código dedicado ao seu aplicativo. Há ainda equipes dedicadas a porções de *software* específicas, do como o núcleo Linux ou *drivers* para rede *Mesh* e as bibliotecas para construção de *software* colaborativo. Nas equipes de desenvolvimento das chamadas atividades, sempre há um desenvolvedor-líder. Este líder é responsável por distribuir as tarefas designadas no calendário de desenvolvimento (*roadmap*), pois todos os componentes do ambiente Sugar têm de obedecer um calendário unificado. O líder também deve distribuir, internamente à equipe, as responsabilidades de correção de defeitos; no procedimento padrão de cadas-

tro de problemas e defeitos (*bugs*, no jargão da comunidade), as tarefas sempre ficam atribuídas ao líder da equipe.

A comunidade OLPC/Sugar Labs tem um ciclo (ou iteração) para lançamento de versões com fases bem distintas: **desenvolvimento**, **congelamento de interface**, **congelamento de código** e **distribuição**. Na fase de **desenvolvimento**, o planejamento consiste em escolher os recursos a serem implementados e os problemas a corrigir nesta iteração; cada uma destas tarefas é marcada com um item (*ticket* no jargão da comunidade) no sistema de acompanhamento (*issue/bug tracking*). Desta forma, o "fechamento" do item no sistema de acompanhamento significa a conclusão da implementação de alguma correção ou funcionalidade. Para manter o acompanhamento dos itens, o responsável pelo gerenciamento do calendário de lançamento pode simplesmente acompanhar o progresso dos itens marcados como resolvidos.

A fase de **congelamento de interface** tem relação direta com o time de tradução: para haver tempo hábil para os tradutores completarem sua tarefa, a estratégia é de estabelecer uma data a partir da qual não se modifica mais a interface. A ferramenta de tradução, gettext [4], captura os textos marcados para a tradução e substitui estas mensagem conforme o idioma instalado no sistema. A tarefa da equipe de tradução é, justamente, produzir as mensagens adequadas em cada idioma, garantindo uma interface consistente.

O **congelamento de código** dá uma referência para a equipe de qualidade ao reportar erros e defeitos nos *softwares* da plataforma. A equipe de qualidade usa as versões "congeladas" dos *softwares* para fazer os testes. Os problemas encontrados são relatados no sistema de acompanhamento e têm tratamento semelhante aos itens da fase de desenvolvimento. Neste ponto, as equipes procuram apenas modificar o código referente aos problemas relatados nos testes feitos pela equipe de qualidade.

A correção dos problemas na fase de congelamento de código permite o posterior empacotamento para **distribuição**. Na distribuição, as equipes de empacotamento do sistema operacional ficam responsáveis por juntar as peças de *software* e compor uma imagem coerente a partir das versões disponíveis. As imagens resultantes desta fase são lançadas para o público e, consequentemente, usadas nos experimentos e escolas vinculados ao projeto OLPC.

A comunidade do Sugar Labs organiza os voluntários em alguns papéis: educador,

[4] veja mais detalhes em `http://www.gnu.org/software/gettext/`

escritor, porta-voz e tradutor, nas responsabilidades não-técnicas, e desenvolvedor e *designer*, com atribuições de caráter técnico. Os papéis e suas tarefas são detalhados a seguir.

Educador: a principal tarefa é fazer parte da equipe de educadores do Sugar Labs, de forma a orientar os princípios pedagógicos na comunidade e também acompanhar o uso dos *laptops* nas escolas. Considera-se importante a participação dos educadores nos Sugar labs locais.

Escritor: responsáveis por cuidar dos principais aspectos de documentação da comunidade, como manuais, guias, tutoriais e compêndios de perguntas e respostas.

Porta-voz: estes assumem o papel de divulgar o projeto através de contato pessoal. Neste sentido, organizam eventos, participam de palestras e lançam *press releases*.

Tradutor: atuam na tradução do ambiente Sugar e seus aplicativos, além de manuais e demais documentos *online*.

Desenvolvedor: além de implementar o ambiente Sugar e aplicativos, os desenvolvedores também testam e empacotam os componentes do ambiente, além de cuidar da infraestrutura usada por todos relacionados ao projeto.

Designer: responsáveis pelo projeto gráfico dos componentes do projeto, como ícones e imagens dos aplicativos, das páginas *web* e outros materiais de circulação geral. Além disso, os esboços (*mockups*) para indicar a aparência geral dos aplicativos do ambiente parte dos *designers*.

Os papéis dentro da comunidade têm um caráter hierárquico, e os membros podem mudar de papel em função de sua participação e interesse. No geral, os membros da comunidade que cuidam dos componentes mais críticos (como as bibliotecas de colaboração e o núcleo Linux) estão formalmente vinculados à fundação OLPC ou à Sugar Labs; estas pessoas também controlam o calendário de desenvolvimento do projeto. Além disso, é comum os membros estarem envolvidos em mais de uma equipe ao mesmo ou terem mais de uma atribuição dentro da comunidade, tal qual descreve-se em Scacchi (2007), Gutwin, Penner e Schneider (2004), Barcellini et al. (2008).

O modelo de equipe virtual é cabível para a comunidade organizada em torno do projeto OLPC/Sugar Labs, mesmo que a proposta das equipes virtuais tenha sido formulada

em um contexto mais comercial. Por isso, é possível descrever a atuação desta comunidade nos termos destacados por Casey (2010): coordenação, visibilidade, comunicação e cooperação. Sendo um projeto de *software* livre, apresenta características muito similares na atuação dos membros em comparação ao que se conhece em *offshoring* e *outsourcing*. A motivação dos desenvolvedores gira muito em torno da ideologia do projeto e do desafio técnico, estando aí a base para a **cooperação** dos membros.

A *coordenação* do projeto fica por conta da fundação; no caso do projeto de hardware, a OLPC é responsável por administrar o calendário de lançamento de versões, distribuir as tarefas para os membros ativos da comunidade e pela garantia do processo de controle de qualidade. No caso do projeto de *software* (sistema operacional e aplicativos), a coordenação é feita pela fundação Sugar Labs. Inicialmente, todo o ambiente criado era compatível somente com a plataforma XO, mas por iniciativa da própria comunidade, o sistema operacional foi compatibilizado com plataformas genéricas, tal qual outras distribuições GNU/Linux disponíveis.

Os papéis e responsabilidades de cada desenvolvedor são conhecidos publicamente, uma vez que os sistemas eletrônicos de atribuição de tarefas podem ser acessados pela Internet sem restrições. Desta maneira, a *visibilidade* destacada por Casey (2010) é tratada através de sistemas eletrônicos abertos que permitem saber o estado de uma tarefa em execução e também conhecer o calendário dos lançamentos para os próximos períodos.

A *comunicação* dos membros da comunidade é feita principalmente através de listas de discussão abertas, como descrito por Gutwin, Penner e Schneider (2004), Scacchi (2007), Barcellini et al. (2008). As informações consolidadas, bem como artigos em destaque (informações sobre como contribuir com o projeto, qual as novidades presentes na última versão) ficam disponíveis nas wikis relacionadas.

Em 2008, a OLPC foi dividida por desentendimentos internos; a organização OLPC passou a tratar exclusivamente do desenvolvimento do *hardware* e da fabricação/distribuição das máquinas e uma nova fundação, batizada de Sugar Labs, passou a tratar do sistema operacional como um todo e do projeto pedagógico. A partir de então, os repositórios de código e as ferramentas de gerenciamento de bugs e de calendário passaram para a fundação Sugar Labs, bem como as responsabilidades de capacitação de pessoal. Ficando estabelecida essa divisão, a nova fundação traçou um planejamento que inclui criar Sugar Labs locais, exatamente nos lugares onde houver presença dos laptop XO; de forma que os objetivos destes estabelecimentos locais incluem (SUGAR LABS,):

- ajudar no suporte técnico-pedagógico;
- criar novos *softwares* e práticas pedagógicas;
- prover tradução para *software*, conteúdos e documentação;
- prover serviços de integração e personalização.

Cada desenvolvedor tem um conjunto de ferramentas a sua disposição para o desenvolvimento de aplicativos e para comunicação. Existe uma série de orientações sobre como se tornar um membro ativo da comunidade [5], variando desde de funções que exigem pouco conhecimento técnico como na equipe de testes, ou para simplesmente fornecer ideias de aplicativos e melhorias. Entre as funções desempenhadas pelos desenvolvedores pela comunidade estão as funções de:

programação: estes desenvolvedores devem cuidar tanto do ambiente sugar quanto dos aplicativos em si;

empacotamento: ficam responsáveis por compilar versões do ambiente sugar e dos aplicativos para distribuição a equipe de testes e para outras distribuições GNU/Linux;

infraestrutura: esta equipe cuida dos servidores de código e comunicação usados por toda a comunidade.

teste: responsáveis por testar todos os componentes do sistema, principalmente aqueles mais interativos. Esta equipe é, por vezes, chamada de equipe de qualidade (ou *quality assurance team*, em inglês).

Os programadores têm de usar algumas ferramentas bastante particulares do projeto. Para o controle de versão, adota-se a ferramenta git [6], que costuma ser adotada para gerenciamento de projetos de grande porte, como o próprio núcleo Linux. Além disso, os programadores frequentemente necessitam de ferramentas para emular o ambiente Sugar em seus espaços de trabalho. Para este fim, usam-se máquinas virtuais, como a dos produtos VirtualBox e VMWare, ou as ferramentas disponíveis no *jhbuild*, um artifício usado em alguns projetos de *software* livre para criar uma nova instância do servidor gráfico como o X11.

[5] para maiores detalhes, vide `http://wiki.sugarlabs.org/go/Sugar_Labs/Getting_Involved`
[6] `http://git-scm.com/`

3.2.2 Projeto gráfico

Para unificar o projeto gráfico do ambiente Sugar, foi adotado o esquema de orientações para interface (*Human Interface Guidelines*, em inglês), conforme já abordado na seção 2.4.3.2. Além de servir como orientação para construção de interfaces coerentes, o documento também detalha os recursos específicos do ambiente Sugar para os desenvolvedores. Este detalhamento é feito justamente por conta da interface gráfica e as bibliotecas por terem sido criadas do zero ou reescritas, de forma que os desenvolvedores, mesmo familiarizados com projetos de *software* livre, não dominam as chamadas de sistemas e API (*Application Programming Interface*). O documento referido apresenta a metáfora da proximidade, criada para fazer a experiência de usuário se aproximar das premissas Construcionistas no contexto de aprendizagem (OLPC, 2007), servindo de fonte para pessoas interessadas em se aprofundar nos objetivos do projeto.

As recomendações de interface do ambiente Sugar estabelecem como requisitos uma série de características a serem cumpridas em seus aplicativos. Neste contexto, os requisitos são chamados de princípios de *design*, e partem do pressuposto do público-alvo ser jovem (entre 5 e 12 anos), sem ou (com pouca) experiência prévia no uso de computadores e pode ser de qualquer parte do mundo (OLPC, 2007). Os princípios são listados a seguir:

- desempenho;
- usabilidade;
- simplicidade;
- confiabilidade;
- segurança (de dados);
- adaptabilidade;
- (capacidade de) recuperação;
- mobilidade;
- transparência;
- acessibilidade.

Os princípios de *design* buscam cumprir os requisitos também através da padronização dos elementos da interface gráfica do ambiente. No caso do Sugar, o estabelecimento da metáfora da proximidade é o ponto mais forte do *design* da interface com a teoria pedagógica que dá a suporte ao projeto como um todo.

O projeto da interface, entretanto, também foi alvo de discussão na comunidade. O documento das orientações de interface foi lançado como um documento de referência *antes* da implementação de todas as ideias existentes para a interface. O documento cumpre, portanto, um papel de uma **especificação de sistema**, mas com o caráter informal característico das comunidades de *software* livre descrito por Scacchi (2007); e o mesmo tipo de recurso informal pode ser observado na documentação da API das bibliotecas do ambiente Sugar.

O processo de evolução da interface se deu somente depois de boa parte das especificações estarem implementadas e de haver experimentos com laptops XO acontecendo em escolas pelo mundo. Pessoas da chamada equipe de qualidade (*quality assurance team*, ou *QA team* na comunidade) receberam laptops e eventualmente reportavam problemas de comportamento na interface em testes com usuários. O autor contribuiu para o processo de evolução fazendo um teste de usabilidade (relatado em Martinazzo et al. (2008)), enviando sugestões para aprimorar a interface em função dos problemas de usabilidade observados.

3.3 Desenvolvimento da plataforma de desenho

O autor atua como pesquisador no Laboratório de Sistemas Integráveis (LSI) da Escola Politécnica da Universidade de São Paulo. No início do projeto UCA, o LSI decidiu implementar alguns aplicativos para a plataforma XO. Em experiências anteriores, foram implementados e testados alguns softwares voltados para autoria e educação musical, além de alguns jogos. A estratégia foi de explorar os recursos da plataforma XO portando alguns destes aplicativos.

Na etapa de levantamento de requisitos, a plataforma física ainda não estava pronta, de forma que os requisitos ainda estavam baseados nas especificações disponíveis. Pelas caraterísticas do *hardware*, é necessário evitar a criação frequente de arquivos temporários (pois a armazenagem é feita em memória tipo Flash, com ciclos de escrita mais limitados que tecnologias magnéticas). O tamanho da tela é reduzido (7,5 polegadas), mas tem

uma resolução bastante grande, de 1200 x 900 pixels, maior que a disponível para os *laptops* da época.

Havia também uma diretriz importante para os desenvolvedores no tocante à duração da bateria. Entre os objetivos do projeto OLPC, está a distribuição dos *laptops* XO para crianças de países pobres, nos quais o acesso irrestrito à energia elétrica pode ser um problema bastante grave. Existe também a possibilidade de muitos dos aplicativos desenvolvidos na comunidade internacional serem adotados por todos os países interessados na compra dos *laptops* XO (estes aplicativos são chamados de *core activities*). Por isso, ao desenvolver aplicativos que fazem uso da rede, é importante maximizar a vida útil da bateria, minimizando o tráfego na rede. Além disso, ao projetar aplicativos colaborativos, é importante observar como é a intensidade da troca de pacotes; o dispositivo foi projetado para ser compatível com o padrão IEEE 802.11s, permitindo que os *laptops* configurem uma rede em malha.

O ambiente Sugar foi desenvolvido ao mesmo tempo em que os aplicativos criados no LSI eram portados para a plataforma. Muitas das API e bibliotecas disponíveis mudavam durante a implementação dos aplicativos, por isso, houve diversos ajustes durante o desenvolvimento. Por uma decisão de *design* da equipe de desenvolvedores da OLPC, o ambiente Sugar foi escrito na linguagem Python, usando a biblioteca gráfica GTK para implementação da interface. A justificativa para a escolha deste conjunto reside na agilidade de desenvolvimento dado pela linguagem Python (já que o ambiente seria implementado quase do zero) e pelo grande suporte à tradução oferecido pela biblioteca gráfica GTK.

Dadas as restrições para o desenvolvimento de *software*, o aplicativo de desenho (até então chamado de Oficina de Desenho) foi transportado para a linguagem Python e para a biblioteca gráfica GTK, a fim de facilitar a integração com o restante do ambiente gráfico. Durante o Fórum Internacional de Software Livre (FISL) de 2007, o autor foi representar o LSI e apresentou os aplicativos desenvolvidos no laboratório para os representantes da OLPC no evento. Os representantes ficaram interessados em incorporar o aplicativo de desenho como parte da distribuição padrão do ambiente Sugar. O autor passou então a liderar a equipe de desenvolvedores na adaptação da interface da Oficina de Desenho. Neste momento, haviam orientações para a integração da interface e até mesmo esboços de como ela deveria ser.

A equipe do LSI responsável por implementar a Oficina de Desenho utilizou a metologia conhecida como Programação Extrema para o desenvolvimento, e o autor assumiu o papel

de *coach* da equipe. Assim, a equipe trabalhou segundo as práticas primárias de XP (vide seção 2.4.2.1 para maiores detalhes nestas práticas), com exceção àquelas relacionadas a testes automatizados, e passou a obedecer o calendário de versões estabelecido para o ambiente Sugar (detalhado no tópico 3.2). A equipe passou a registrar as tarefas e funcionalidades (ou seja, as *histórias* no método XP) da implementação no sistema *web* de acompanhamento do projeto. Outros membros, do controle de qualidade, faziam testes dos aplicativos e reportavam defeitos das versões, conforme detalhado na seção 3.2.

A estratégia de comunicação da equipe com a comunidade baseou-se em obter *feedback* o rápido possível e com a maior frequência possível. Por isso, os membros da equipe participavam diariamente dos canais de *chat* da comunidade OLPC, a fim de tirar dúvidas e debater decisões de arquitetura. Quando o debate ganhava complexidade, a equipe de desenvolvimento usava a lista de discussão para manter o histórico e tornar as decisões mais transparentes para os outros membros da comunidade, como é praxe no desenvolvimento colaborativo (BARCELLINI et al., 2008). Além disso, uma vez ao ano, no Fórum Internacional de Software Livre, pelo menos o autor (com mais um representante, eventualmente) encontrava-se com representantes da fundação OLPC. A participação nestes eventos permitia esclarecer pontos em aberto e questões sobre os rumos no médio prazo para o desenvolvimento da plataforma XO e do ambiente Sugar.

Em um dado momento, porém, um teste de usabilidade revelou diversos problemas na interface e provocou uma reflexão sobre a interação com o ambiente Sugar. Assim, o autor propôs modificações na interface (a interface original é exposta na figura 3.1), baseando-se no resultado dos testes. Depois de algum processo de discussão do projeto gráfico, chegou-se ao esboço mostrado na figura 3.2.

A proposta original usava várias abas para cada um dos tipos de ferramenta disponível para o usuário. Na proposta feita depois dos testes, as ferramentas ficariam concentradas em uma única barra, para evitar a necessidade de se mudar a aba. Os testes com usuários evidenciaram que esta poderia ser um conceito confuso para alguns usuários (MARTINAZZO et al., 2008).

Além dos testes funcionais feitos a cada ciclo semanal completado, a atividade foi testada *com usuários* em vários momentos pela equipe de desenvolvimento. No entendimento do autor, que liderou também os testes com usuários, é importante nos testes com usuários abranger não somente as *crianças*, mas também os *professores*. Dentro do contexto do projeto UCA, diferentemente do projeto OLPC, a entrega e o uso dos *laptops*

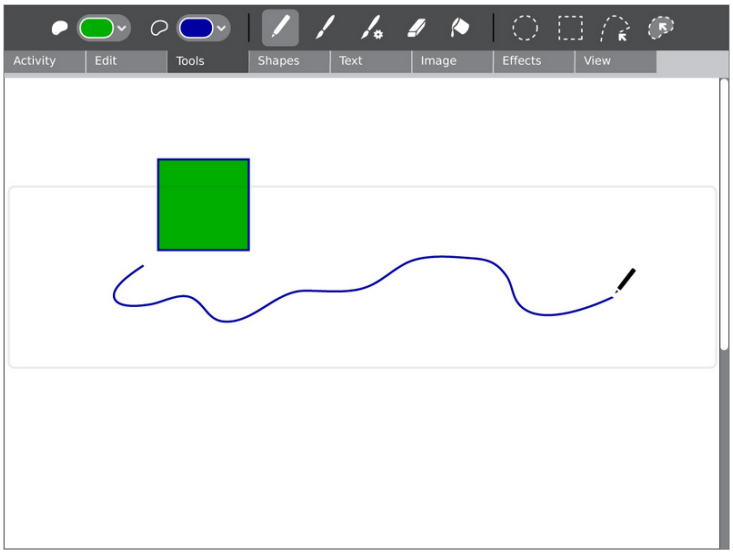

Figura 3.1: *Mockup* do primeiro *design* para o aplicativo de desenho (retirado de `http://wiki.laptop.org/go/Paint`).

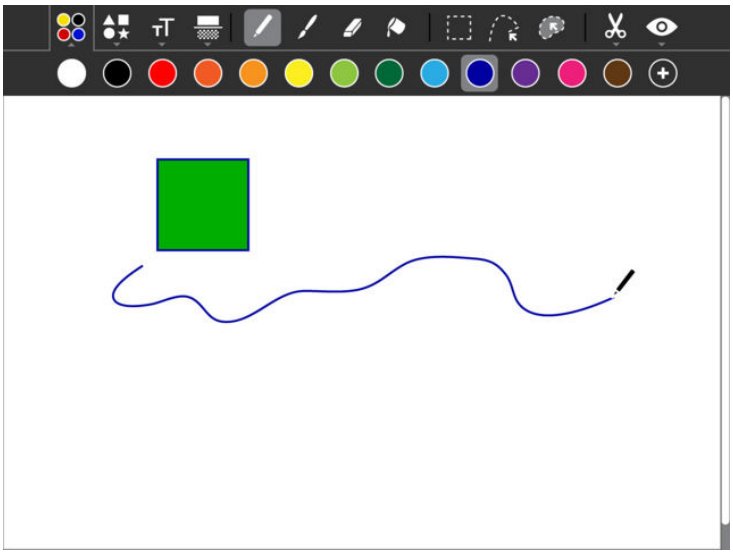

Figura 3.2: *Mockup* da evolução do *design* para o aplicativo de desenho (retirado de `http://wiki.laptop.org/go/Design/Toolbars`).

está necessariamente vinculada à uma escola, ou seja, o uso dos computadores portáteis mais frequentemente dar-se-á em sala de aula. O uso do aplicativo tem, portanto, um componente relacionado ao estudante e outro relacionado ao professor, que pode propor atividades em sala de aula (conforme abordou-se na seção 2.1, no estudo dos paradigmas de aprendizagem). As publicações resultantes deste, e de outros estudos, encontram-se no apêndice A.

Embora o autor tenha atuado na liderança da equipe de programadores do aplicativo de desenho, depois de cerca de um ano e meio, a participação foi perdendo intensidade até deixar de haver contribuição em linhas de código. Com o passar do tempo, entretanto, outros membros da comunidade assumiram a liderança da implementação, ficando com as atribuições do cargo. Assim, a participação da comunidade *cresceu* ao longo do tempo, enquanto a participação da equipe da universidade *diminuiu*.

Mesmo assim, pessoas ligadas ao Sugar Lab da Argentina [7] dão continuidade ao desenvolvimento desde 2009. Mesmo estando em outro país, os integrantes deste Sugar Lab participam ativamente da implantação do projeto OLPC no Uruguai [8], principalmente ajudando no desenvolvimento e na tradução de atividades. O rumo do desenvolvimento da atividade Pintar está, agora, ligado à melhoria de desempenho e à correção de defeitos. O fato de os desenvolvedores terem representantes próximos ao usuários finais permite acesso a uma grande quantidade de dados no tocante à gargalos de desempenho e a detecção de *bugs*. Professores e alunos das escolas uruguaias podem fornecer informações aos desenvolvedores sobre quais são as prioridades no desenvolvimento deste e de outros aplicativos, sempre do ponto de vista do uso real da ferramenta. Além destas contribuições das escolas, eventuais descrições provenientes de projetos pilotos em outras partes do mundo também podem ser usadas para guiar o desenvolvimento.

Este cenário é similar àquele em que o autor esteve envolvido: contribuições de várias partes do mundo através do sistema de acompanhamento *online* (*issue/bug tracker*) e crianças numa escola local usando os aplicativos desenvolvidos, permitindo a coleta de informações capazes de permitir um *design* centrado no usuário final. Por isso, mesmo com a mudança da responsabilidade do desenvolvimento, as informações coletas a partir do usuário final permitem uma abordagem centrada nas necessidades dos usuários por parte da comunidade Sugar.

[7] mais informações em `http://ar.sugarlabs.org/`.

[8] o projeto é chamado de Plan Ceibal no país; vide `http://www.ceibal.edu.uy/`.

3.4 Observações finais

Neste capítulo relatou-se a experiência do autor no desenvolvimento de *software* voltado para Educação, em particular para os projetos UCA e OLPC. O autor ainda atuou na implantação do projeto UCA em uma escola participante dos primeiros experimentos no Brasil.

Este capítulo também detalhou o funcionamento da comunidade organizada em torno do projeto OLPC. Embora o relato seja majoritariamente sobre como esta comunidade desenvolve *software*, foram tratados alguns pontos de seu funcionamento no tocante ao desenvolvimento de *hardware* e sua abordagem pedagógica. A aliança entre aspectos tecnológicos e pedagógicos é um dos maiores diferenciais da comunidade organizada em torno deste projeto específico.

O projeto UCA, por outro lado, tem um caráter institucional mais forte, pois conta com investimento contínuo do governo federal brasileiro. Outro ponto de destaque do programa brasileiro é a proposta de formação nas escolas envolvendo especialistas em informática na Educação de todo o Brasil. Ainda assim, a pluralidade das necessidades do projeto UCA requer o exercício de diversos papéis para o funcionamento do projeto como um todo. Logo, os papéis técnicos e pedagógicos propostos pela comunidade do projeto OLPC para a criação de Sugar Labs locais fazem sentido para o contexto nacional. O diferencial do projeto UCA, entretanto, é ter o suporte do governo brasileiro, trazendo algumas condições novas. Portanto, os papéis para a realidade brasileira, ao considerar o desenvolvimento colaborativo de *software* são:

- educador: atua na formação de professores de escolas participantes e no suporte pedagógico;
- desenvolvedor: projeta e implementa os aplicativos;
- *designer*: responsável projeto gráfico dos aplicativos;
- suporte de infraestrutura: pessoas responsáveis por prover e manter a infraestrutura elétrica e de rede das escolas em funcionamento. Existem esferas de governo que têm empresas públicas somente para este fim, como a Prefeitura de São Paulo através da PRODAM.

Não existe uma comunidade de *software* livre organizada para atuação específica no

projeto UCA. Entretanto, há diversos brasileiros participando da comunidade do projeto OLPC, além de iniciativas da *Associação de Software Livre* e do governo em diferentes esferas tentando fomentar o uso de *software* livre na Educação. Este trabalho tem, portanto, a intenção de prover diretrizes para ajudar na interação entre especialistas em informática na Educação, amplamente disponíveis e ativos nas universidades brasileiras, com comunidades de *software* livre.

4 MODELAGEM DO PROCESSO

Este capítulo apresenta uma modelagem do processo de interação entre especialistas em Aprendizagem Móvel e desenvolvedores, analisando o contexto de desenvolvimento colaborativo das comunidades de *software* livre.

4.1 Considerações iniciais

Uma abordagem centrada no aprendiz, conforme levantado na bibliografia (vide capítulo 2), sugere uma abordagem mais **pessoal** no projeto das tecnologias. Além disso, para dar apoio à interação entre aprendizes é necessário que estas tecnologias tenham capacidade de se *comunicar em rede*. Como apontado por Syvänen et al. (2005), os locais nos quais se pode aprender não são controlados e conhecidos, por isso pode afetar a disponibilidade e a qualidade de ferramentas e serviços online. Nem todos os locais de aprendizagem, portanto, têm conexão à Internet disponível; e esta pode ser uma restrição para o compartilhamento de informações e colaboração mediada por computador, embora a restrição em si possa encorajar os aprendizes a criar seus próprios locais de aprendizagem.

Tecnologias pessoais com capacidades de rede são capazes de suportar atividades dos paradigmas de aprendizagem Construtivista, Situado, Colaborativo e Informal, conforme discutido na seção 2.1. Tecnologias móveis também são muito bem adaptadas para aplicações sensíveis ao contexto (SHARPLES; CORLETT; WESTMANCOTT, 2002; SHARPLES, 2000).

Comparativamente, os computadores desktop têm mais poder computacional e gráficos mais poderosos, em geral. Estes recursos permitem aplicações mais complexas e uso mais intensivo da CPU nas plataformas desktop. Esse tipo de computador também é suscetível de ser utilizado como um recurso compartilhado e não oferece mobilidade física em si. Desta forma, ao projetar uma ferramenta de autoria em um ambiente com computadores desktop, é importante analisar se haverá pessoas em movimento ou se o arranjo será mais

estático, conforme os quadrantes expostos na figura 2.1. O segundo caso permite explorar mobilidade ao longo do tempo (considerando-se uma tecnologia estática com as pessoas em movimento). Estas considerações podem ajudar os projetistas a desenvolver modelos de tarefas, como requer, por exemplo, a Engenharia sócio-cognitiva (vide seção 2.4.1).

O sistema de entrada é uma diferença muito importante entre ferramentas de autoria projetadas para *desktops* e computadores móveis. Embora *desktops* forneçam entradas mais complexas (como textos longos usando teclados e periféricos, como *joystick*, *mouse* e câmeras ou combinações destas possibilidades) e, consequentemente, permitir as produções mais ricas, os dispositivos móveis têm a vantagem da portabilidade: os aprendizes podem carregá-los e usar em situações cotidianas. Mesmo as produções feitas com dispositivos móveis sejam mais simples, a característica forte destes dispositivos é ser sensível ao contexto, e isto é importante para a ligação entre situações formais e informais de aprendizagem (KUKULSKA-HULME et al., 2009).

O desenvolvimento de *software* livre tem sido apontado como uma abordagem para se aprender a lidar com desenvolvimento distribuído (HAWTHORNE; PERRY, 2005). Esta é uma modalidade de Engenharia de Software cada vez mais importante para os profissionais da área (PRIKLADNICKI; AUDY, 2010; LIU, 2005). Usar modelos iterativos para a gestão de projetos de *software* é particularmente recomendado no caso de os requisitos serem pouco conhecidos ou mudarem ao longo da execução do projeto (LARMAN, 2003). Paralelamente, o projeto de tecnologias pessoais, e, particularmente as aplicadas no contexto da aprendizagem, tem maior respaldo nas abordagens centradas no usuário (SHARPLES et al., 2002). A Engenharia sócio-cognitiva tem coerência com a teoria da Aprendizagem Móvel, mas seu modelo de desenvolvimento é demasiado exigente com relação ao perfil profissional. Suas etapas podem exigir testes com usuários e análises teóricas a respeito destes resultados ou até mesmo levantamentos nos ambientes de uso das tecnologias, sendo necessário um nível de especialização em informática na Educação.

Há, de um lado, a força do desenvolvimento das comunidades de *software* livre e, do outro, o alto nível de especialização na comunidade científica no tocante a aplicação de tecnologias na Educação. Um desenvolvimento feito segundo a Engenharia sócio-cognitiva poderia ser impulsionado pela força técnica das comunidades de *software* livre. Sabe-se também que as comunidades carecem de abordagens favorecendo o usuário final, principalmente quando este tem pouco conhecimento técnico (YEATS, 2006). Dada a especialização disponível na comunidade científica, é possível aliar estas características a

fim de produzir aplicativos mais adequados para aprendizagem. Para esta nova dinâmica, entretanto, é necessário prover formas de interação entre os modelos baseados em ciclos de desenvolvimento (iterativos) adotados por especialistas e o modelo distribuído da comunidade.

O caráter voluntário do desenvolvimento colaborativo de *software* é provavelmente sua marca mais profunda (SCACCHI, 2007). Desta forma, é comum os projetos de *software* livre serem iniciados por um empresa ou universidade e depois receberem ajuda de outras pessoas ou de outros grupos. No caso da informática na Educação, as propostas de desenvolvimento de aplicativos também costumam partir de instituições especializadas, seguindo, de certa forma, o fluxo de atividades do desenvolvimento colaborativo.

Na comunidade organizada em todo do projeto Sugar, existem alguns casos que demonstram este fluxo. A empresa Red Hat, dos Estados Unidos, assumiu a implementação do ambiente e do núcleo customizado desde o começo. Já a empresa Collabora, da Inglaterra, desenvolve algumas das bibliotecas usadas para criação de aplicativos colaborativos do ambiente Sugar; ambas recebem contribuições em código de programadores voluntários. As chamadas atividades do Sugar seguem exemplo semelhante, em que universidades contribuíram ou implementaram aplicativos (como o jogo da memória, programas para composição musical e o aplicativo de desenho, narrado na seção 3.3) e recebem contribuição de programadores voluntários.

A seguir, são feitas algumas considerações a respeito do **projeto** de aplicativos para Educação, além de se **modelar** o desenvolvimento colaborativo de *software* dentro do contexto dos projetos UCA e OLPC. Conforme detalhado nas seções 2.3.1, 2.3.2 e 2.3.3 e no anexo A, estes projetos partem do sistema operacional GNU/Linux, gerando requisitos similares. Outro ponto importante desta semelhança é o fato de muitos dos aplicativos poderem ser versões modificadas de aplicativos já difundidos para plataformas *desktop*, de forma que este trabalho também aborda o porte de aplicativos disponíveis em desktop para as plataformas disponíveis nos projetos. Por outro lado, o desenvolvimento colaborativo feito em comunidades de *software* livre tem características distintas dos modelos de Engenharia de *Software* cuja premissa é uma equipe presencial. Nas próximas seções, são detalhados os requisitos (das plataformas, de aprendizagem e dos usuários) e as etapas deste modelo de processo de desenvolvimento.

4.2 Premissas e requisitos

Ao fazer a modelagem de um processo voltado ao desenvolvimento de *software* para os projetos UCA e OLPC, assumem-se algumas premissas com relação aos dispositivos disponíveis e ao público-alvo dos produtos deste processo. Por isso, explicitam-se as seguintes características da plataforma:

- ser portátil;
- ter tela reduzida (cerca de 7 polegadas);
- ser aplicada em um público em idade escolar (de 6 a 14 anos).

Embora as características acima limitem as condições de aplicação, é importante frisar que os dispositivos de entrada usados poderão variar de um aplicativo para outro. Para as plataformas disponíveis nos projetos UCA e OLPC, os dispositivos de entrada são os seguintes:

- teclado [1];
- dispositivo apontador (*mouse* ou *touchpad*);
- câmera;
- microfone.

Existe também a possibilidade de conexão de outros dispositivos de entrada através das portas USB. No caso da plataforma Classmate, uma "caneta digital" pode ser conectada à porta USB e ser convertida para imagem, de forma que o usuário desenha do papel para o computador. O mesmo poderia acontecer para outros aparelhos de aquisição de sinais, como uma placa Arduino [2] ou o brinquedo WeDo da empresa Lego (VENANCIO et al., 2008b, 2008a).

Os requisitos físicos do tamanho da tela e da portabilidade são bastante importantes pelo campo de aplicação deste trabalho. As plataformas computacionais usadas nos projetos mais recentes de implantação de tecnologias móveis nas escolas públicas (em particular

[1] o teclado costuma ser considerado de tamanho pequeno em função da dimensão total do aparelho
[2] `http:arduino.cc/`

o projeto UCA no Brasil) dispõem de telas pequenas e podem ser facilmente carregadas pelos estudantes em questão.

No tocante ao sistema operacional, sabe-se que se trata de alguma ramificação de GNU/Linux. No projeto UCA, esta adoção faz parte da licitação das máquinas, e, no projeto OLPC, há uma distribuição GNU/Linux própria para seu *hardware*. Uma possibilidade que surge na adoção do ambiente Sugar (criado dentro do projeto OLPC) é a criação de aplicativos colaborativos: o ambiente foi concebido de forma a facilitar este desenvolvimento por já conter bibliotecas para este fim.

A especificação da licitação do projeto UCA não determina que as máquinas tenham de ter o ambiente Sugar instalado; o modelo vencedor da licitação de 2008 vinha com uma distribuição GNU/Linux conhecida como Metasys. Mesmo assim, a possibilidade de se utilizar as bibliotecas de *software* colaborativo do ambiente Sugar ainda persiste, pois a maior diferença entre estes ambientes está na interface gráfica e não nas partes mais fundamentais do sistema. Além disso, os Classmate também contam com dispositivo de rede compatível com redes *Mesh* (INTERNATIONAL SYST, 2009).

Por serem baseados no núcleo Linux e serem projetos *open source*, muitos aplicativos, bibliotecas e recursos disponíveis nas diversas distribuições GNU/Linux ficam disponíveis para estes dois ambientes gráficos. Entre as bibliotecas disponíveis, estão o D-Bus (biblioteca para comunicação entre processos) e o Telepathy (biblioteca para uso de protocolos de comunicação diversos), que juntas permitem a construção de aplicativos colaborativos tal qual o Sugar. Desta forma, é *possível* instalar estas bibliotecas na distribuição vencedora da licitação e criar aplicativos colaborativos compatíveis com aqueles do ambiente Sugar.

O *hardware* especificado na licitação do projeto UCA, determina valores específicos para capacidade de armazenamento, memória e conectividade (estes valores são detalhados no anexo A). Os valores colocados na licitação são compatíveis com os *netbooks* disponíveis no mercado hoje em dia, mas na época de lançamento da licitação poucos equipamentos eram capazes de cumprir as exigências em termos de *hardware*. Apesar da especificação para a compra ser bastante minuciosa quanto ao *hardware*, a especificação para o sistema operacional e para o conjunto de *software* instalado é bastante branda. A exigência para o sistema operacional é ser GNU/Linux, e a especificação para o conjunto de *software* instalado é baseado em categorias simples, como editor de texto e jogos educacionais. Desta forma, várias distribuições GNU/Linux são capazes de cumprir as

exigências de *software*. A tabela 4.1 sintetiza as características para as plataformas-alvo do desenvolvimento dos aplicativos.

Tabela 4.1: Principais caraterísticas dos computadores móveis para o projeto UCA.

Dispositivos de entrada	Teclado padrão ABNT-2
	Touchpad
	Câmera
	Microfone
	Entradas USB
Sistema operacional	GNU/Linux
Hardware	Conectividade compatível com IEEE 802.11 b/g/s
	Memória RAM de pelo menos 256MB

Em termos de requisitos de *software*, a plataforma impõe algumas restrições para o desenvolvedor, como usar pouca memória RAM e pouco espaço em disco, pois os valores especificados são bem menores que os disponíveis em outros computadores portáteis. Além de a capacidade ser menor do que a disponível em outras plataformas, o fato de o armazenamento ser feito usando memória Flashq implica numa vida útil menor em capacidade de operações de escrita (comparativamente com tecnologias de disco).

Levando em consideração o contexto do *software* livre, o desenvolvimento das aplicações tende a ser feito em conjunto com uma comunidade de desenvolvedores. Desta forma, a dinâmica das equipes de desenvolvimento passa a ter **características de desenvolvimento distribuído**. Neste trabalho, assume-se que o desenvolvimento do aplicativo será feito como *software* livre, pois esta é a realidade a que estão submetidas as iniciativas do projeto UCA e OLPC. Outro ponto muito importante no projeto do aplicativo é a *abordagem centrada no usuário*: Sharples, Corlett e Westmancott (2002), Kukulska-Hulme et al. (2009) destacam a importância do projeto de uma tecnologia voltada para Educação ser concebida desta forma. A teoria de Aprendizagem Móvel dá suporte a esta afirmação, principalmente por defender o uso de tecnologias móveis e adaptadas ao contexto como ferramentas de aprendizagem (BULL et al., 2005; VAVOULA; SHARPLES, 2002; SYVÄNEN et al., 2005). Portanto, a proposta desta pesquisa de modelagem do desenvolvimento para aplicativos voltados aos projetos UCA e OLPC parte da Engenharia sócio-cognitiva e da Programação Extrema (vide seções 2.4.1 e 2.4.2.1), mas considerando como é a interação com a comunidade.

O entendimento e a aplicação da teoria de Aprendizagem Móvel, entretanto, exigem um nível de especialidade razoavelmente raro, pois a teoria alia aspectos pedagógicos, cognitivos e de engenharia. É pouco realístico esperar de uma equipe de programadores esta gama de conhecimento. Desta forma, parte-se da premissa de que é necessário haver pelo menos um especialista em Aprendizagem Móvel na equipe de desenvolvimento, especialmente para a concepção do aplicativo (caso seja um aplicativo inédito) e para fazer testes com usuários. A participação deste especialista justifica-se por ser necessário entender as necessidades cognitivas do público-alvo e o contexto de aplicação, conforme preconizam a Aprendizagem Móvel e a Engenharia sócio-cognitiva (NAISMITH et al., 2004; SHARPLES et al., 2002).

Considera-se também que este aplicativo é desenvolvido por um grupo presencial apoiado por uma comunidade de *software* livre ou por uma comunidade apoiada por um grupo presencial. Nas duas possibilidades, o grupo presencial deve usar o modelo de desenvolvimento da Engenharia sócio-cognitiva ou da Programação Extrema. Nesta pesquisa, consideram-se apenas estes modelos de desenvolvimento; mas ambos são baseados em métodos iterativos (LARMAN; BASILI, 2003; LAYMAN et al., 2006; SHARPLES et al., 2002), o que leva a crer que outros modelos de desenvolvimento seriam passíveis desta aplicação, como outros métodos ágeis. Os métodos foram escolhidos baseados na experiência do autor neste tipo de desenvolvimento e no relato da literatura de um método concebido para aplicação em implementação de *software* para aprendizagem.

A tabela 4.2 resume os requisitos e premissas assumidas para o desenvolvimento de *software* pertinente ao projeto UCA. Nesta tabela, o termo "especialista" refere-se a alguém capaz de **analisar o uso da ferramenta** desenvolvida no contexto escolar.

Tabela 4.2: Resumo dos requisitos e premissas para desenvolvimento de *software*.

Características físicas	Portátil
	Tela pequena (cerca de 7 polegadas)
Restrições da plataforma para o aplicativo	Evitar ocupar muita memória RAM
	Evitar ocupar muito espaço em disco
	Evitar gerar arquivos temporários
Desenvolvimento	Ser *software* livre
	Abordagem centrada no usuário
	Engenharia sócio-cognitiva ou Programação Extrema
	Participação de especialista no desenvolvimento

A adoção do desenvolvimento nos termos do *software* livre, ou do desenvolvimento distribuído, torna algumas ferramentas fundamentais. Estas ferramentas visam garantir a **coordenação** efetiva da **equipe virtual** (ou seja, do grupo de desenvolvedores e da comunidade), justamente por estabelecer uma base de comunicação (GUTWIN; PENNER; SCHNEIDER, 2004; BARCELLINI et al., 2008). O desenvolvimento deve estar, portanto, apoiado em ferramentas capazes de prover:

- lista de *emails*;
- canal de *chat*;
- repositório de código;
- sistemas de acompanhamento (*issue/bug tracker*);
- páginas para documentação do projeto.

Ressalta-se, entretanto, que diversos serviços disponíveis (e gratuitos) na Web possuem estas funcionalidades; por exemplo: SourceForge [3], Free Software Fundation [4], Portal do Software Público [5], GitHub [6], Google Code [7], Bit Bucket [8]. As ferramentas de comunicação mais comuns em comunidades de *software* livre são simples, quase sempre baseadas em texto. Por isso, é possível, e talvez necessário para algumas equipes de desenvolvedores, o uso de ferramentas mais complexas para melhorar a coordenação entre os desenvolvedores (GUTWIN; PENNER; SCHNEIDER, 2004). No contexto de desenvolvimento distribuído de *software* para empresas (tanto em *outsourcing* quanto em *offshoring*), são reportadas outras formas de comunicação entre as equipes, como vídeo conferência, telefonemas ou até mesmo eventuais reuniões presenciais (CASEY, 2010).

4.3 Descrição

As seções subsequentes procuram modelar a interação de uma equipe trabalhando segundo um método já estabelecido para uma equipe presencial interagindo com uma

[3] http://www.sf.net/
[4] http://www.fsf.org/
[5] http://www.softwarepublico.gov.br/
[6] http://www.github.com/
[7] http://code.google.com/
[8] http://www.bitbucket.com/

comunidade de *software* livre. A ênfase fica, portanto, em estabelecer a conexão com o caráter distribuído e assíncrono do desenvolvimento colaborativo.

4.3.1 Modelagem para a Programação Extrema

Conforme destacado por Layman et al. (2006) e relatado na seção 3.3, o valor da comunicação aumenta bastante no contexto do desenvolvimento colaborativo e distribuído. Por isso, a comunicação entre os membros da equipe virtual tem grande importância para o funcionamento adequado do método.

Tal qual a descrição da programação extrema, esta modelagem não adota a sistematização em etapas sequenciais, mas sim a adoção de *valores*, *princípios* e *práticas*. Assim como proposto por Beck e Andres (2004), os valores, princípios e práticas que modelam o desenvolvimento de aplicativos para o contexto do projeto UCA não precisa ser adotado em sua totalidade. De acordo com a necessidade do projeto e/ou da equipe, pode-se adotar um subconjunto do que está proposto nas seções a seguir. A única exceção fica para o princípio da **diversidade**, pois a figura do especialista é tida como absolutamente necessária para o desenvolvimento para o projeto UCA. Mais uma vez, usa-se o termo "especialista" para fazer referência a alguém capaz de **analisar o uso da ferramenta** desenvolvida quando esta estiver em uso. É preciso salientar, também, a importância do contato direto da equipe de desenvolvimento com este especialista; daí a obrigatoriedade do princípio da diversidade.

Para este modelo, o planejamento no tempo é bastante livre, mas é possível que a implementação do aplicativo esteja submetida ao calendário da comunidade. Se este for o caso, a equipe deverá ajustar suas práticas ao calendário da comunidade, geralmente através do ajuste do ciclo trimestral e do ciclo semanal.

4.3.1.1 Valores

Os valores propostos por Beck e Andres (2004) - *Comunicação, Simplicidade, Realimentação, Coragem e Respeito* - **têm validade** neste contexto de desenvolvimento, mas o valor *Comunicação* deve ser revisto. Além disso, outros valores devem ser acrescentados para refletir a interação com a comunidade de *software* livre e as especificidades da Educação.

As *modificações* no valor Comunicação são as seguintes:

- Comunicação: além da comunicação interna da equipe ter a mesma importância destacada originalmente, a equipe deverá cuidar da comunicação com a comunidade de *software* livre. Assim, a equipe presencial deverá encontrar meios para trocar informações com seus colaboradores da comunidade.

Os *novos valores* propostos são:

- Visibilidade: o estado atual do projeto e seu histórico sempre devem estar acessíveis para quem está dentro e para quem está fora da equipe virtual. As perguntas "em que ponto este projeto está?" e "qual o objetivo deste desenvolvimento?" devem ser respondidas com facilidade.

- Transparência: as decisões do projeto devem seguir critérios conhecidos e os responsáveis por cada tarefa também devem ser conhecidos. A comunidade colabora de maneira voluntária, por isso sempre deve ter acesso aos rumos que o projeto toma e vai tomar. Ao abrir espaço para o debate dos rumos do projeto, pode-se abrir espaço para angariar novos voluntários.

- Autoria, Criatividade e Colaboração: as ferramentas criadas devem favorecer a aprendizagem do estudante. As tecnologias móveis favorecem muito o desenvolvimento de aplicativos sensíveis ao contexto (BULL et al., 2005; SHARPLES, 2000) e capazes de favorecer a criatividade através da autoria e colaboração (PAPERT, 1999; DILLENBOURG, 1999; RESNICK et al., 2009). Ao projetar um aplicativo, este valor é muito importante para determinar como o *software* se insere no contexto escolar.

- Mobilidade: em se tratando de tecnologias móveis, o fato de a tecnologia poder ser movida é importante. Sob a visão da Aprendizagem Móvel, entretanto, o estudante tem sua mobilidade também considerada com relação ao contexto, tempo e espaço (VAVOULA; SHARPLES, 2002). Por isso, a mobilidade pode ser um critério para analisar como o aplicativo será usado.

4.3.1.2 Princípios

Os princípios estão no caminho entre os valores e as práticas da Programação Extrema, sendo aplicadas nos momentos em que as práticas não fazem sentido ou em que os valores

são demasiadamente abstratos. No caso do modelo proposto nesta pesquisa, os princípios também têm a função de estabelecer pontos a serem alcançados ou critérios recorrentes nas decisões.

Os princípios propostos originalmente também têm validade nas condições deste trabalho, mas entende-se que a adoção do princípio da *Diversidade* passa a ser **obrigatório** ao invés de opcional. Sendo assim, as modificações deste princípio são as seguintes:

- Diversidade: o time de desenvolvimento deve ter competências diversificadas, ou seja, além da variedade de habilidades entre os programadores, é necessário haver dentro do time profissionais ligados à Educação. Esta pessoa é quem vai avaliar o uso do aplicativo e sua relevância para o contexto da aprendizagem, por isso, entende-se que sua formação deve estar na área de Aprendizagem Móvel, Informática na Educação ou Educação. Sua atuação será no cotidiano da equipe de desenvolvimento, desempenhando o papel de representante do cliente destacado por Layman et al. (2006).

Outros princípios devem ser acrescentados aos originais propostos por Beck e Andres (2004), a fim de aperfeiçoar a participação da comunidade de *software* livre e a aplicação da ferramenta desenvolvida para os aprendizes.

- Registro: os passos dados dentro da equipe presencial (como reuniões e testes com usuários) devem ser registrados de forma a poderem ser acessados pela comunidade. Vídeos e relatos descritivos (como em *wikis*, *emails* e *blogs*) são bons para esta finalidade.

- Usabilidade: o projeto de interface tem grande relevância neste contexto, principalmente no tocante à escolha das mensagens para o usuário. Considerando que o *netbook* será usado por aprendizes entre 6 e 14 anos, não faz sentido colocar mensagens textuais se o aplicativo em questão for usado por crianças não-alfabetizadas, por exemplo. Além disso, pode ser que existam recomendações de interface, tal qual as relatadas nas seções 2.4.3.2 e 3.2, devendo ser seguidas pelos projetistas do *software*.

- Desempenho: a especificação do *hardware* das máquinas adotadas no projeto UCA estão bastante abaixo dos computadores mais modernos disponíveis no mercado.

Por isso, projetar um aplicativo com bom desempenho *nas plataformas escolhidas* é fundamental.

- Comunidade: o sucesso do desenvolvimento depende da colaboração dentro da equipe virtual. Por isso, é importante valorizar a participação dos voluntários na comunidade por mais simples que seja seu papel. Além disso, é importante dar retorno aos resultados alcançados para todos os envolvidos no projeto, ou seja, mostrar a todos os membros as metas alcançadas, resultados de testes, implantação em escolas e tudo o mais relacionado aos desdobramentos do desenvolvimento.

- Aprendizagem: atender o público escolar é o principal motivo de existência do projeto UCA. Por isso, é necessário que os desenvolvedores e todos os outros membros da equipe virtual estejam cientes o tempo todo da necessidade de se agregar valor à aprendizagem.

4.3.1.3 Práticas

As prática propostas estão muito associadas com o relacionamento com a comunidade e com a participação de usuários, pois não há necessidade de mudanças muito específicas na implementação em si. Entretanto, devido a ajustes do calendário das comunidades de *software* livre, é preciso ajustar as práticas *Ciclo trimestral* e *Ciclo semanal*. Todas as outras práticas podem ser aplicadas da mesma forma como proposta por Beck e Andres (2004).

- Ciclo trimestral: Além do planejamento temático com as histórias, é desejável incluir testes com usuários no **final** de cada iteração para avaliar o quão a ferramenta está próxima do que se deseja para fins educativos.

- Ciclo semanal: ao definir quais as tarefas para a equipe presencial, o líder da equipe também atribui responsabilidades também para os outros membros da equipe virtual. Atribuição de tarefas, tanto dos membros presenciais quanto dos remotos, deve ser feita através do sistema de acompanhamento (*issue/bug track*) do projeto.

Em adição à práticas originais, são propostas novas práticas que refletem atividades rotineiras da equipe de desenvolvimento e decisões relacionadas aos rumos do desenvolvimento.

- *Chat*: todos os desenvolvedores devem participar o máximo possível dos canais de *chat* disponíveis na comunidade, a fim de esclarecer dúvidas e tomar decisões relacionadas ao código.

- Lista de *email*: as comunidades de *software* livre usam a lista para registrar boa parte das informações. A lista de discussão é muito usada a fim de debater decisões com implicações de longo prazo (BARCELLINI et al., 2008). Desta forma, todos os desenvoledores devem estar a par do conteúdo da lista de discussão do projeto.

- Representante: um membro da equipe presencial deverá ser responsável por estabelecer a ponte entre a comunidade e a equipe presencial. Este membro deverá ser capaz de se comunicar em inglês, já que a abrangência dos projetos pode ser mundial.

- Testes com usuários: estes testes são bastante relevantes para avaliar o impacto do aplicativo na aprendizagem. O local mais adequado para os testes é justamente a escola, onde alunos e professores estão amplamente disponíveis. Embora os estudantes sejam superiores no número de usuários, é importante que os testes também incluam os professores em algum momento (separadamente ou juntamente com os estudantes), pois professores atuam na proposição de atividades para os alunos.

- Arquitetura distribuída: com a possibilidade de construir aplicativos colaborativos, a equipe de desenvolvimento deve estar atenta ao uso de arquiteturas distribuídas. Nas plataformas com redes *Mesh*, a troca de pacotes ganha bastante destaque por conta das características da topologia da rede e do número de possíveis saltos que os pacotes têm de dar para chegar ao destino final (ABELÉM et al., 2007; CARRANO et al., 2007). Assim, a arquitetura deve ser projetada a fim de trocar o menor número de pacotes possível pela rede.

4.3.2 Modelagem para a Engenharia sócio-cognitiva

Tomando como base o método da Engenharia sócio-cognitiva, esta proposta procura estender suas etapas e práticas. A extensão é feita a fim de modelar a interação de uma equipe *presencial* desenvolvendo segundo o modelo da Engenharia sócio-cognitiva com uma comunidade de *software* livre. O método descrito por Sharples et al. (2002) já prevê a participação de profissionais capazes de entender o contexto de uso da ferramenta digital, por isso não há um novo papel a ser criado. Nos tópicos a seguir, o profissional com este perfil será chamado de "especialista".

Analogamente ao apresentado na seção 2.4.1, o ciclo de desenvolvimento da Engenharia sócio-cognitiva pode ser resumido como apresentado na figura 4.1.

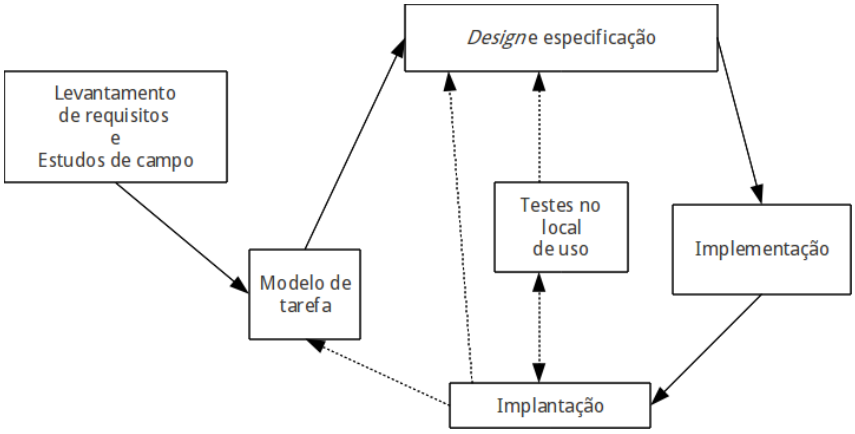

Figura 4.1: Simplificação do processo de desenvolvimento da Engenharia sócio-cognitiva.

Esta simplificação pode ser descrita pelas seguintes etapas:

1. Levantamento de requisitos geral e estudos de campo;
2. Modelagem de tarefa;
3. Design e especificação;
4. Implementação do sistema;
5. Implantação no local de uso;
6. Testes.

As etapas 3 a 6 são repetidas indefinidamente; ficando a critério da equipe de desenvolvimento quantas vezes o ciclo será percorrido. Não se deve esquecer de que a duração de cada ciclo está ligada ao calendário de lançamento da comunidade, pois se trata de um desenvolvimento colaborativo. O calendário de lançamentos pode ter etapas muito bem definidas, como é o caso do projeto OLPC, cujo esquema foi discutido na seção 3.2. Pode acontecer, entretanto, de ser necessário readequar o modelo de tarefa estabelecido anteriormente (etapa 2), pois a ideia é implantar o sistema no seu local projetado de uso. Segundo a proposta da Engenharia sócio-cognitiva, os testes são feitos em todas as etapas; apesar disso, os testes da etapa 6 na listagem anterior referem-se apenas aos testes no local de implantação, ficando implícita a ideia de testagem contínua do processo.

Um detalhe importante a destacar do processo é a realimentação **contínua** provida pelo desenvolvimento colaborativo: a comunidade pode contribuir com testes funcionais dos aplicativos, relato de defeitos e com implementação de funcionalidades no código, por exemplo. As etapas são detalhadas nas seções a seguir.

4.3.2.1 Levantamento de requisitos geral e estudos de campo

O planejamento inicial tem o intuito de estabelecer as bases do desenvolvimento, como definição da equipe, prazos e ambiente de desenvolvimento. Através do levantamento de requisitos e dos estudos de campo, são conhecidas as caraterísticas dos usuários e quais os principais objetivos a se atingir com o uso da tecnologia. As características básicas da plataforma, bem como alguns requisitos básicos estão resumidos nas tabelas 4.1 e 4.2.

Esta etapa é executada exclusivamente pela equipe presencial, ou seja, pela equipe com o(s) especialista(s) em Aprendizagem, pois é necessário entender o contexto de uso da tecnologia. A equipe presencial deve, entretanto, documentar este processo e suas conclusões e divulgá-los nos canais de documentação do sistema eletrônico escolhido (alguns dos serviços disponíveis para este fim foram listados na seção 4.2).

4.3.2.2 Modelagem de tarefa

O modelo de tarefas deve sintetizar como as pessoas envolvidas agem dentro do contexto atual, como interagem entre si e mapear os contextos. Este modelo também deverá representar como as pessoas externalizam seu trabalho (como notas e diagramas), as regras envolvidas no exercício das atividades e a terminologia geral. O modelo de tarefas tem a função de indicar como os usuários serão capazes de cumprir seus objetivos usando o aplicativo; sendo também uma tarefa para a equipe presencial.

Embora esta etapa tenha um caráter mais interno à equipe de especialistas, seus resultados devem ser divulgados para a comunidade. A representação pode seguir qualquer esquema preferido pela equipe presencial, mas as comunidades de *software* livre tendem a usar mais as ferramentas textuais e a preferir recursos informais (SCACCHI, 2007). Para favorecer a colaboração da comunidade, as ferramentas descritivas, como um *wiki*, podem ser mais adequadas para criar os modelos de tarefa.

4.3.2.3 Design e especificação

Tipicamente, o *design* em um projeto de *software* inclui a definição da arquitetura dos módulos do sistema e também das interfaces de usuário. No caso específico da comunidade OLPC, há contribuição relativamente mais frequente em termos de interface de usuário, membros da própria comunidade enviam esboços de interface para os líderes do projeto. É possível existirem também as recomendações de interface, caso o desenvolvimento do aplicativo esteja inserido em um contexto de agregação de *software* (conforme discutido na seção 2.4.3.2). Neste caso, o projeto de interface do aplicativo terá uma aparência quase pré-definida, já que uma das ideia das recomendações de interface é manter um aparência padronizada entre os aplicativos. A arquitetura, porém, é o ponto abstrato em que mais há contribuição da comunidade, sendo alvo frequente de debate nas listas de discussão dos desenvolvedores (BARCELLINI et al., 2008). Os produtos de *design*, como esboços de tela e imagens e ícones da interface, devem ser disponibilizados para a comunidade; há comunidades que preferem ter estes arquivos em formatos livres, como SVG ou PNG.

A especificação, por outro lado, determina as atividades de maneira objetiva para os desenvolvedores, como qual a função de cada módulo do sistema ou qual desenvolvedor fica responsável por cada parte da implementação. A especificação do sistema deve ser feita usando os sistemas de acompanhamento, ou seja, a especificação gera *tickets* para a comunidade. Com a evolução do projeto, a tendência é ter mais desenvolvedores voluntários recebendo as tarefas geradas na especificação.

No caso de o aplicativo desenvolvido permitir trabalho colaborativo para os usuários, a recomendação é adotar uma arquitetura de objetos distribuídos fazendo uso da biblioteca D-Tubes para implementação da comunicação entre as instâncias do aplicativo distribuído.

4.3.2.4 Implementação do sistema

A implementação pode usar quaisquer linguagens de programação suportadas pela plataforma; entretanto, linguagens e bibliotecas podem ganhar preferência de acordo cada plataforma. Isso pode variar segundo as recomendações de interface e as especificações do sistema operacional disponível para as plataformas. Conforme apresentado na seção 2.3, a associação entre o *hardware* e o *software* é bastante estreita, e este acoplamento limita as decisões de implementação do sistema.

No caso do ambiente Sugar, a linguagem mais apropriada para trabalhar é Python,

embora outras linguagens possam ser usadas. Além disso, é necessário implementar usando a biblioteca gráfica GTK, pois é a que está disponível no ambiente. A maior vantagem de usar Python no ambiente Sugar é a integração do aplicativo com o resto do ambiente.

Nos ambientes MeeGo e Metasys, a restrição para o desenvolvimento é usar preferencialmente a biblioteca gráfica Qt. Não há restrições para as linguagens de programação, mas a principal linguagem das aplicações disponíveis é C++.

Há a possibilidade de construir aplicativos colaborativos nas condições estabelecidas para o projeto UCA. É preciso ter atenção, entretanto, à troca de pacotes decorrente do uso da rede em malha.

Esta é a fase com maior contribuição em comunidades de *software* livre, pois a maior parte dos participantes é justamente de pessoal tecnicamente capacitado (GUTWIN; PENNER; SCHNEIDER, 2004). Com relação ao escopo do trabalho desta fase, o propósito é desenvolver aquilo que tiver sido determinado dentro da especificação. O produto desta fase é um protótipo ou uma versão do aplicativo.

4.3.2.5 Implantação no local de uso

Esta fase consiste em levar o aplicativo produzido para seu local de aplicação; no contexto do projeto UCA, isto quase sempre significa levar a tecnologia para alguma escola. Este ponto pode exigir formação com professores para o uso ser efetivo no futuro, afinal, não se pode esperar que o professor comece a usar dentro da sua proposta pedagógica sem conhecer suas ferramentas.

Não há participação possível da comunidade ligada ao desenvolvimento do aplicativo, mas a *documentação em vídeo* do aplicativo sendo usado na escola pode ser bastante motivador para os membros da comunidade que não participaram da implantação. Este registro implica em haver permissão para filmagem na referida escola/ambiente.

4.3.2.6 Testes

Os testes com o público-alvo não são muito comuns em todos os ciclos de desenvolvimento, pois implicam em custos adicionais para a execução dos testes. Quando testes de usabilidade são possíveis, é recomendado testar em escolas, já que o público-alvo está amplamente localizado neste ambiente. Embora os estudantes sejam superiores no número

de usuários, é importante que os testes também incluam os professores em algum momento (separadamente ou juntamente com os estudantes). Esta afirmação tem valor pois os professores atuam na proposição de atividades escolares. Desta forma, a inclusão de professores nos testes de usabilidade permite entender melhor o uso do aplicativo dentro dos paradigmas de aprendizagem e da proposta anterior do modelo de tarefa.

Os testes nos locais de uso também geram novos requisitos, pois está sendo constituído um novo sistema sócio-técnico (SHARPLES et al., 2002). A responsabilidade da condução dos testes com usuários e das observações pertinentes é tarefa do(s) especialista(s) da equipe presencial. Embora estes não tenham necessariamente reflexo para os desenvolvedores da comunidade, também é recomendada a divulgação dos resultados e conclusões principais através dos canais de comunicação disponíveis (como lista de *email* ou *wiki*).

Dependendo das características e dos papéis da comunidade, é possível que o time de qualidade possa executar os testes com usuário de maneira independente. Se o relato dos testes com usuário tiverem sido bem executados, seus resultados podem ser usados da mesma maneira que os testes feitos pela equipe especializada.

4.4 Observações finais

Este capítulo apresentou as considerações a respeito do desenvolvimento de aplicativos voltados para as plataformas móveis dos projetos UCA e OLPC. A proposta deste trabalho leva em consideração como se daria a interação dos especialistas na questão do uso com os desenvolvedores de comunidades de *software* livre.

Para a programação extrema, foi proposto um conjunto de práticas, princípios e valores para modelar a interação com uma comunidade de *software* livre dentro do contexto do projeto UCA. Já para a Engenharia sócio-cognitiva, a modelagem vai no sentido de acrescentar novos passos dentro das etapas originais de forma a facilitar a interação com a comunidade de *software* livre.

A tabela 4.3 resume o modelo proposto para o método da Programação Extrema, mostrando novos **valores**, **princípios** e **práticas**, além de considerações sobre os mesmos no contexto dos projetos UCA e OLPC. Os itens destacados em negrito indicam que o mesmo valor, princípio ou prática tiveram alterações em relação ao apresentado em Beck e Andres (2004).

Tabela 4.3: Síntese da proposta para Programação Extrema. As palavras em negrito destacam valores, princípios ou práticas ajustados nesta proposta.

DESCRIÇÃO	ORIGINAL	PROPOSIÇÕES ADICIONAIS
Valores	**Comunicação** Simplicidade Realimentação Coragem Respeito	**Comunicação** Visibilidade Transparência Autoria, Criatividade e Colaboração Mobilidade
Princípios	**Diversidade** Humanidade Economia Benefício mútuo Auto semelhança Melhoria Reflexão Fluxo Oportunidade Redundância Falha Qualidade Passos pequenos Responsabilidade aceita	**Diversidade** Registro Usabilidade Desempenho Comunidade Aprendizagem
Práticas	**Ciclo semanal** **Ciclo trimestral** Design incremental Sentar junto Time completo Área de trabalho informativa Trabalho energizado Programação pareada Histórias Folga Build de 10 minutos Integração contínua Testar antes de escrever código	**Ciclo trimestral** **Ciclo semanal** *Chat* Lista de *email* Representante Testes com usuários Arquitetura distribuída

Para o método da Engenharia sócio-cognitiva, a proposta traz considerações com relação à **comunicação** com a comunidade, um aspecto destacado por Layman et al. (2006) ao tratar da interação entre equipes distribuídas. A tabela 4.4 mostra as considerações relevantes para o método da Engenharia sócio-cognitiva, proposto por Sharples et al. (2002), no contexto dos projetos UCA e OLPC.

Tabela 4.4: Síntese da proposta para a Engenharia sócio-cognitiva.

ETAPA	MODIFICAÇÕES PROPOSTAS
Levantamento de requisitos geral e estudos de campo	Divulgar requisitos e principais conclusões nos canais de comunicação da comunidade
Modelagem de tarefa	Descrever o modelo de tarefa com ferramentas textuais
Design e especificação	Disponibilizar esboços de interface e ícones (pode ter de obedecer recomendações de interface) Usar sistema de acompanhamento da comunidade para especificar o sistema Usar arquitetura distribuída se o aplicativo for colaborativo
Implementação do sistema	A tendência é receber contribuições da comunidade
Implantação no local de uso	Não há participação da comunidade, mas registros em vídeo podem motivar participação na comunidade
Testes	Não há participação da comunidade (exceto se houver time de qualidade) Executar testes de usabilidade com estudantes e professores em escolas

5 CONCLUSÕES

Esta pesquisa visou esclarecer os pontos principais para o desenvolvimento de aplicativos dentro do contexto brasileiro de Educação. Para tal, analisaram-se quais as iniciativas de uso da tecnologia no Brasil e quais suas principais características.

O panorama brasileiro indica para a utilização massiva de *netbooks* – fato também observado em outros países do mundo – de forma que muitas considerações sobre Educação a partir destas tecnologias encontram-se na Aprendizagem Móvel. Além das consequências para mobilidade no espaço, tempo e contexto (KUKULSKA-HULME et al., 2009; BULL et al., 2005; SYVÄNEN et al., 2005; SHARPLES, 2000), a aprendizagem suportada por tecnologia também tem implicações para o desenvolvimento da criatividade e do trabalho em equipe (RESNICK, 2002; DILLENBOURG, 1999).

O embasamento em teorias de Aprendizagem permite tecer considerações sobre o *uso* da tecnologia no contexto escolar, tais como a importância do projeto de *software* colaborativo (no sentido de permitir o trabalho em equipe) e da autoria em ferramentas digitais. Além disso, conhecer estas teorias e a realidade escolar são elementos importantes para entender os usuários (ou seja, estudantes e professores), e, eventualmente, modelar seu comportamento.

O surgimento e a adoção de plataformas móveis com novas características de *hardware* e alta mobilidade (tanto pelas dimensões quanto pelo peso) levaram ao desenvolvimento de ambientes computacionais dedicados, como o MeeGo e o Sugar (detalhados na seção 2.3). Estes ambientes foram projetados de forma a explorar as características de mobilidade e conectividade das plataformas a que se destinam. Desta forma, suas interfaces têm projetos inovadores do ponto de vista de metáfora, explorando conceitos como colaboração e comunicação em rede.

Do ponto de vista de desenvolvimento de *software*, esta pesquisa estuda dois métodos criados para equipes presenciais, a Engenharia sócio-cognitiva e a Programação Extrema,

e o funcionamento de equipes distribuídas, com foco maior no funcionamento das comunidades de *software* livre. Prikladnicki e Audy (2010) indicam que os modelos de desenvolvimento para equipes distribuídas ainda não estão maduros, mesmo assim, comunidades de desenvolvimento colaborativo são consideradas boas fontes de informação para o assunto, principalmente para o ensino de Engenharia de *Software* (HAWTHORNE; PERRY, 2005; LIU, 2005). Estudando o desenvolvimento distribuído de *software*, Layman et al. (2006) relatou o uso de Programação Extrema, que é altamente baseada em estratégias de comunicação informal. Para o contexto distribuído, os autores destacam o papel fundamental da comunicação entre as equipes. Já no contexto do desenvolvimento colaborativo (ou seja, do *software* livre), Barcellini et al. (2008) destacam a importância da agilidade na comunicação, principalmente nos momentos de tomada de decisão.

A experiência do autor no desenvolvimento da ferramenta de desenho da plataforma Sugar também aponta a mesma importância para a estratégia de comunicação, conforme relatado na seção 3.3. Desta forma, esta pesquisa indica que a interação bem-sucedida entre equipes presenciais e comunidades de *software* livre depende do sucesso da comunicação, tal qual o que acontece no desenvolvimento distribuído para empresas. Para o caso de desenvolvimento colaborativo, as estratégias de colaboração síncrona não são possíveis, por isso, a estratégia de comunicação faz uso intenso de ferramentas de *chat*, de *email* e de sistemas de acompanhamento (*issue/bug tracker*). Estas estratégias foram usadas na implementação da ferramenta de desenho previamente mencionada, tendo tido uso bem-sucedido.

Tanto o projeto UCA quanto o projeto OLPC demandam profissionais de áreas técnicas e pedagógicas. O envolvimento direto do governo federal no projeto UCA, entretanto, traz algumas particularidades. A rede de formação de professores, abrangendo também universidades, é um diferencial proporcionado por este aspecto. Para o desenvolvimento de *software*, uma consequência é o surgimento de empresas para manutenção dos sistemas e aplicativos usados nas escolas participantes. A participação da International Syst no projeto UCA como integrante do consórcio vencedor é um exemplo muito bom para esta observação. Espera-se que estas diretrizes sejam capazes de auxiliar no projeto de novas tecnologias móveis voltadas para a aprendizagem, também para este exemplo.

Esta pesquisa também buscou fundamentar a interação entre uma equipe presencial e a comunidade de *software* livre através da proposta de equipes virtuais. Casey (2010) propõe o monitoramento de 4 fatores para equipes virtuais: coordenação, visibilidade,

comunicação e cooperação. Estes fatores são válidos também para o contexto do desenvolvimento colaborativo, mesmo tendo sido desenvolvidos para o contexto de *offshoring*, conforme a análise do trabalho de Gutwin, Penner e Schneider (2004) permite afirmar.

Assim, este trabalho assume o desenvolvimento em parceria com comunidades de *software* livre como uma premissa e provê diretrizes para a interação nesta colaboração. Assume-se também que uma equipe presencial opera segundo a Programação Extrema ou Engenharia sócio-cognitiva, contando com a colaboração de um especialista em Aprendizagem Móvel, Informática na Educação ou especialista em Educação. Depois dessas premissas, a pesquisa define uma série de requisitos que tem a ver com as plataformas de *hardware* compradas para o projeto UCA e desenvolvidas para o projeto OLPC, além de tecer considerações sobre o público-alvo. Todas estas colocações levam ao desenvolvimento de diretrizes para interação da equipe presencial com a comunidade de *software* livre.

Os 4 fatores de monitoramento propostos por Casey (2010) são observados na interação com as comunidades, tanto no caso da Programação Extrema quanto da Engenharia sócio-cognitiva. A **coordenação** e a **visibilidade** têm suporte das ferramentas comuns de acompanhamento das comunidades de *software* livre. A **comunicação** merece atenção com relação à estratégia necessária para fazer as ferramentas como *email* e *chat* funcionarem adequadamente. Já a **cooperação** não têm diretrizes específicas, visto que tem muito a ver com o envolvimento dos participantes. Gutwin, Penner e Schneider (2004), Casey (2010) indicam a necessidade de resolução caso a caso para este quesito e esta pesquisa apenas indica algumas possibilidades de resolução para motivar a participação nas comunidades.

Esta pesquisa procurou elaborar as diretrizes de forma a servir como guia para a criação e reengenharia de aplicativos para os projetos UCA e OLPC, respondendo, portanto, a pergunta norteadora (*o que desenvolvedores de software precisam saber para criar um novo aplicativo (ou portar um aplicativo já existente) no contexto dos projetos UCA e OLPC?*), apresentada na seção 1.1.

As diretrizes propostas consistem em **estender os métodos presenciais** mencionados anteriormente: a *Programação Extrema* e a *Engenharia sócio-cognitiva*. Para a Programação Extrema, foi proposto um conjunto adicional de **práticas**, **princípios** e **valores**, bem como modificações em relação a proposta original de Beck e Andres (2004) para modelar a interação com uma comunidade de *software* livre. Já para a *Engenharia sócio-*

cognitiva, a modelagem acrescenta **novas atividades** dentro das etapas propostas em Sharples et al. (2002) também de forma a orientar a interação com a comunidade de *software* livre. Para este método, as diretrizes estão centradas nas considerações com relação à *comunicação* com a comunidade, um aspecto destacado por Layman et al. (2006) ao tratar da interação entre equipes distribuídas.

5.1 Contribuições científicas

Este trabalho estudou como se dá o desenvolvimento colaborativo de *software*, usando a comunidade OLPC como ponto de partida. Como o contexto de aplicação e os objetivos dos projetos UCA e OLPC são muito semelhantes, as diretrizes servem para ambos os projetos. Considerando que equipes especializadas podem desenvolver *software* para Educação em métodos iterativos como XP ou Engenharia sócio-cognitiva, a pesquisa procurou relacionar como uma comunidade de *software* livre pode interagir com a equipe de especialistas.

As considerações construídas ao logo desta pesquisa são as principais contribuições científicas deste trabalho. O conjunto de diretrizes de desenvolvimento de *software* para os projetos UCA e OLPC foram concretizadas através de:

- aplicação dos fatores de monitoramento de "equipes virtuais" em uma comunidade de *software* livre;
- definição de requisitos específicos das plataformas e do público-alvo;
- modelos para interação entre uma equipe presencial, com apoio de especialista, e a comunidade de *software* livre.

Além disso, foram **estendidos** dois modelos de Engenharia de Software: a *Programação Extrema* e a *Engenharia sócio-cognitiva*. Para estas extensões, foram considerados os fatores de monitoramento de Casey (2010), as condições gerais dos projetos UCA e OLPC e a experiência do autor no desenvolvimento de aplicativos para estes projetos. Desta forma, procurou-se aplicar os modelos pedagógicos validados na Aprendizagem Móvel para projetar *software*. A lista das publicações feitas ao longo desta pesquisa nesta área estão no apêndice A.

Esta também proposta aborda aspectos diferentes dos trabalhos relacionados, no sentido de propor atuação direta na metodologia de trabalho das equipes. Desta forma, este trabalho está num nível mais próximo das atividades diárias das equipes de desenvolvimento e menos no nível da gestão.

5.2 Trabalhos futuros

Esta pesquisa foi elaborada a partir de projetos em Educação, mas sua aplicação em outras áreas ainda é possível. O trabalho em equipe tem sido cada vez mais enfatizado nas diversas atividades humanas, inclusive nas profissionais. Desta forma, o desenvolvimento de aplicativos que permitam a colaboração síncrona entre os usuários tem importância. Uma nova pesquisa traria à tona quais as questões mais relevantes para o desenvolvimento colaborativo de *software* em outras áreas, que não a Educação.

Esta pesquisa escolheu a Programação Extrema e a Engenharia sócio-cognitiva como métodos geradores, por conta da experiência do autor no desenvolvimento no desenvolvimento de *software* para Educação e no relato da literatura nesta mesma área. O trabalho de Larman e Basili (2003) sobre as origens de métodos iterativos aproximam outros métodos dos escolhidos para esta pesquisa. Desta forma, outros modelos de desenvolvimento poderiam ser usados na geração de diretrizes de aplicação no contexto dos projetos UCA e OLPC, considerando também a participação de comunidades de *software* livre.

Este trabalho não discutiu as implicações econômicas do modelo de desenvolvimento proposto. Manter uma equipe de programadores e especialistas trabalhando certamente envolve despesas de pessoal, mas esta pesquisa não discorre sobre este aspecto. O modelo está ajustado para uma realidade em que a equipe presencial encontra-se apoiada financeiramente por alguma instituição. A experiência de desenvolvimento da ferramenta de desenho para o ambiente Sugar é um exemplo desta dinâmica, no qual a Universidade de São Paulo viabilizou o desenvolvimento.

O trabalho voluntário na comunidade também está sujeito a variações sazonais; muitos projetos são abandonados por falta de programadores envolvidos. Este trabalho indica algumas estratégias para *manter* voluntários envolvidos através da *transparência* na condução do projeto. Apesar da transparência ser valorizada no desenvolvimento colaborativo de *software*, este aspecto não garante a participação de programadores na comunidade, pois a participação dos mesmos muitas vezes envolve desafios técnicos e fama (SCACCHI,

2007). Ainda assim, a força do trabalho voluntário fornecido pela comunidade de *software* livre poderia vir de estudantes de universidades em cadeiras de Engenharia de *Software*. Tal experiência é narrada por Goldman et al. (2004) na aplicação de Programação Extrema em laboratórios da disciplina. Os estudantes participam da implementação de sistemas em uso, tornando a experiência de desenvolvimento mais realística.

REFERÊNCIAS

ABELÉM, A. J. G. et al. Redes mesh: Mobilidade, qualidade de serviço e comunicação em grupo. In: *Livro de Minicursos do Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Belém, PA, Brasil: [s.n.], 2007. p. 59–112.

BAJARIN, T. *Jeff Hawkins and the World's First Netbook*. nov. 2008. Disponível em: <http://www.pcmag.com/article2/0,2817,2335072,00.asp>. Acesso em: 10 jun 2009.

BARCELLINI, F. et al. A socio-cognitive analysis of online design discussions in an open source software community. *Interacting with Computers*, v. 20, n. 1, p. 141–165, jan. 2008. ISSN 0953-5438.

BECK, K. Embracing change with extreme programming. *Computer*, v. 32, n. 10, p. 70–77, 1999. ISSN 00189162.

BECK, K.; ANDRES, C. *Extreme Programming Explained: Embrace Change*. 2. ed. Upper Saddle River, NJ, EUA: Addison-Wesley Professional, 2004. ISBN 0321278658.

BECK, K. et al. *Manifesto for Agile Software Development*. 2001. Disponível em: <http://agilemanifesto.org/>. Acesso em: 27 jan 2011.

BRASIL. Ministério da Educação. Secretaria de Ensino à Distância. *UCA | Um Computador por Aluno*. 2010. Disponível em: <http://www.uca.gov.br/institucional/projeto.jsp>. Acesso em: 7 jan 2011.

BULL, S. et al. Adapting to different needs in different locations: Handheld computers in university education. In: IEEE INTERNATIONAL WORKSHOP ON WIRELESS AND MOBILE TECHNOLOGIES IN EDUCATION. *Proceedings...*. Washington, DC, EUA: IEEE Computer Society, 2005. p. 48–52. ISBN 0-7695-2385-4.

CARRANO, R. et al. Mesh networks for digital inclusion - testing OLPC's XO mesh implementation. In: WORKSHOP DE SOFTWARE LIVRE DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO. *Anais...*. Porto Alegre, RS, Brasil, 2007.

CASEY, V. Virtual software team project management. *Journal of the Brazilian Computer Society*, v. 16, n. 2, p. 83–96, ago. 2010. ISSN 0104-6500.

CORRÊA, A. G. D. et al. Avaliação de aceitabilidade de um computador portátil de baixo custo por criança. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO. *Anais...*. Brasília, 2006. v. 1, p. 288–297.

DILLENBOURG, P. What do you mean by "collaborative learning"? In: DILLENBOURG, P. (Ed.). *Collaborative-learning: Cognitive and Computational Approaches*. Oxford, USA: Elsevier, 1999. p. 1–19.

ERAUT, M. Non-formal learning and tacit knowledge in professional work. *British Journal of Educational Psychology*, v. 70, p. 113–136, mar. 2000.

GOLDMAN, A. et al. Being extreme in the classroom: Experiences teaching XP. *Journal of the Brazilian Computer Society*, v. 10, n. 2, p. 5–21, 2004. ISSN 0104-6500.

GUTWIN, C.; PENNER, R.; SCHNEIDER, K. Group awareness in distributed software development. In: ACM CONFERENCE ON COMPUTER SUPPORTED COOPERATIVE WORK - CSCW '04. *Proceedings...*. Chicago, Illinois, USA, 2004. p. 72.

HAWTHORNE, M. J.; PERRY, D. E. Software engineering education in the era of outsourcing, distributed development, and open source software. In: THE 27TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING - ICSE '05. *Proceedings...*. St. Louis, MO, USA, 2005. p. 643.

HIGHSMITH, J.; COCKBURN, A. Agile software development: the business of innovation. *Computer*, v. 34, n. 9, p. 120–127, 2001. ISSN 0018-9162.

INTERNATIONAL SYST. *Metasys - Solução para Educação baseado nos Intel® Classmate PC*. 2009. Disponível em: <http://www.scribd.com/doc/22418337/Metasys-Solucao-para-Educacao-baseado-nos-Intel%C2%AE-Classmate-PC>. Acesso em: 11 fev 2011.

KUKULSKA-HULME, A. et al. Innovation in mobile learning: a european perspective. *International Journal of Mobile and Blended Learning*, v. 1, n. 1, p. 13–35, 2009.

LARMAN, C. *Agile and Iterative Development: A Manager's Guide*. [S.l.]: Addison-Wesley Professional, 2003. ISBN 0131111558.

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

LAYMAN, L. et al. Essential communication practices for extreme programming in a global software development team. *Information and Software Technology*, v. 48, n. 9, p. 781–794, set. 2006. ISSN 0950-5849.

LINUX FOUNDATION. *Introduction to the MeeGo Project*. [S.l.], 2010. 11 p. Disponível em: <http://wiki.meego.com/images/MeeGo_Introduction.pdf>. Acesso em: 22 jul 2010.

LIPPONEN, L. Exploring foundations for computer supported collaborative learning. In: COMPUTER SUPPORTED COLLABORATIVE LEARNING 2002 CONFERENCE. *Proceedings...*. Boulder, Colorado, USA, 2002. p. 72–81.

LIU, X. Collaborative global software development and education. In: THE 29TH ANNUAL INTERNATIONAL COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE, Edinburgh, Escócia. *Proceedings...*. Washington, DC, EUA: IEEE Computer Society, 2005. v. 1, p. 371. ISBN 0730-3157.

MAGALHÃES, D.; KNIGHT, P.; COSTA, E. M. da. Will the soccer world cup of 2014 help bridge the social gap through the promotion of ICT and e-government in Brazil? In: MIA, I.; SOUMITRA, D. (Ed.). *The Global Information Technology Report 2008-2009: Mobility in a Networked World*. SRO-Kundig. Geneva, Switzerland: World Economic Forum, 2009. p. 133–143. ISBN 9789295044197.

MARTINAZZO, A. A. G. et al. Testing the OLPC drawing activity: An usability report. In: IEEE INTERNATIONAL CONFERENCE ON ADVANCED LEARNING TECHNOLOGIES. *Proceedings...*. Santander, Spain, 2008. p. 844–846. ISBN 978-0-7695-3167-0.

NAISMITH, L. et al. *Report 11: Literature Review in Mobile Technologies and Learning*. Bristol, UK, dez. 2004. Disponível em: <http://www.futurelab.org.uk/resources/publications-reports-articles/literature-reviews/Literature-Review203/>. Acesso em: 29 mai 2009.

ONE LAPTOP PER CHILD. *One Laptop per Child (OLPC): Vision*. 2007. Disponível em: <http://laptop.org/en/vision/index.shtml>. Acesso em: 13 jun 2010.

PAPERT, S. Introduction: What is logo? and who needs it? In: *Logo Philosophy and Implementation*. [s.n.], 1999. p. V–XVI. ISBN 2-89371-494-3. Disponível em: <http://www.microworlds.com/support/logo-philosophy-papert.html>. Acesso em: 11 jun 2009.

PRIKLADNICKI, R.; AUDY, J. L. N. Process models in the practice of distributed software development: A systematic review of the literature. *Information and Software Technology*, v. 52, n. 8, p. 779–791, ago. 2010. ISSN 0950-5849.

PRIKLADNICKI, R.; AUDY, J. L. N.; SHULL, F. Patterns in effective distributed software development. *IEEE Software*, v. 27, n. 2, p. 12–15, 2010. ISSN 0740-7459.

RESNICK, M. Rethinking learning in the digital age. In: KIRKMAN, G. (Ed.). *The Global Information Technology Report: Readiness for the Networked World*. Oxford, USA: Oxford University Press, 2002. p. 32–37.

RESNICK, M. Thinking like a tree (and other forms of ecological thinking). *International Journal of Computers for Mathematical Learning*, v. 8, n. 1, p. 43–62, 2003.

RESNICK, M. et al. Scratch: programming for all. *Communications of the ACM*, v. 52, p. 60–67, nov. 2009. ISSN 0001-0782.

SATO, D. *Uso Eficaz de Métricas em Métodos Ágeis de Desenvolvimento de Software*. 100 f. Dissertação (Mestrado em Ciência da Computação) — Instituto de Matemática e Estatística da Universidade de São Paulo, São Paulo, Brasil, ago. 2007.

SCACCHI, W. Free/open source software development. In: THE 6TH JOINT MEETING OF THE EUROPEAN SOFTWARE ENGINEERING CONFERENCE AND THE ACM SIGSOFT SYMPOSIUM ON THE FOUNDATIONS OF SOFTWARE ENGINEERING. *Proceedings...*. Dubrovnik, Croatia: ACM, 2007. p. 459–468. ISBN 978-1-59593-811-4.

SHARPLES, M. The design of personal mobile technologies for lifelong learning. *Computers & Education*, v. 34, n. 3-4, p. 177–193, abr. 2000. ISSN 0360-1315.

SHARPLES, M.; CORLETT, D.; WESTMANCOTT, O. The design and implementation of a mobile learning resource. *Personal Ubiquitous Comput.*, v. 6, n. 3, p. 220–234, 2002.

SHARPLES, M. et al. Socio-cognitive engineering: A methodology for the design of human-centred technology. *European Journal of Operational Research*, v. 136, n. 2, p. 310–323, jan. 2002. ISSN 0377-2217.

SIEMENS, G. Connectivism: A learning theory for the digital age. *International Journal of Instructional Technology and Distance Learning*, v. 2, n. 1, jan. 2005. ISSN 1550-6908.

SUGAR LABS. *Education*. Disponível em: <http://www.sugarlabs.org/>. Acesso em: 9 jan 2011.

SYVÄNEN, A. et al. Supporting pervasive learning environments: Adaptability and context awareness in mobile learning. In: IEEE INTERNATIONAL WORKSHOP ON WIRELESS AND MOBILE TECHNOLOGIES IN EDUCATION. *Proceedings...*. [S.l.]: IEEE Computer Society, 2005. p. 251–253. ISBN 0-7695-2385-4.

THE OLPC WIKI. *Human Interface Guidelines*. 2007. Disponível em: <http://wiki.laptop.org/go/OLPC_Human_Interface_Guidelines>. Acesso em: 11 jan 2011.

VAUGHAN-NICHOLS, S. J. MeeGo, the new netbook linux, arrives. *PCWorld*, maio 2010. Disponível em: <http://www.pcworld.com/article/197414/meego_the_new_netbook_linux_arrives.html>. Acesso em: 11 ago 2010.

VAVOULA, G.; SHARPLES, M. KLeOS: a personal, mobile, knowledge and learning organisation system. In: IEEE INTERNATIONAL WORKSHOP ON WIRELESS AND MOBILE TECHNOLOGIES IN EDUCATION. *Proceedings...*. [S.l.], 2002. p. 152–156.

VENANCIO, V. et al. Collaborative learning supported by Mini-Robotics kits and low cost laptops. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO. *Anais...*. Fortaleza, Brasil, 2008. ISBN 857 669 207-4.

VENANCIO, V. et al. Relato de avaliação de aceitabilidade de mini kit robótica com interface para computador móvel de baixo custo. In: WORKSHOP PROJETO UM COMPUTADOR POR ALUNO (UCA) – BRASIL: PANORAMA, AVALIAÇÃO E PERSPECTIVAS, SBIE. *Anais...*. Fortaleza, Brasil, 2008. ISBN 857 669 207-4.

WARSCHAUER, M.; AMES, M. Can one laptop per child save the world's poor? *Journal of International Affairs*, v. 64, n. 1, p. 33–51, 2010.

WILLIAMS, L.; COCKBURN, A. Agile software development: it's about feedback and change. *Computer*, v. 36, n. 6, p. 39–43, 2003. ISSN 0018-9162.

YEATS, D. *Open-source software development and user-centered design: a study of open-source practices and participants*. 204 f. Tese (Doutorado) — Texas Tech University, Texas, USA, jun. 2006.

APÊNDICE A – PRODUÇÕES BIBLIOGRÁFICAS

As publicações produzidas ao longo desta pesquisa são listadas a seguir.

A.1 Publicações em congressos

MARTINAZZO, A. A. G.; et al. The Mário Schenberg Spaceship: Experiencing Science in a Collaborative Learning VR Environment. In: *Proceedings of the IEEE International Conference on Advanced Learning Technologies*. **Anais...** . p.626-628. doi: 10.1109/ICALT.2009.102, 2009.

MARTINAZZO, A. A. G.; et al. Interdisciplinary Learning Using Low Cost Mobile Platforms: Proposal In Geometry And Arts. In: *Proceedings of the IADIS International Conference Mobile Learning*. **Anais...** . v. 1, p.140-144. Inmaculada Arnedillo Sánchez e Pedro Isaías (eds.). 2008.

MARTINAZZO, A. A. G.; PATRICIO, N.; BIAZON, L.; FICHEMAN, I. K.; LOPES, R. D. Testing the OLPC Drawing Activity: An Usability Report. In: *Proceedings of the IEEE International Conference on Advanced Learning Technologies*. **Anais...** . p.844-846. Santander, Spain. doi: 10.1109/ICALT.2008.200, 2008.

A.2 Capítulo de livro

MARTINAZZO, A. A. G.; LOPES, R. D. Designing Collaborative Authoring Tools for Mobile Learning. In: HIJÓN-NEIRA, R. (Org.); **Advanced Learning**. p.105-114. INTECH. Recuperado de http://sciyo.com/articles/show/title/designing-collaborative-authoring-tools-for-mobile-learning, 2009.

ANEXO A – TRECHO DO EDITAL DE COMPRA PARA O PROJETO UCA

Este anexo contém as seções 7, 8 e 9 do Anexo I (Termo de Referência) do Edital de pregão eletrônico número 59/2007, destinado à compra de 150.000 *laptops* educacionais. O edital foi lançado em 18/12/2007 pelo Ministério da Educação através do Fundo Nacional de Desenvolvimento da Educação (FNDE).

7. ESPECIFICAÇÕES TÉCNICAS GERAIS

7.1. Nenhum componente do equipamento especificado poderá apresentar conexões, fios, jumpers ou outros elementos que indiquem erro ou imprecisão de projeto da parte do fabricante ou do montador/integrador;

7.2. Deverão ser fornecidos e instalados apenas componentes novos, sendo vedado, em quaisquer circunstâncias, o uso de produtos recondicionados, reciclados, enfim, provenientes de reutilização de material já empregado;

7.3. Deverá ser fornecida, pelo licitante classificado em primeiro lugar, em no máximo 48 (quarenta e oito) horas, após ter sido declarado vencedor, 10 (dez) amostras completas do equipamento ofertado, bem como todos os acessórios, cabos de conexão lógica e elétrica, softwares e programas, além de toda a documentação técnica, necessários ao teste de aderência.

• A amostra deverá ser apresentada, após solicitação e comunicação oficial do pregoeiro, em no máximo 48 (quarenta e oito) horas;

• As amostras deverão ser acompanhadas de toda a documentação técnica necessária para a operação, instalação e configuração do respectivo equipamento;

• As amostras ficarão em poder da Contratante até o final da vigência do Contrato, ou o final do prazo de garantia, o que terminar por último, e serão utilizadas, como referência, nas averiguações de campo que vierem a ser executadas pela equipe gestora do Contrato;

• A Contratante reserva-se o direito de mandar proceder, por laboratórios ou técnicos devidamente qualificados, a seu exclusivo critério, testes das amostras para comprovação das especificações exigidas neste Termo de Referência;

• O prazo máximo para comprovação das exigências editalícias será de 10 (dez) dias úteis;

7.4. Todos os equipamentos entregues deverão ser iguais à amostra fornecida para fins de testes de verificação de aderência;

7.5. A qualquer momento, durante a vigência do Contrato, poderá haver atualização tecnológica dos equipamentos, a pedido da Contratada, sendo, neste caso, obrigatória à apresentação de novo conjunto de amostras, conforme descrito no item 7.3, para aprovação pelos técnicos da SEED/MEC, sem aumento de custos para a Contratante, observando-se, ainda, o seguinte:

a) somente poderá ser realizada após a emissão de documento oficial pela Contratante ou seus prepostos, aceitando a atualização e após ser demonstrando a superioridade tecnológica da nova solução em relação a anterior;

b) A amostra deverá ser encaminhada juntamente com documento técnico justificando a mudança por motivos alheios à vontade da Contratada.

c) A Contratante reserva-se o direito de mandar proceder, por laboratórios ou técnicos devidamente qualificados, a seu exclusivo critério, testes das amostras para comprovação das especificações exigidas neste Termo de Referência;

7.6. A Contratante reserva-se o direito de testar e avaliar, através de visitas à linha de produção/distribuição, os equipamentos e/ ou os conjuntos objeto desta licitação, para verificação pontual de aderência às exigências deste Termo de Referência;

7.7. A Contratante reserva-se ao direito de vistoriar e testar os equipamentos entregues, às suas expensas, sendo tais testes amostrais e podendo serem feitos a qualquer tempo;

7.8. Os equipamentos deverão trabalhar em ambientes com temperatura variável entre 5C (cinco graus Celsius) e 41C (quarenta e um graus Celsius) e umidade relativa do ar

(sem condensação) variável de 20% (vinte por cento) a 80% (oitenta por cento);

7.9. Se houver discrepância, na documentação entregue pela Contratada, entre valores expressos em algarismos e por extenso, prevalecerá o valor grafado por extenso;

7.10. Com a finalidade de facilitar a identificação dos equipamentos nos processos de vistorias e acompanhamento das etapas de execução e pós- execução do Contrato, todos os equipamentos devem ter gravados, na cor verde (padrão bandeira do Brasil) ou outra indicada pela SEED/ MEC, na parte superior dos equipamentos, os seguintes dizeres: SEED/MEC - FNDE/MEC – PROJETO UCA

a) a gravação será mediante processo serigráfico ou equivalente, utilizando-se tinta eletrostática ou qualquer outra tecnologia/solução que evite o desgaste da gravação e aumente sua resistência à remoção por abrasivos e/ou raspagem, não sendo aceita a utilização de etiquetas adesivas;

b) caso a cor verde não proporcione contraste aprovado pela Contratante para a identificação desejada, será definido em conjunto com a Contratante outro padrão de cor que venha a atender esse item;

c) os equipamentos destinados aos testes de aderência (amostras) não precisam possuir a gravação aqui exigida.

7.11. Tendo em vista que os equipamentos poderão ser utilizados por crianças com idades a partir de 6 (seis) anos, torna-se imprescindível que os mesmos sejam submetidos a uma avaliação, por instituição indicada pelo MEC e acreditada pelo INMETRO, quanto a segurança no uso, em particular quanto a saúde dos indivíduos. Essa avaliação deverá necessariamente contemplar proteção contra:

a) choques elétricos,

b) ferimentos causados por partes cortantes, pontiagudas e moveis;

c) queimaduras causadas por partes aquecidas.

7.12. Os equipamentos devem ser entregues com a compatibilidade comprovada com o sistema operacional GNU/Linux, permitindo a configuração dos equipamentos em rede, com compartilhamento de seus periféricos e sistema de arquivos. Essa característica deve ser garantida por meio de declaração do fabricante do equipamento ou documentação técnica / manuais em que conste explicitamente a característica exigida nas especificações técnicas, a ser anexada aos documentos de habilitação.

Declarações que não puderem ser comprovadas durante o teste de aderência estarão sujeitas às penalidades previstas na legislação pertinente;

7.13. Os equipamentos deverão ser entregues com sistema operacional GNU/Linux pré-instalado e configurado;

7.14. Todos os manuais, bem como a documentação técnica dos equipamentos deverá estar em português do Brasil.

8. ESPECIFICAÇÕES DETALHADAS

8.1. Requisitos técnicos do equipamento

8.1.1. Placa-Mãe (Motherboard)

a) Padrão da arquitetura de barramento: PCI de 32 bits ou superior ou equivalente;

8.1.2. Microprocessador

a) Somente serão aceitas soluções baseadas em processadores desenhados para a arquitetura de computadores móveis.

b) O equipamento deverá possuir solução de refrigeração compatível com as características exigidas pelo fabricante do processador;

8.1.3. Memória RAM

a) Memória RAM, com no mínimo 256 MB (duzentos e cinqüenta e seis Megabytes), padrão DDR 333 12.4061(g)436(d)1.746066(n)-8.91262 (o)12.4080221(t)0.874347(e)1.74609(r)2.5 8176(m)-9.0317366(()2 q6o íqd..3.

8.1.8. Teclado

a) Integrado ao gabinete;

b) Em conformidade com a norma ABNT-2;

c) Ter proteção contra derramamento de líquidos.

8.1.9. Dispositivo apontador

a) Integrado ao gabinete do equipamento.

8.1.10. Dispositivo Wireless

a) Controladora de rede sem fio integrada ao equipamento, não sendo aceitos adaptadores externos;

b) Suporte para os padrões 802.11 b/g;

c) O equipamento deve possuir suporte a rede ad-hoc de múltiplos saltos, conhecida como rede em malha (mesh network), na qual cada equipamento (laptop) funcione como um roteador, encaminhando os quadros de outros equipamentos semelhantes até o destino final, que pode ser outro laptop (que não está ao alcance direto do equipamento de origem) ou outro destino qualquer na Internet. Eventuais falhas de rotas devem ser tratadas dinamicamente, permitindo que novas rotas sejam automaticamente encontradas, se existirem;

d) Os laptops devem ser compatíveis com a rede em malha especificada acima e também com os padrões 802.11 b/g, bem como poder exercer as funções de ponto de acesso (AP), integrando os laptops da escola entre si e com a internet, concomitantemente

e) Possuir certificação ANATEL;

f) Deve possuir led, externo, indicativo de operação.

8.1.11. Interface de áudio

a) Áudio integrado com pelo menos 16 bits;

b) Possuir microfone integrado ao gabinete do equipamento;

8.1.12. Câmera de vídeo/fotográfica, em cores

a) Acoplada do gabinete do equipamento;

b) Resolução mínima de 640x480 com 30 (trinta) quadros por segundo;

c) Software, integrado ao sistema operacional, que permita a filmagem e a tiragem fotos;

d) Possuir ajuste de brilho, cores e foco, automáticos;

8.1.13. Fonte de alimentação e carregador de bateria

a) Adaptador externo para corrente alternada;

b) Tensão de entrada de 100 à 240V (60 Hz) com tolerância de +- 10%, com comutação automática;

c) Atender a norma UL60950;

8.1.14. Bateria

a) Integrada ao gabinete do equipamento;

b) Bateria de Lithium-Ion, Ni-MH ou LiFeP;

c) Substituível, pelo usuário, sem perda da garantia;

d) Autonomia mínima: 3 (três) horas com o equipamento ligado e a tela de LCD ativa;

e) Atender a norma UL2054;

f) Tempo de carregamento: máximo de 3,5 (três vírgula cinco) horas;

8.1.15. Gabinete

a) Material ou revestimento externo do gabinete anti-deslizante;

b) O gabinete não poderá apresentar saliências, pontas ou estruturas externas perfurantes ou cortantes;

c) Resistência a impactos dinâmicos a uma altura de pelo menos 1 (um) metro em piso rígido (tipo cerâmico);

d) Possuir indicadores visuais de: carga de bateria, rede sem-fio e de equipamento ligado/desligado;

e) Deve possuir teclas para controle de luminosidade do monitor;

8.1.16. O equipamento deverá possuir alça para transporte pelo usuário, integrada ao próprio equipamento ou com a utilização de acessório externo;

8.1.17. Devem ser fornecidos todos os cabos e adaptadores necessários ao funcionamento dos equipamentos, além de mídias com todos os softwares e drivers, dos dispositivos do equipamento;

8.1.18. Peso do equipamento: máximo de 1,5 kg com a bateria instalada;

8.1.19. Consumo máximo de energia: 15 watts

8.1.20. Sistema de segurança

a) Solução de segurança, por hardware, que permita o bloqueio do equipamento caso o mesmo seja extraviado ou permaneça fora da rede lógica da unidade escolar por um tempo determinado, configurável;

b) A solução deverá contemplar, ainda, o serviço de gerenciamento, que permanecerá instalado no servidor da escola;

c) A solução deverá possuir mecanismos que permitam, exclusivamente, a autenticação no servidor da escola;

d) As informações trafegadas entre os equipamentos (laptops) e o servidor da escola deverão ser criptografadas;

e) Os equipamentos deverão ser entregues com o sistema de segurança ativado e bloqueados;

f) A solução deverá estar integrada ao software do servidor descrito no item 8.2.3, deste termo de referência.

8.2. Requisitos Funcionais do equipamento

8.2.1. Sistema operacional:

a) Baseado em software livre e de código aberto;

b) Idioma português do Brasil;

c) Possuir interface gráfica e amigável;

d) Deve permitir a utilização de todas as funcionalidades de hardware do equipamento;

e) Permitir, de forma amigável, a utilização de dispositivos externos, tais como pendrive e câmeras fotográficas;

f) Prover interface gráfica para configuração das funcionalidades da rede sem-fio descrita no subitem 8.1.10 deste Termo de Referência;

8.2.2. Recurso de segurança e interação do equipamento, com o servidor da escola descrito no subitem 8.2.3, abaixo:

a) Possuir recurso de software que permita a interação com o servidor da escola. Este recurso deverá prover, no mínimo, as seguintes funcionalidades:

• sincronização, de forma automática e transparente, dos dados do usuário contidos no equipamento;

• autenticação do usuário;

• autenticação do equipamento;

b) A Contratada deverá fornecer todos os softwares necessários, tanto na parte cliente como na do servidor, para a implementação e gerenciamento da solução;

c) O recurso de interação deverá ser totalmente compatível e trabalhar de forma integrada com o software de gerenciamento da interação instalado no servidor conforme descrito no subitem 8.2.3, abaixo;

8.2.3. Software para o servidor

a) Deverá possuir pelos menos as seguintes funcionalidades:

• Estar em português do Brasil;

• Possuir interface gráfica de gerenciamento;

• Cadastramento e gerenciamento de usuários e grupos e perfis;

• filtro de conteúdo de páginas da internet que permita o bloqueio de conteúdos acessados pelos laptop;

• controle da sincronização com agendamento;

• autenticação do usuário do laptop;

• autenticação do equipamento;

• bloqueio, individualizado, dos laptop;

• backup e recuperação das informações contidas no servidor;

• permitir a visualização e impressão de relatórios gerenciais com pelo menos as seguintes informações: cadastro dos usuários,

• permitir a visualização e impressão de relatórios de monitoramento com no mínimo as seguintes informações: estatística de tráfego, sítios mais acessados, estatística de utilização por aluno;

b) A interface de configuração do software deverá permitir preferencialmente acesso via Web;

c) A Contratada deverá fornecer, instalar e configurar todos os softwares necessários, tanto na parte cliente como na do servidor, para a implantação e gerenciamento da solução;

d) A contratada deverá disponibilizar software que permita a gestão das informações, garantindo assim a integralidade dos dados coletados, bem como a continuidade do serviço.

As informações coletadas deverão ser armazenadas em banco de dados, o qual poderá ser visualizado por meio de interface web e organizado em nível de escola;

e) O hardware a ser utilizado na solução do servidor será fornecido pela SEED/MEC e a Contratada deverá propiciar a devida compatibilidade dos softwares ofertados com o mesmo.

8.2.4. Software (aplicativos) instalados:

a) Baseado em software livre e de código aberto;

b) Idioma português do Brasil;

c) Possuir interface gráfica e amigável;

d) Deve possuir aplicações para:

• Processamento de textos com suporte ao formato ODT e com recursos mínimos para: negrito, itálico, utilização de imagens gráficas no texto, alteração do tipo e do tamanho da fonte, trabalhar com tabelas;

• Planilha eletrônica;

• Edição e visualização de imagens;

• Navegação web que permita o acesso a sítios que utilizem plugins Java e Flash, além da reprodução áudio e vídeo em tempo real. O navegador deverá possuir total compatibilidade com os citados plugins;

• Chat;

• Logo;

• Squeak

• Jogos educacionais (xadrez, palavras cruzadas, etc);

• Exibição de vídeos;

• Reprodução de arquivos de sons pelo menos no formato ogg;

• Gravação de sons;

• Leitura de arquivos PDF.

8.3. Requisitos de garantia

a) O prazo de garantia contra defeitos de fabricação, tanto do hardware quanto do software, deverá ser de, no mínimo, 36 (trinta e seis) meses, abrangendo todo o território brasileiro;

b) A Contratada deverá possuir estrutura que garanta a manutenção corretiva, a reposição de peças e o suporte técnico, para o funcionamento dos equipamentos, em termos de hardware e software, durante o período de garantia;

c) Caberá à Contratada o ônus e a responsabilidade pela logística de retirada e devolução dos equipamentos à unidade escolar, em caso de cumprimento de garantia ou reposição;

d) Deverá ser fornecida, pela Contratada, garantia à obtenção de atualizações e correções de software em função de problemas descobertos após a entrega dos equipamentos. Estas atualizações deverão ser mantidas em um sítio na Internet pelo menos durante a vigência da garantia do equipamento. Este sítio deverá estar em português do Brasil;

e) A contratada deverá prover atualizações tecnológicas do Sistema Operacional e demais softwares utilizados pelo período de 36 (trinta e seis) meses, contados a partir da instalação dos equipamentos;

f) Os serviços de garantia de atualização tecnológica devem abranger:

• o fornecimento de novas versões do Sistema Operacional e dos demais softwares utilizados, de forma a disponibilizar evoluções tecnológicas implementadas, quando se fizer necessário;

• implementação de garantia do sistema operacional e demais softwares utilizados, para a correção de possíveis falhas, decorrentes de erros ou problemas na sua implementação, de forma a propiciar o seu perfeito funcionamento;

• mecanismos que permitam, de forma simples e amigável, a atualização do sistema operacional e dos softwares utilizados.

9. SOBRE A GARANTIA DE FUNCIONAMENTO

9.1. Apresentação de garantia contra defeitos de fabricação de no mínimo 36 (trinta e seis) meses, contados a partir da data da entrega constante do Termo de Recebimento

(ENCARTE B) deste Termo de Referência;

9.2. Entende-se por garantia, o período em que a Contratada compromete-se a manter os equipamentos em funcionamento, considerando, ainda, os seguintes aspectos e condições:

1) Prazo de Garantia de Funcionamento é o período em meses, dentro do qual, nas condições registradas na Proposta Técnica e constantes do respectivo Termo de Garantia, a Contratada compromete-se em manter os equipamentos, por ela fornecidos, em perfeito funcionamento, configurados da forma especificada neste Termo de Referência, e a fornecer mídias eletrônicas necessárias ao restabelecimento do funcionamento, nas condições e configurações constantes deste Termo de Referência, no local de sua instalação. A responsabilidade sobre a solução de "recuperação" da imagem inicial dos equipamentos fornecidos (licença do utilitário de reinstalação da imagem do equipamento), é da Contratada;

2) Dependendo da dimensão ou gravidade do dano, identificada no acionamento da garantia, a Contratada deverá fazer a substituição do equipamento enquanto providencia a solução do problema em suas próprias instalações ou em um dos seus agentes credenciados e autorizados;

3) Entende-se por perfeito funcionamento quando, após o atendimento em garantia, os equipamentos estiverem operacionais conforme exigido por este Termo de Referência e as demais funcionalidades idênticas às da imagem instalada em fábrica;

4) Os agentes credenciados ou autorizados pela Contratada para prestação dos serviços de garantia, devem atuar na própria unidade federada em que os equipamentos estiverem instalados;

5) O prazo máximo para atendimento do chamado de garantia não poderá ser superior a 5 (cinco) dias e começará a ser contado a partir da abertura do chamado. Este atendimento da garantia não poderá ultrapassar 10 (dez) dias a partir do chamado. Caso este período seja ultrapassado deverá ocorrer a substituição imediata do equipamento para cumprimento da garantia;

6) Declaração do licitante, em que conste o endereço da página Internet de garantia aos equipamentos, declarando explicitamente que a página possibilita cópia e instalação dos drivers de dispositivos mais recentes, bem como possuir informações da garantia do produto;

7) A Contratada deverá prover estrutura de Central de Atendimento, gratuita, por meio

de linha telefônica 0800, para o acionamento da garantia, devendo atender aos seguintes requisitos:

1) Funcionar em dias úteis, das 8 às 18 horas;

2) Estar em funcionamento a partir da data da homologação da licitação e assim permanecer até o término da garantia dos equipamentos;

3) Comprovar está solicitação por meio de declaração a ser entregue junto com a proposta comercial.

8) No caso de substituição de equipamento defeituoso, a Contratada deverá seguir os seguintes critérios:

1) fazê-lo por modelo igual ou superior ao ofertado;

2) em havendo substituição por solução superior a Contratada deverá se responsabilizar pela integração dos equipamentos, no que se refere ao hardware e software, com os entregues anteriormente;

3) o ônus pela substituição caberá à Contratada.