Gestão de Projetos em Software Livre

(Título Provisório)

Cesar Brod

com a colaboração de Joice Käfer

Agradecimentos

Obrigado à minha mulher Meire e minhas filhas Natália, Aline e Ana Luiza por estarem sempre ao lado deste nerd.

Obrigado a meus pais por me apontarem o caminho que levou-me a fazer sempre o que amo.

Obrigado ao Rubens Queiroz de Almeida, por ter plantado a semente deste livro.

Obrigado ao meu editor Rubens Prates, por sua paciência de Jó em esperar por este livro.

Obrigado a inúmeras pessoas da Solis, da Univates, da UFRGS e da Unicamp que fazem parte de muitos capítulos deste livro.

Obrigado especial à minha mais do que melhor amiga, sócia e crítica, Joice Käfer, pelo incentivo, apoio, mimos e broncas na dosagem sempre certa.

Apresentação

Índice

Prefácio

Os e-mails trocados acerca deste livro, com meus amigos e meu editor, datam de meados de 2003. Na época, através de minha empresa, a BrodTec, eu atuava como gestor de TI no Centro Universitário Univates, em Lajeado, onde desde 1999 desenvolvíamos e integrávamos soluções utilizando tecnologias livres. Em 2000 criamos o primeiro repositório de software livre brasileiro, o CodigoLivre.Org.Br e nele colocávamos a nossa própria produção: softwares para a gestão acadêmica, para bibliotecas, para a gestão de conteúdo web e também um framework de desenvolvimento que incorporava muito do que havíamos amadurecido na direção de uma metodologia orgânica de desenvolvimento.

Mas aliada à evolução de uma metodologia de desenvolvimento que honrava a colaboração e o compartilhamento de informações através do uso de softwares livres, em conjunto com a reitoria da Univates, em especial com o Professor Eloni José Salvi, concebi um modelo de negócios que deu origem à primeira cooperativa de desenvolvimento e integração de soluções em software livre de que se tem notícia: a Solis, fundada em janeiro de 2003. Foi por aí que o Rubens Queiroz de Almeida, mantenedor do portal Dicas-L (www.dicas-l.com.br) começou a dizer que eu devia contar toda esta história em um livro e o outro Rubens, o Prates, editor da Novatec, comprou a ideia.

Meu senso crítico, porém, nunca permitiu que eu passasse da página 30 e do esboço inicial de 10 capítulos. Eu viajo bastante, especialmente palestrando em vários lugares do Brasil e do mundo sobre o trabalho que venho desenvolvendo com softwares livres. A cada nova palestra, a cada conversa com gestores, professores, alunos, equipes de desenvolvimento eu sentia que o livro que eu escrevia ainda carecia de uma fundamentação maior. Ao mesmo tempo, eu queria que, antes de mais nada, ele mantivesse o formato de um relato de experiências, um conjunto de pequenas histórias que, colocadas na devida sequência, auxiliassem o leitor em sua aprendizagem nas práticas e métodos para a gestão de projetos que me foram úteis.

Sem conseguir finalizar o livro, fui colocando em minhas palestras e nos artigos que eu produzo para o Dicas-L muito daquilo que fazia parte das eternas 30 páginas. Elas foram revistas, ampliadas, novos assuntos foram abordados de tal forma que, em determinado ponto, eu achei que o livro era desnecessário, já que todo o conteúdo estava na web e eu podia seguir acrescentando novos artigos a meu bel prazer. Alguns fatores fizeram-me mudar de ideia.

O primeiro é que acabei escrevendo uma parte do livro *Free as in Education*[1], publicado pela OneWorld.Net em 2004 com o patrocínio do Ministério de Apoio ao Desenvolvimento da Finlândia. Este livro aborda a importância de softwares de código livre e aberto na geração de emprego e renda, no desenvolvimento de países emergentes e na potencial solução de conflitos. Coube a mim a análise da utilização de softwares livres e de código aberto na América Latina. Nico Coetzee fez o estudo relativo à África e Frederick Noronha à Ásia não industrializada. Niranjan Rajani foi o coordenador do trabalho.

No final de 2007 eu e minha sócia, Joice Käfer, trabalhamos na tradução para o português do livro *Redes sem Fio no Mundo em Desenvolvimento*[2], publicado no início de 2008 e que teve mais de dois milhões de downloads entre 2008 e 2009. Entre 2009 e 2010 traduzi, com a revisão técnica da Joice, dois livros de Fred Brooks Jr.: *O Mítico Homem-Mês* e *O Projeto do Projeto – do modelo à implementação*, ambos publicados no Brasil pela editora Campus Elsevier. A leitura que fizemos durante a tradução reforçou nossa percepção óbvia de que é mais agradável, ainda hoje, ter um livro nas mãos, na bagagem, na cabeceira da cama, do que ficar com o computador ligado o tempo todo,

1 O livro completo pode ser obtido em http://miud.in/cRs. A parte relativa à América Latina, escrita por Cesar Brod, pode ser obtida em http://brodtec.com/helsinki.pdf

2 Disponível em http://wndw.net/

lendo informações na tela – isto talvez mude em um futuro próximo, quando os tablets realmente tornem-se populares. Além disto, estas três traduções, em conjunto, representam mais de mil páginas digitadas e revisadas, um esforço que poderia ser também dedicado para a elaboração de meu livro.

O último fator é que meus artigos relativos à gestão e metodologia de desenvolvimento são bastante lidos e referenciados na web, o que parece indicar que há um público para este assunto.

De 2006 a 2009 também trabalhamos com a gestão de projetos de desenvolvimento em código aberto e interoperabilidade junto à Universidade Federal do Rio Grande do Sul (UFRGS) e à Universidade Estadual de Campinas (Unicamp), através de um contrato com a Microsoft, o que ampliou ainda mais a nossa experiência com a gestão de equipes e nos deu a oportunidade de exercitar, ainda mais, metodologias ágeis de desenvolvimento, em especial o Scrum.

A participação da BrodTec na coordenação do temário nas edições de 2009 e 2010 da Latinoware, implicando em mais conversas com mais pessoas, e a consolidação de parte do conhecimento que eu havia adquirido de forma prática e empírica na disciplina de mestrado em Engenharia de Software, ministrada pelo Professor Marcelo Pimenta na UFRGS, para a qual fui convencido pela Joice a assistir como aluno ouvinte (ela assistiu como aluna especial) foram fundamentais para que eu me convencesse que, finalmente, eu poderia produzir um material digno de meus potenciais leitores.

Como ler esse livro

Tentei colocar os capítulos deste livro em uma ordem mais ou menos cronológica, buscando seguir o roteiro de minha própria aprendizagem e enfatizando o aprimoramento “orgânico” de uma metodologia de gestão e desenvolvimento que usa uma série de elementos dos quais me apropriei pelo caminho. Ao longo do livro, procurei introduzir uma série de métodos, práticas e conceitos sem, de forma alguma, competir com o vasto material já existente sobre muitos assuntos sobre os quais eu trato aqui. Por outro lado, busquei não obrigar o leitor a correr atrás de referências externas a todo o momento. Quando necessário, uso notas de rodapé com links que permitem um aprofundamento maior em um ou outro tema. A Bibliografia, ao final do livro, constitui o meu próprio conjunto básico de referência e aprendizagem.

Mesmo optando pela ordem cronológica, o leitor deve sentir-se à vontade para usar capítulos específicos como material de pesquisa ou referência, especialmente aqueles mais conceituais (os que introduzem, por exemplo, UML, Extreme Programming e Scrum, por exemplo). Aqueles mais interessados em modelos de negócios podem focar sua leitura na parte mais “histórica” da evolução de formas de criação de produtos, geração de emprego e renda com software livre, ideias sobre contratos com clientes e outros assuntos relacionados, deixando completamente de lado os aspectos mais técnicos.

Vendendo a idéia

Luciano, Carlos e Maurício,

Pensei bastante sobre a metodologia de implementação do novo Sistema Administrativo. Sei que nesta fase de estudo de uma nova ferramenta ainda podem surgir polêmicas sobre o método de migração e a própria ferramenta empregada, o que é totalmente natural. Por isto mesmo, peço a vocês um voto de confiança na adoção das regras de desenvolvimento deste novo sistema e na linha mestra que adotaremos na implementação do mesmo.

Aqui segue um primeiro "draft" para o desenvolvimento do novo Sistema Administrativo, que vocês devem transformar em um documento final. Sou flexível em vários aspectos, à exceção das datas chaves do cronograma que devem ser mantidas a todo o custo.

Projeto Interno

Novo Sistema Administrativo

Nome-código: UniverSis

Coordenador: Prof. Luciano Brandão

Objetivo: Migração do Sistema Administrativo existente na Univates para uma estrutura baseada em Intranet, com bases de dados padrão SQL e acesso dos clientes através de browsers padrão de mercado. O novo UniverSis utilizará ferramentas de sistemas abertos e será publicado para a comunidade acadêmica que poderá participar do projeto.

Pré-requisitos:

<NOTA>

Sei que isto é uma imposição, certamente passível de questionamento. Aqui é onde peço, justamente, o maior voto de confiança.

</NOTA>

1. A interface do usuário será exclusivamente através de browser padrão (Netscape, Internet Explorer ou outro)

2. A base de dados para o desenvolvimento inicial será o MySQL

<NOTA>

A migração para o Oracle ou outra base SQL deve ser tranquila, mas devemos começar com uma base na qual consigamos suporte da comunidade "OpenSource" com facilidade.

</NOTA>

3. A base de dados de testes deve refletir a modelagem da atual, mas deve ser INDEPENDENTE da existente.

<NOTA>

Luciano, sei que tens críticas quanto a isto. Podemos discutir os malefícios e benefícios dos métodos infinitamente. Quero evitar neste momento qualquer entrave no desenvolvimento que possa ter como raiz a migração dos dados e os testes de volume. Confio que no meio do processo teremos um bom plano de migração de bases de dados.

</NOTA>

4. Todas as ferramentas utilizadas serão OpenSource, ou em último caso OpenSource para o ambiente acadêmico.

Ferramentas adotadas

SO Servidor: Linux

SO Cliente: Não importa, desde que use browsers padrão

Base de dados: MySQL – inicialmente, com possibilidade de porte para qualquer outra base padrão SQL

Linguagens: PHP (preferencialmente) e qualquer acessório OpenSource necessário.

Cronograma:

20.11.99 – Documentação do projeto completa

01.12.99 – Publicação do projeto na Internet e contato com as principais Universidades do país

01.12.99 – 01.03.00 – Definição de atribuições de colaboradores externos

```
- a cada mês, teste de módulos principais -


01.05.00- Laboratório de funcionalidades e início da migração
das bases de dados

01.07.00 à 30.07.00 - Migração total das bases e testes de
stress com situações reais em paralelo com o sistema original

01.07.00 à 30.07.00 - Ajustes e correções

01.08.00 - UniverSis em produção, com backups diários e
possibilidade de retorno ao sistema tradicional
```

O texto acima é a transcrição de uma mensagem de correio eletrônico enviada à Maurício de Castro (hoje sócio-proprietário da empresa Sol7) e outros membros da equipe de desenvolvimento em 20 de outubro de 1999. Este é o primeiro registro do Sagu, na época chamado de UniverSis.

O Sagu (Sistema Aberto de Gestão Unificada) foi o primeiro projeto em software livre desenvolvido na Univates. O cronograma acima foi cumprido à risca, com um grupo que se manteve em três desenvolvedores até a entrada do sistema em produção, em agosto de 2000. Esta equipe foi crescendo até que em 2003, com 20 pessoas, formou, com o incentivo da Univates, a Solis, Cooperativa de Soluções Livres. A Solis hoje conta com mais de 50 associados que vivem exclusivamente de serviços em software livre e presta serviços para clientes espalhados por todo o Brasil.

É fundamental para que *qualquer projeto* seja bem sucedido que:

1. a equipe esteja engajada, comprometida com ele;
2. a equipe tenha fé e respeito em seu líder;
3. que toda a estrutura organizacional superior o apoie.

Isto é verdade para qualquer projeto, mas quando falamos de software livre, o desafio ainda é maior e mais interessante. Em 1999 o conceito de software livre ainda não era tão difundido como é hoje (mesmo na mensagem acima, eu usava o termo OpenSource ao invés de Free Software ou Software Livre). Quando desenvolvemos um *projeto em software livre*, ele será bem sucedido se:

1. a equipe está engajada, comprometida com ele, entende a filosofia do software livre e com isto está disposta a trabalhar de forma colaborativa, mesmo com pessoas externas à equipe;
2. a equipe tenha fé e respeito em seu líder, que deve entender que está coordenando um projeto que pode ter colaboração que está além de sua gestão;
3. que toda a estrutura organizacional superior o apoie e saiba que não é “dona” do que está sendo desenvolvido.

Na Univates não foi muito difícil de se conseguir isto, especialmente porque em uma instituição de ensino privada o negócio já é a produção e a “venda” do conhecimento, sem que ela seja “dona” do mesmo. Além disto, o pró-Reitor Administrativo/Financeiro da instituição na época, o Professor e Mestre em Economia Eloni José Salvi, já tinha uma boa experiência na lida tecnológica e a mente bastante aberta à novas propostas e tecnologias. Ele mesmo encarregou-se de conseguir o apoio de seus colegas na reitoria. Mais adiante, tivemos algumas vezes que “provar” que a nossa escolha pelo software livre foi a correta não só filosoficamente, mas também técnica e economicamente. No capítulo [incluir aqui] retornaremos a este aspecto. Ainda assim, repetidamente tínhamos que

esclarecer às pessoas com as quais trabalhávamos ou a quem nos reportávamos o que é, afinal, software livre, especialmente evitando confusões sobre “livre” e “grátis”, confusão que, infelizmente, importamos graças ao dúbio sentido da palavra *free* em inglês. A confusão que persiste, ainda hoje, é parcialmente devida ao forte *lobby* daqueles a quem interessa perpetuar *ad infinitum* o moribundo modelo do software proprietário e em parcela ainda maior por culpa nossa, os que trabalham com software livre, já que não fomos ainda capazes de esclarecê-la suficientemente.

O que é software livre?

Software livre não é um conceito novo. Na verdade, ele é mais antigo que o conceito de software proprietário, este definido como um programa ou conjunto de programas de computadores que são de propriedade de uma pessoa ou empresa, protegidos por mecanismos conhecidos como "patentes", "direitos autorais", "marcas registradas" ou outros.

Para poder utilizar um software proprietário você deve aceitar a "licença de uso" que o acompanha. Esta licença normalmente limita a responsabilidade da empresa que fornece o software e estabelece seus deveres como usuário do produto. Tipicamente, estas licenças garantem o direito de uso, mas não de posse do programa de computador (num arranjo similar ao aluguel de um bem material qualquer) e proíbem a cópia do programa e sua modificação. A Microsoft[3], uma grande empresa que desenvolve e comercializa softwares proprietários orientava seus distribuidores a comparar um programa de computador com um apartamento:

> *"Quando você adquire um software de sistema operacional Windows, obtém na verdade uma licença para utilizar esse software e não direitos totais de posse sobre ele. Existem limites sobre o que você pode fazer com o software de sistema operacional e esses limites são explicados detalhadamente na licença. Na verdade, esse tipo de acordo já é algo bem comum nos dias de hoje: você precisa de diversos acordos para utilizar coisas o tempo todo, mesmo que não tenha direitos plenos para fazer o que quiser com elas. Por exemplo: Você aluga um apartamento, mas não pode realizar alterações estruturais, ou até mesmo estéticas, sem o consentimento do proprietário."*[4]

Programas de computadores, porém, não são "entidades físicas" como um apartamento e não deveriam ser tratados como tal. Um programa de computador é um conjunto de instruções de base lógica e matemática que explica a uma máquina o que ela deve fazer. O programa abaixo, por exemplo, escrito na linguagem de programação Python, ilustra como um computador pode conjugar verbos regulares em português:

```
# Programando em Python
# Exemplo 3 - Cesar Brod
#
# modulo_conjuga.py
#
# Módulo que conjuga verbos regulares em Português
# versão 0.0 07/07/2001 20h17
#
def conjuga(verbo):
    "Conjuga o tempo presente de verbos regulares"
    conjugado=[]
    conjuga_ar = ['o','as','a','amos','ais','am']
    conjuga_er = ['o','es','e','emos','eis','em']
    conjuga_ir = ['o','es','e','imos','is','em']
    termina_em = verbo[-2:]
    if termina_em == 'ar':
        for terminacao in conjuga_ar:
            conjugado.append(verbo[:-2]+terminacao)
    elif termina_em == 'er':
        for terminacao in conjuga_er:
            conjugado.append(verbo[:-2]+terminacao)
```

3 http://www.microsoft.com

4 http://windowslicensing.msoem.com/portuguese/3mod_operating.asp#use (este link não está mais ativo, mas seu conteúdo está armazenado em http://web.archive.org/web/20020906180605/http://windowslicensing.msoem.com/portuguese/3mod_operating.asp)

```
        elif termina_em == 'ir':
            for terminacao in conjuga_ir:
                conjugado.append(verbo[:-2]+terminacao)
        else:
            print 'Tem certeza que '+verbo+' é um verbo regular?'
        return conjugado
```

As linhas precedidas de "#" são comentários, servem para documentar o programa - o computador não faz nada com elas. As linhas seguintes "ensinam" ao computador que ele deve verificar a terminação do verbo em "ar", "er" ou "ir" e substituir esta terminação pela da respectiva conjugação. Assim, o verbo "amar" é conjugado da seguinte forma:

```
      Verbo "amar"

      Na linha:

      if termina_em == 'ar':

      O programa passa a saber que as terminações para o verbo "amar" são:

      conjuga_ar = ['o','as','a','amos','ais','am']

      Em:

      for terminacao in conjuga_ar:
                  conjugado.append(verbo[:-2]+terminacao)

      O que o programa faz é pegar o verbo "amar" sem as duas últimas
letras - verbo[:-2] - e acrescentar a terminação que ele obtém da lista
conjuga_ar.

      O resultado é:

      amo, amas, ama, amamos, amais, amam
```

O conhecimento que utilizei para criar este programa é encontrado em muitos livros de gramática e ensinado por nossos professores nos primeiros anos de escola. De forma similar, mesmo os programas mais complexos são construídos a partir de uma base que é o conjunto do conhecimento humano. A partir de qual momento, então, torna-se justo tornar proprietário um programa que usa em sua construção conhecimento que é de domínio público? Eu poderia "patentear" o meu simples "conjugador de verbos", vendê-lo a alguma escola e impedir que o mesmo seja modificado e copiado? Certamente que sim! Muitas empresas fazem isto. Mas isto é moralmente aceitável?

Entre meados dos anos 50 - quando computadores começaram a ser usados comercialmente - até o final dos anos 60, programas eram livremente trocados entre empresas e programadores com o incentivo dos fabricantes de computadores. As máquinas eram caríssimas e quanto mais programas estivessem livremente disponíveis, mais "valor agregado" percebiam as empresas que eram capazes de comprar os computadores[5]. Com a popularização dos computadores nas empresas e a queda de seu preço, o software passou a se tornar também um negócio e programas que eram livres passaram a ser comercializados. No meio acadêmico, programas ainda eram livremente desenvolvidos e compartilhados até o final dos anos 70, quando novas máquinas (mini e microcomputadores) com

5 An early history of software engineering by Robert L. Glass - http://www.cs.colorado.edu/~kena/classes/5828/s99/comments/srinivasan/01-29-1999.html

sistemas operacionais proprietários passaram a se tornar populares. Os sistemas operacionais proprietários acabavam por prender seus usuários a um conjunto de ferramentas de produtividade e desenvolvimento que tinham que ser obtidas dos próprios fabricantes dos computadores ou de "software-houses" independentes. Em resumo, o software passou a ser vendido quando a margem de lucro no "hardware" (os computadores) diminuiu. A partir do início dos anos 80, tornou-se comum a venda de computadores com software, cujo preço estava embutido no valor do equipamento. Um usuário leigo, mesmo nos dias de hoje, compra um computador sem saber que uma grande parcela do que paga não é relativa ao equipamento, mas sim aos programas que o acompanham. Como o mercado de informática é muito novo, especialmente para o usuário doméstico e pequenas empresas que apenas passaram a ter acesso aos computadores na última década, a lógica de mercado que se impôs predominantemente é a de que é natural "alugar" um software como se aluga um apartamento.

No início dos anos 80, um programador do laboratório de inteligência artificial do MIT (Massachussets Institute of Technology), Richard Stallman, iniciou um movimento que busca conscientizar as pessoas de que programas de computadores são uma manifestação do conhecimento humano e, como tal, não podem ser propriedade de ninguém. Em 1984 Richard criou a Free Software Foundation e a licença de uso de programas de computador GPL - General Public License. Um programa distribuído sob esta licença pode ser copiado, modificado e redistribuído. Muitos programadores concordaram com as ideias formatadas originalmente por Stallman e hoje existem programas e sistemas bastante complexos desenvolvidos totalmente em software livre. Dentre os mais populares estão o sistema operacional Gnu/Linux e o conjunto de aplicativos para escritório OpenOffice.Org.

Para que um programa possa ser copiado, basta que isto seja legalmente permitido, uma vez que qualquer pessoa que possui um computador é dona dos meios de produção[6] que permitem que a cópia seja efetuada. O que dá o direito legal, ou não, de que o programa seja reproduzido é sua licença de uso. As licenças de software proprietário limitam, ou efetivamente impedem, que o mesmo seja copiado. A cópia de um software proprietário constitui-se em uma ação ilegal chamada comumente de pirataria, cuja pena inclui multas altíssimas e mesmo a prisão do infrator[7]. A licença de um software livre[8] garante a seu usuário o direito de copiá-lo, assim, não existe pirataria em software livre.

Para que um programa possa ser modificado, além de que isto seja legalmente permitido por sua licença, é necessário que se tenha acesso a seu código-fonte, sua receita. É possível, por exemplo, que meu programa "conjugador de verbos" seja estendido para outros tempos verbais, uma vez que temos acesso à "receita" do programa original. Todo o software livre garante acesso a seu código-fonte e sua licença permite que o mesmo seja, legalmente, modificado, desde que a versão modificada também seja um software livre - isto é o que garante que o conhecimento que levou à produção deste software continue de propriedade coletiva e não de algumas poucas pessoas ou empresas.

Mesmo que você não entenda nada de programação, o acesso ao código-fonte permite que você contrate os serviços de quem entende disto para que modifique um programa em software livre, adequando-o à sua necessidade. O modelo de negócios que sustenta empresas e pessoas que vivem de software livre tem por base a venda de serviços. Claro que este modelo não interessa a empresas que, mesmo com poucos anos de existência, tornaram-se milionárias (e fizeram milionários) com a venda de conhecimento fechado em caixas de conteúdo idêntico que são vendidas milhares de vezes, e, como são estas as empresas que hoje dominam o mercado de informática, é perfeitamente

6 Liberdade em Software: Direitos, Deveres, Metáfora e Ganhando a Vida, Robert J. Chassell - http://www.brod.com.br/handler.php?module=sites&action=view&news=35

7 Lei de Software - http://www.mct.gov.br/legis/leis/9609_98.htm

8 Licença Creative Commons GPL - http://creativecommons.org/licenses/GPL/2.0/

natural uma enorme resistência e investimentos em propaganda contra o software livre.

Eu e o software livre

Pensei em vários títulos para este capítulo, especialmente um que não parecesse pretensioso ou egoísta. Todos soaram como alguém pretensioso ou egoísta procurando um título que não fosse pretensioso ou egoísta. Assim, ficou o título acima, que é secundário perante a história que se inicia a partir do próximo parágrafo, esta sim, uma verdadeira *egotrip*.

Em 1988 eu estudava Física na UFRGS em Porto Alegre, o primeiro curso superior que não conclui (na segunda Universidade, comecei este mesmo curso na USP, em São Paulo) e por diletantismo lia sobre inteligência artificial e sistemas especialistas. Eu possuía um computador MSX, que tinha entre suas várias peculiaridades a capacidade de endereçar 64 Kbytes de memória, 32 Kbytes por vez. Um dos programas-exemplo em um dos livros que eu estava lendo simulava um "psicólogo" conversando com o usuário e tentando, através do reconhecimento de uma lista de palavras-chave, mudar de assunto e fazer perguntas "surpreendentes". O livro era em espanhol e o programa estava escrito em Basic. Decidi tornar o programa mais interessante transformando o "psicólogo" no "Analista de Bagé", simulando uma conexão com o consultório do mesmo e trabalhando a lista de palavras-chave de forma a que, quando usadas, trocavam o "cenário" da consulta, invariavelmente levando o consulente a falar da relação que tinha com sua mãe. O diagnóstico da "loucura" ou "faceirice" do "bagual" era sempre "um causo de Édipo". Na época, eu prestava serviços para o Banco do Brasil, em Porto Alegre, que havia recentemente adquirido microcomputadores cujo formato de gravação em disquete era o mesmo que o meu MSX. Assim, reescrevi o programa para que o mesmo código pudesse ser executado em MSX-Basic (minha máquina) e Mbasic (as máquinas do Banco do Brasil). Isto garantiu-me uma quantidade de usuários que, nas horas vagas (ao menos acho que eram horas vagas), podiam brincar com o meu programa e contribuir com sugestões. O programa passou a armazenar a conversa dos usuários e apontar palavras mais usadas, que passaram a ser usadas como palavras-chave em novas versões. Nas últimas versões, era capaz de armazenar declarações do consulente que continham algumas palavras-chave, lembrar delas e fazer perguntas com base nas mesmas. Por exemplo: se o consulente digitasse "Não convivo bem com minha família", a palavra "família" era uma das palavras-chave e o verbo "conviver" estava na lista de conjugações. Assim, mais adiante o "analista" poderia perguntar algo do tipo "Antes disseste que não convives bem com tua família, explica melhor...". Esta utilização de elementos que eram inseridos pelo próprio consulente, junto a cenários que poderiam vir à tona aleatoriamente ou através do reconhecimento de palavras-chave tornou o sistema bastante interessante, um brinquedo agradável. Como me apropriei indevidamente do "Analista de Bagé", eu distribuía o programa em disquete, com instruções, código-fonte e recomendações para que o usuário comprasse a obra do escritot Luís Fernando Veríssimo, na esperança de nunca ser processado pelo mesmo.

O tempo passou, perdi contato com as pessoas que trabalhavam comigo na época, e, infelizmente, a última cópia que eu tinha do programa foi doada por descuido junto com o meu MSX em um disquete que foi formatado e aproveitado para alguma outra coisa. Há algum tempo, tentei fazer uma "busca do software livre perdido". Fiquei sabendo que alguém colocou o meu programa para rodar em uma implementação de BASIC para o ambiente /370 da IBM, por volta de 1992 e que o mesmo podia ser acessado através de terminais 3270, mas nunca mais tive acesso ao programa, que um dia espero reescrever, talvez em Python.

Desde que concluí o curso técnico em eletrônica, em 1982, passei a trabalhar na área de suporte técnico, onde era bastante comum entre os colegas a troca de informações e pequenos programas que nos ajudavam na solução de problemas, ainda que alguns poucos colegas preferissem "esconder o ouro". Quando escrevi o "Analista de Bagé - o programa" jamais cheguei a pensar em ganhar dinheiro com ele, uma vez que, a rigor, nenhuma das ideias que usei nele eram novas, mesmo que eu as houvesse combinado de maneira criativa. O fato de eu ter distribuído o "Analista" como um

software livre foi mais uma questão circunstancial, ainda que reflexo da forma como eu já pensava, do que algum compromisso filosófico que acabei por abraçar mais tarde.

Meu primeiro contato com o software livre, conforme definido e protegido pela Free Software Foundation, também foi casual (não que eu acredite em casualidade ou destino, mas isto também já é outra história): no início de 1993 eu estava trabalhando na Tandem Computers (hoje a divisão de produtos NonStop da HP), que havia vendido um sistema Unix à Telemig, em Minas Gerais. Eu havia pedido a meu chefe que adquirisse uma licença do SCO-Unix para que eu rodasse em meu laptop (precursor do notebook) e assim pudesse simular situações e problemas que tínhamos na instalação em Minas. O custo do SCO-Unix, porém, fez com que meu chefe negasse o pedido e que eu fosse atrás de outras alternativas. Havíamos, recentemente, instalado um link de 64 Kbps com nossa matriz, na Califórnia e eu já havia aprendido com a equipe técnica da empresa como usar este link para ter acesso à Internet (ainda uma pequena parcela do que é hoje) e aos grupos de notícias (USENET). Usando ferramentas como o Gopher e o Archie (precursores dos mecanismos de busca atuais) acabei descobrindo informações sobre o Minix e posteriormente sobre o Linux, sistemas Unix que eu poderia instalar em meu laptop sem nenhum custo. Fiz o download do Linux com um conjunto de GNU *tools* de uma Universidade da Califórnia e, seguindo algumas instruções, consegui fazer com que o GNU/Linux rodasse em meu laptop sem maiores traumas, para meu deleite e a total incompreensão da grande maioria dos meus colegas de trabalho, que não entendiam o porque da minha felicidade com aquilo que "mais parecia o DOS". Mas eu sabia! Eu tinha um Unix!

Com o aumento da equipe técnica da Tandem, chegamos a ter um pequeno "grupo de usuários" que em 1995 já rodava o Linux com o Xfree86 (modo gráfico) e fazia uma série de experiências interessantes, especialmente com as máquinas em rede.

O software livre na Univates

Em 1996, com verbas da Rede Tchê (RNP), a Univates[9] conectou-se à Internet e implementou seu provedor de acesso. Fábio Wiebbelling (hoje empresário, proprietário da ITS) foi à Porto Alegre fazer um curso de Linux na UFRGS com a Professora Liane Tarouco[10], depois de já ter tido alguma experiência com o Solaris em uma máquina da Sun Microsystems.

Também em 1997 eu começava a planejar meu retorno para o sul, especialmente para o Vale do Taquari, investindo junto com outros sócios, na criação de um provedor de acesso à Internet[11], que acabou unindo forças com o provedor da Univates. Durante o final de 1997 e o início de 1998, eu e o Fábio, interagíamos bastante na manutenção do provedor e trocávamos muitas idéias sobre a possibilidade de adoção do GNU/Linux em maior escala pela Univates. Entre 1997 e 1999, sob a coordenação do Fábio, todos os servidores que usavam sistemas operacionais proprietários na Univates foram substituídos por servidores GNU/Linux. Na mesma época eu deixava a sociedade no provedor de acesso para fundar a Brod Tecnologia[12] e dedicar-me à consultoria e desenvolvimento de soluções em software livre. Em agosto de 1999, eu e o Fábio, viajamos para a Califórnia para participar da LinuxWorld Conference and Expo onde, dentre outras coisas, participamos de tutoriais sobre PHP e MySQL. Em outubro de 1999, a Brod Tecnologia foi contratada para assumir a gestão dos recursos de informática da Univates, intermediando também a relação da instituição com a Solis, Cooperativa de Soluções Livres[13], sobre a qual escrevo no capítulo [completar aqui], quando propusemos a migração do sistema acadêmico/administrativo existente e desenvolvido em software proprietário para um novo sistema, a ser desenvolvido internamente, totalmente em software livre. Como o sistema existente já estava dando sinais de que não aguentaria o crescimento em sua base de dados que seria causado por novas matrículas de alunos e uma alternativa comercial proprietária implicaria em investimentos da ordem de R$ 100.000,00, no mínimo, obtivemos a autorização da reitoria para tocar o projeto que mais tarde foi batizado de Sagu.

Ainda no final de 1999, a Univates havia feito um investimento na ampliação de sua estrutura de informática, com a instalação de mais de 100 novas máquinas desktop. Decidimos que não adquiriríamos licenças de softwares de produtividade e escolhemos o StarOffice como padrão. A razão, pela qual justificamos esta escolha foi o custo do pacote equivalente, enquanto o StarOffice era distribuído gratuitamente pela Sun Microsystems. A principal vantagem técnica do StarOffice, porém, é a de que o mesmo possuía uma interface consistente independente do sistema operacional adotado, o que veio a permitir a migração do sistema operacional das máquinas dos usuários para o GNU/Linux.

Mesmo considerando as dificuldades iniciais em qualquer migração de ambiente, o sucesso na adoção do software livre na Univates motivou o desenvolvimento de novos sistemas como o Gnudata[14],o Gnuteca[15] e especialmente o framework Miolo[16] entre tantos outros, assim como a utilização de outros softwares livres desenvolvidos externamente, especialmente os sistemas Teleduc[17] e Rau-Tu[18], originados na UNICAMP.

9 http://www.univates.br
10 http://penta.ufrgs.br/liane.html
11 http://www.bewnet.com.br/
12 http://www.brod.com.br
13 http://www.solis.coop.br
14 Infra-estrutura para a implementação de bases de dados analítico-estatísticas usada no Banco de Dados Regional da Univates - http://www.bdr.univates.br
15 Sistema de gestão de acervo, empréstimos e colaboração para bibliotecas - http://gnuteca.codigolivre.org.br
16 http://www.miolo.org.br
17 Sistema para a educação à distância - http://teleduc.nied.unicamp.br/teleduc/
18 Sistema para a criação de bases de conhecimento - http://www.rau-tu.unicamp.br

Os reflexos da adoção do software livre na Univates são visíveis até hoje. Várias empresas incubadas na Inovates, a incubadora empresarial da instituição, são usuárias ou desenvolvedoras de softwares livres. Marcelo Malheiros, que desenvolveu os projetos Rau-Tu e Nou-Rau, idealizados por Rubens Queiroz de Almeida, da UNICAMP, é hoje coordenador dos cursos de informática da Univates. O Núcleo de Tecnologia da Informação da instituição, o antigo CPD, desenvolve a quase totalidade dos sistemas internos com softwares livres como a linguagem PHP e a base de dados PostgreSQL. A própria Solis ainda é uma das grandes fornecedoras de serviços de suporte em infraestrutura para a Univates.

É muito difícil imaginar, hoje, o que poderia ter acontecido caso uma fortuita série de fatores não houvessem se combinado no final dos anos 1990 e princípio dos anos 2000, justamente tendo a Univates como cenário. É verdade que construí um projeto pessoal e profissional onde imaginava desenvolver soluções a partir de tecnologias livres e eu precisava de um ambiente, como a Univates, que servisse de terreno intelectualmente fértil onde estas ideias pudessem ser semeadas. Há alguns anos antes, em 1995, ainda na Tandem, eu havia participado de um treinamento chamado "Solution Selling[19]", onde aprendíamos técnicas para uma venda honorável, na qual cliente e fornecedor sairiam sempre ganhando. Ao rever o material para este livro noto o quanto este treinamento ficou carimbado na minha mente e o quanto ele influenciou minha "técnica de venda" e até de gestão de desenvolvimento. As técnicas ensinadas no "Solution Selling" visam habilitar o vendedor a entender a "dor" de seu cliente e, a partir daí, buscar a melhor cura para esta dor, sempre reconhecendo que quem conhece efetivamente a dor que sente é o próprio cliente.

Mas analisando mais profundamente o que coloco no capítulo "Vendendo a ideia", vamos rever a construção do ambiente favorável na Univates para que a ideia da adoção de soluções em software livre fosse comprada pela instituição.

Quando desenvolvemos um *projeto em software livre*, ele será bem sucedido se:

1. a equipe está engajada, comprometida com ele, entende a filosofia do software livre e com isto está disposta a trabalhar de forma colaborativa, mesmo com pessoas externas à equipe;
2. a equipe tenha fé e respeito em seu líder, que deve entender que está coordenando um projeto que pode ter colaboração que está além de sua gestão;
3. que toda a estrutura organizacional superior o apoie e saiba que não é "dona" do que está sendo desenvolvido.

O Grupo de Usuários Gnurias

Ninguém sabia muito bem o que era software livre no início dos anos 2000. O projeto que começávamos a desenvolver na Univates, implicando no desenvolvimento e na adoção de softwares livres exigia que, rapidamente, conseguíssemos "aliados" à nossa causa. A quase totalidade das pessoas que trabalhavam conosco no CPD era formada por alunos da instituição. Em especial, nossos monitores de laboratórios tinham que, ao mesmo tempo que aprendiam a usar o Linux e o StarOffice (que depois tornou-se o OpenOffice.Org), lidar com a resistência dos seus colegas usuários dos laboratórios, acostumados a usar os softwares da Microsoft em suas casas e exigiam que a instituição lhes oferecesse o mesmo tipo de ferramentas.

Tomamos uma série de ações para minar esta resistência, garantir o engajamento não só da equipe mas também da maior parte possível dos usuários e evangelizar as pessoas na filosofia do software livre. Ministramos cursos internos sobre o uso das ferramentas, promovemos seminários e buscamos que nossa equipe participasse dos eventos sobre software livre que começavam a surgir

19 http://en.wikipedia.org/wiki/Solution_selling

no Brasil. Eu sempre buscava que, cada coisa que aprendêssemos, descobríssemos ou desenvolvêssemos ganhasse o formato de um artigo ou palestra que pudesse ser apresentado em algum evento. Como tudo era muito novo, nossa produção era intensa e muitos da nossa equipe palestraram e participaram de eventos em vários lugares do Brasil e do mundo. Só estas histórias dariam um livro à parte, mas vou concentrar-me na história do grupo de usuários Gnurias.

Desde a primeira edição do Fórum Internacional de Software Livre a nossa equipe do CPD da Univates participou ativamente do evento, desde a apresentação de palestras até o suporte à estrutura de acesso à Internet. Durante a segunda edição, em 2001, eu e a Ana Paula Araújo, a Preta, assistíamos a uma palestra do grupo LinuxChix[20] e ela teve a ideia de criarmos um grupo com uma ação mais local, com foco menos técnico e mais inclusivo, com ações de popularização do uso de tecnologias livres. Na mesma hora sugeri o nome Gnurias e a coisa pegou. Em um primeiro momento a Preta, a Joice Käfer, a Ana Paula Fiegenbaum, a Viviane Berner e a Marceli Arnhold formaram o grupo do qual fui nomeado, orgulhosamente, padrinho.

Dentre as ações do grupo estavam a “Semana do Pinguim”, que entre 2002 e 2004 acontecia todos os semestres na Univates, em 2005 passou a ser anual e em 2006 teve a sua última edição. Durante a “Semana do Pinguim” vinham para a Univates palestrantes de vários lugares do Brasil falar sobre suas experiências com o uso de software livre, o que servia como um grande incentivo aos alunos da Univates. Na edição de 2006 fomos além do software livre, com oficinas sobre pandorgas e rádios de galena.

Em parceria com uma escola municipal de nossa região, o grupo ministrou oficinas de informática para crianças, sempre com a ideia da tecnologia tornar-se um elemento de apoio na aprendizagem das matérias que já eram ministradas na escola. Além disto, em outras escolas foram ministradas oficinas de criação de páginas web, de uso do OpenOffice.Org, de ferramentas educacionais como o gCompris e muitas outras. Em parceria com a Univates as meninas do grupo ainda deram aulas de informática para as pessoas da melhor idade, ensinando os vovôs e vovós a trocarem e-mails com seus filhos e netos, muitas vezes distantes e com os quais há muito não se comunicavam.

Acredito que a última ação do grupo foi um curso que ministramos para jovens desempregados, em 2007, em uma parceria com o SINE da cidade de Arroio do Meio, onde em uma semana dávamos aos jovens elementos para que eles pudessem enriquecer seus currículos com o conhecimento básico de ferramentas de informática para o uso em escritórios. A produção do currículo era, em si, um dos elementos do curso.

Infelizmente o grupo Gnurias foi minguando na medida em que suas componentes se formaram, saíram de Lajeado ou começaram outros empreendimentos. Por alguma razão para a qual ainda buscamos explicação, não conseguimos manter aceso o entusiasmo do grupo e fazer com que o mesmo sobrevivesse com novos membros. Ainda assim, de uma forma ou outra, as ações voluntárias continuaram acontecendo de uma forma ou outra com a participação de uma ou outra componente do grupo.

O Gnurias foi, entretanto, fundamental quanto à disseminação e o engajamento com a cultura do software livre enquanto desenvolvíamos nossos projetos na Univates.

Sagu Workshop e SDSL

Na gestão de qualquer projeto inovador é necessária uma grande capacidade de liderança. No caso de um projeto com software livre eu ouso dizer que o gestor deve ser quase um líder espiritual, um grande motivador da equipe. Além disto, a equipe deve estar sempre aberta à colaborações externas, vindas de descobertas de códigos melhores já produzidos por outros ou mesmo pela colaboração

20 http://www.linuxchix.org/

espontânea na forma de código ou críticas. Busco exercer a liderança de forma a permitir que todos na equipe tenham voz e sejam valorizados pelo seu talento. Sempre que possível, exponho este talento ao reconhecimento externo, de desconhecidos. O que pode parecer a um novo colaborador, a princípio, assustador, acaba por ser muito gratificante quando outros, além da equipe, passam a valorizar o seu talento.

Quando estávamos ainda no início do desenvolvimento do Sagu, nosso sistema de gestão acadêmica sobre o qual falaremos mais adiante, decidi que, antes de colocar o sistema em produção, ofereceríamos à comunidade o Sagu Workshop, onde contaríamos a todos o que estávamos fazendo, incluindo nossas motivações e expondo o nosso código. Em março de 2000 ministramos o primeiro Sagu Workshop na Univates, com 20 participantes vindos de todos os lugares do Brasil – uma tremenda surpresa para nós! Durante uma semana ministramos oficinas de PHP, PostgreSQL e mostramos como o Sagu, que ainda entraria em produção em julho do mesmo ano, funcionaria. O sucesso foi tanto que ainda tivemos mais duas edições do Sagu Workshop, até ele tornar-se o SDSL – Seminário de Desenvolvimento em Software Livre, evento que era organizado em parceria com a Univates, Unisinos e Unicamp e mais adiante, teve a parceria também da UnB.

Ainda que a contribuição direta ao código do Sagu (e de outros sistemas que desenvolvemos depois) não tenha sido muito grande, a contribuição indireta na forma de troca de ideias sobre o melhor uso do PHP, do PostgreSQL e outros componentes do sistema sempre foi de valor inestimável. A exposição dos desenvolvedores às ideias e códigos alheios faz com que o apego ao próprio código diminua em função do amor maior a um conhecimento que pode ser construído coletivamente.

A Univates como referência, não como proprietária

Quando assumi a gestão de TI da Univates a “dor” relativa ao sistema acadêmico existente era muito presente e forte. Ele já mostrava sinais de desgaste e simplesmente não suportaria o volume de dados que seria gerado a partir do vestibular de inverno de 2000. Eu e o Fábio Wiebbelling embasamos muito bem nossa proposta de construção de um novo sistema acadêmico utilizando softwares livres e o modelo do sistema anterior, bem conhecido pela equipe de desenvolvimento que estava na instituição. Como a Univates estava em um momento de crescimento, com necessidade de investimento financeiro nas mais variadas áreas (em especial, na construção de novos espaços para acomodar os novos alunos), nossa proposta, ainda que temerária, veio a calhar. Hoje, revendo tudo o que passou, vejo que qualquer outra proposta seria tão temerária quanto a nossa, mas a nossa foi a mais transparente e sincera, oferecendo uma nova tecnologia para a construção de uma solução que aproveitaria o conhecimento adquirido na solução existente. Obviamente, era também a mais barata.

O apoio da organização existiu desde o princípio, o que não impediu repetidas perguntas sobre a propriedade do que estava sendo desenvolvido, um assunto eternamente retomado por muitos anos. Não sei se até hoje.

Nossa equipe que já desenvolvia o Sagu e outros sistemas tinha a plena consciência de que havia se apropriado de uma série de coisas para poder escrever o Sagu, a começar pela própria linguagem de programação, o PHP, e a base de dados, o PostgreSQL, sem contar com a infinidade de bibliotecas para a geração de documentos, autenticação de usuários, e a lista segue interminável. Este era um dos argumentos que usávamos para dizer que o Sagu (e outros sistemas produzidos por nós) deveriam ser entregues à comunidade como software livre, da mesma forma que nos apropriamos de seus componentes. Mas o argumento mais forte sempre foi comparar nosso modelo de desenvolvimento ao próprio modelo de negócio da instituição, que era o de formatar o conhecimento para fornecer a seus alunos, sem ser necessariamente a dona do mesmo. Esta argumentação ficou mais fácil quando passamos a usar sistemas desenvolvidos por outras

instituições, como o Teleduc e o Rau-Tu.

O desenvolvimento do Sagu

Ainda antes da efetiva contratação da BrodTec pela Univates, o trabalho junto ao CPD da instituição acabou fazendo com que eu fizesse uma série de pequenos trabalhos pontuais de consultoria e efetivas intervenções no ambiente de informática da instituição em conjunto com o Fábio Wiebbelling, chefe do CPD e uma das pessoas que mais conhece o ambiente GNU/Linux em todo o mundo mesmo nos dias de hoje. Quando fomos juntos à LinuxWorld Expo na Califórnia, em 1999, já tínhamos como missão levantar alternativas para o desenvolvimento de um novo sistema administrativo, uma vez que o que estava sendo usado, baseado em softwares proprietários, começava a dar sinais de que sua vida estava chegando ao final. Contínuos problemas de instabilidade do sistema indicavam que conseguiríamos mantê-lo com alguma confiabilidade apenas até meados de 2000. O incremento da base de dados com as matrículas que seriam efetuadas no vestibular de inverno daquele ano levariam o sistema ao colapso. Mesmo antes disto, enquanto o Sagu já estava sendo desenvolvido, sofremos problemas de corrupção da base de dados que nos obrigaram algumas vezes a restaurar backups de dois ou três dias anteriores aos problemas, limpar a base (muitas vezes limitando o acesso de usuários à informações históricas) e voltar o sistema à ativa com riscos cada vez maiores. Por outro lado, o modelo ER (Entidade Relacionamento, sobre o qual estarei falando no capítulo [aqui]) usado para a construção do sistema havia sido muito bem projetado pelo Professor Luciano Brandão (hoje sócio da OM Informática), auxiliado pelo Maurício de Castro (hoje sócio da Sol7) e pelo Carlos Bush (hoje gerente de soluções e inovações da Processor) e foi essencial para o rápido desenvolvimento do Sagu, uma vez que o adotamos quase integralmente.

Na LinuxWorld, enquanto o Fábio concentrava-se na aquisição de experiências e troca de idéias em implementação de ambientes seguros de redes e desempenho, eu pesquisava linguagens e bases de dados em software livre que fossem boas alternativas para o nosso desenvolvimento. Foi quando tomei maior contato com o PHP e pude ver a facilidade e velocidade que a linguagem permitia na criação de aplicações voltadas para a web, que já era o nosso desejo. Passamos noites muito divertidas no Motel 6, debruçados em cima do único livro sobre PHP que havia disponível [citar o livro] e testando uma série de conceitos necessários ao desenvolvimento do nosso novo sistema.

Quando retornamos ao Brasil, apresentamos nossas ideias ao CPD e montamos um projeto de migração que levamos à reitoria. O projeto consistia em desenvolver um novo sistema que replicasse toda a funcionalidade do existente e desse espaço para novas implementações de forma segura e com boa escalabilidade. Isto deveria ser feito em menos de seis meses. Assim, a consultoria da BrodTec foi contratada em regime integral e o projeto começou. Lembro que o Professor Eloni Salvi, nosso pró-Reitor administrativo e financeiro, que efetivamente convenceu seus colegas na reitoria disse a mim e ao Fábio na aprovação do projeto: “Senhores, vossa cabeça acaba de ser colocada à prêmio e, junto delas, a minha...”. Como vocês estão lendo este texto, podem concluir que nossas cabeças estão à salvo, ainda que envolvidas em novos projetos.

O Sagu começou a ser desenvolvido com as seguintes premissas:

> **Independência de base de dados**: Como os problemas que enfrentávamos no sistema antigo eram em grande parte devidos à amarração que tínhamos com uma base de dados proprietária, o novo sistema nos deveria permitir a fácil migração para qualquer outra base caso isto se mostrasse necessário.
>
> **Interface Web**: Os usuários deveriam poder acessar o sistema independente do sistema operacional que rodassem em seus computadores e o novo sistema também deveria permitir que no futuro migrássemos as estações dos clientes também para software livre (o que começou a ser feito em janeiro de 2000). A melhor maneira de se conseguir isto era

utilizando um browser padrão (na época o Netscape ou o Internet Explorer) como cliente.

Modularidade: O sistema seria dividido em módulos específicos e bem definidos, de forma que o processo de desenvolvimento pudesse ser melhor dividido entre os membros da equipe e facilitasse a colaboração de outras pessoas.

Em janeiro de 2000 o novo sistema processou em paralelo o vestibular de verão da Univates, com sucesso e velocidade superiores ao esperado. Processamentos de classificação que levavam mais de três horas no sistema anterior, em um servidor Pentium II 400 com 512 Mbytes de memória, demoravam apenas alguns minutos em uma máquina Pentium 200 com 64 Mbytes de memória usada nos testes (o valoroso desktop do Maurício!).

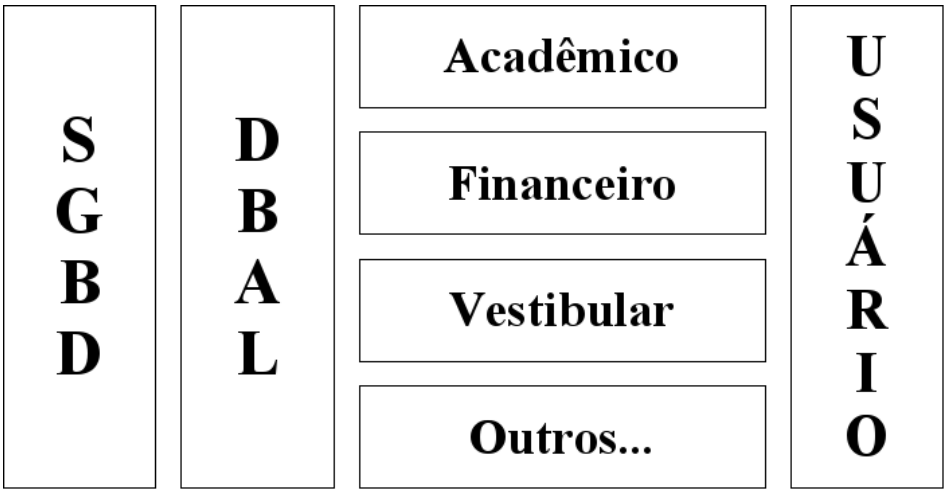
O sistema foi batizado de Sagu (inicialmente um acrônimo para Sistema Aberto de Gestão Universitária e hoje Sistema Aberto de Gestão Unificada) pelo Professor Eloni Salvi quando precisamos de um nome para apresentar o projeto[21] no Workshop de Software Livre, que ocorreu em paralelo ao 1.o Fórum Internacional de Software Livre, nos dias 4 e 5 de maio de 2000.

O Sagu entrou em produção em julho de 2000 e até a sua substituição pelo projeto Alpha, desenvolvido internamente pela Univates, foi o responsável pela automação do relacionamento acadêmico/administrativo e financeiro de mais de 10.000 alunos com a Instituição de Ensino. Ainda hoje, o Sagu é um dos carros-chefe do faturamento da Solis.

A arquitetura inicial do Sagu

O desenvolvimento do Sagu deu-se em tempo recorde, aproveitando a modelagem da base de dados do sistema anterior. Se por um lado, isto nos permitiu colocar o Sagu em produção na Univates em menos de seis meses, por outro herdamos uma estrutura de dados que não era a mais adequada ao crescimento e ao desenvolvimento cooperativo do sistema. Para a equipe do Sagu era relativamente simples olhar o modelo ER (Entidade-Relacionamento) da base de dados e entender as necessidades de negócio atendidas por este modelo. Para quem estava de fora, porém, um conjunto de tabelas e seus relacionamentos não ilustram de maneira fácil a finalidade do Sagu.

O Sagu possuía um módulo de abstração da base de dados (DBAL - Database Abstraction Layer), que consistia no programa common.php3 e permitia que o sistema utilizasse outras bases de dados que não o PostgreSQL (ainda que, na prática, isto não tenha sido feito). Cada um dos módulos do Sagu, porém, era responsável pelo acesso à base de dados e à interface com o usuário.



21 http://www.inf.unisinos.br/~marinho/Teaching/local/WSL2000/Definitivo/016-brod.ps

Arquitetura Miolo

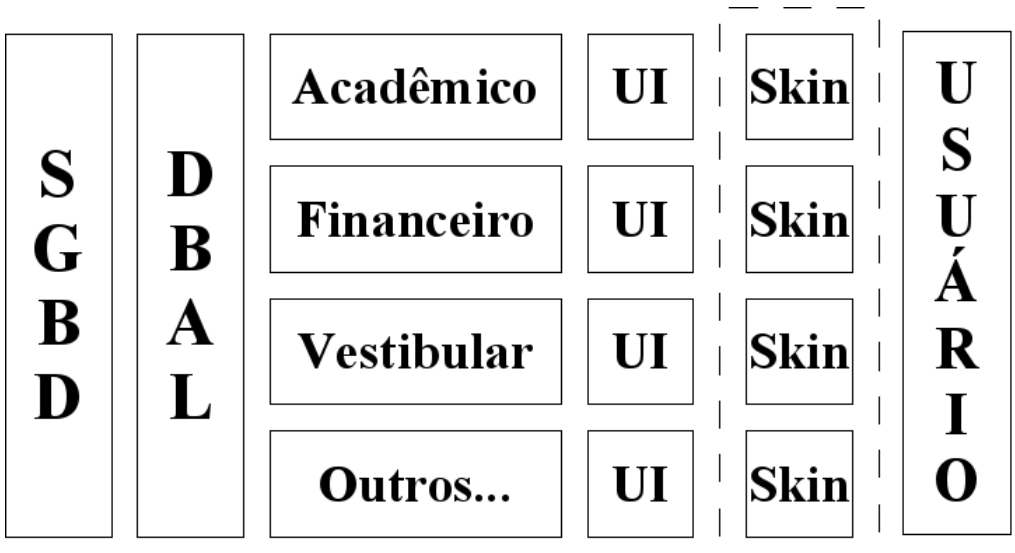
Com o Sagu já em produção, na segunda metade do ano 2000 iniciamos a produção de um sistema para o Banco de Dados Regional do Vale do Taquari[22], hospedado na Univates. Durante o desenvolvimento deste sistema acabamos aproveitando muito do código desenvolvido para o Sagu e começamos a sistematizar melhor o nosso desenvolvimento. Em 2001 convidamos algumas pessoas externas à instituição, como o desenvolvedor Alessandro Binhara, o Prof. Leandro Pompermayer e o consultor Thomas Sprietersbach (que nos ajudou também em inúmeras outras ocasiões) para que tomassem conhecimento do que estávamos desenvolvendo e pudessem contribuir com suas sugestões. O resultado concreto deste workshop interno de transferência de tecnologia foi o framework de desenvolvimento Miolo, base da nova versão do Sagu e de praticamente todas as soluções desenvolvidas pela Univates e posteriormente pela Solis. Durante o desenvolvimento do Miolo tivemos também a honra de ter contribuições diretas de Rasmus Lerdorf, criador da linguagem PHP, que trabalhou com nossa equipe, em Lajeado, por cerca de dez dias.

As seguintes premissas foram a base para o desenvolvimento do Miolo:

- a documentação do sistema deveria ser clara e criada de maneira dinâmica, ao mesmo tempo em que o sistema é desenvolvido;
- o framework deveria tratar todas as questões de segurança e perfis de usuários, isolamento e comunicação entre os módulos desenvolvidos com ele;
- os módulos funcionais que compõem os sistemas a serem desenvolvidos deveriam refletir a necessidade de negócio que atendem, de tal maneira que, quem os programe, não precise necessariamente conhecer a base de dados;
- a construção da interface de usuário deveria poder ser feita por um webdesigner, que não necessitasse conhecer a fundo as características internas do sistema.

Desta forma, conseguiríamos dividir melhor as tarefas de desenvolvimento do Sagu e de outros sistemas. Um designer, por exemplo, poderia concentrar-se em criar uma interface agradável para o usuário, sem ter que conhecer a base de dados. Um administrador de base de dados poderia concentrar-se em aspectos de performance da base, não necessitando conhecer profundamente os programas que a acessam.

Assim, a nova arquitetura proposta para o Sagu, com o framework Miolo, passou a ser a seguinte:



Note que agora, seguimos com o modelo de abstração de base de dados, mas não mais na forma de

22 http://www.bdr.univates.br/

um programa como na versão anterior do Sagu, mas sim através de uma série de objetos fornecidos pelo framework Miolo que fazem a conexão e demais funções de armazenamento e busca de informações na base. Outros objetos do Miolo permitem a construção da lógica de negócio, sempre usando o paradigma de orientação à objetos. A forma como a comunicação com o usuário será feita pelo programador da lógica de negócio, mas a interface pode ser feita por um webdesigner, através do uso de temas e folhas de estilo, que podem ser coerentes para todos os módulos do sistema, ou diferenciados, de acordo com as necessidades e características dos usuários.

Gnuteca – um teste de conceito para o Miolo

Neste capítulo começo a tratar mais explicitamente questões de gestão e metodologia de desenvolvimento, falando também sobre as opções que sempre se apresentam e as decisões que temos que tomar, mesmo sem termos todos os elementos para tal.

Quando desenvolvemos o Sagu na Univates, tínhamos uma pressão tremenda de tempo em função da necessidade do esgotamento do sistema administrativo usado anteriormente. Aproveitamos o modelo ER deste sistema anterior, buscamos mapear o máximo possível às telas da interface web para que tivessem alguma similaridade à aplicação com a qual os usuários já estavam acostumados e entre janeiro e julho de 2000 o sistema foi desenvolvido e entrou em produção. Durante este período, nossa equipe ainda estava aprendendo a usar a linguagem PHP. Não houve um rigor maior na documentação do sistema ou mesmo uma preocupação com sua possibilidade de seu crescimento. Ainda assim, o Sagu provou-se viável e fomos agregando a ele uma série de funcionalidades e adicionando usuários ao sistema. Sabíamos que teríamos que reescrevê-lo desde o princípio e esta foi uma das principais razões de termos criado o framework Miolo. Mas o Sagu tinha uma complexidade tal que preferimos optar por amadurecer o Miolo com o desenvolvimento de uma aplicação nova, o Gnuteca.

O Gnuteca[23] é um sistema em software livre composto de módulos para gerir acervos bibliográficos, controlar empréstimos, pesquisar em bases bibliográficas e administrar o sistema de forma local e remota, promovendo maior agilidade e qualidade aos serviços prestados por bibliotecas, assim como prover o fácil intercâmbio de informações com outros sistemas de Bibliotecas, mantendo-se dentro de normas internacionais ISO e do padrão internacional de catalogação MARC 21. É um sistema abrangente e genérico que pode moldar-se a diferentes realidades de diferentes usuários, permitindo também a criação de uma infraestrutura de colaboração entre bibliotecários e demais funcionários das bibliotecas, evitando a repetição desnecessária de trabalho: uma vez feita a catalogação de um título em uma biblioteca, estes dados catalográficos podem ser "importados" para o sistema de outra biblioteca que adquira o mesmo título.

Com a integração de novas pessoas ao nosso CPD e já com alguma perspectiva de agregar desenvolvedores externos a nossos projetos, decidimos adotar alguma metodologia consagrada que nos permitisse documentar nossos sistemas, facilitando a interação com os usuários (clientes) e a comunicação de necessidades e resultados ao público externo. Optamos por utilizar a UML (*Unified Modeling Language*) como padrão de desenvolvimento e algumas características de *Extreme Programming* (XP)[24] como padrão de atitude.

Antes de iniciar o projeto, realizamos uma oficina prática de UML e orientação à objetos para trazer à nossa equipe a experiência de profissionais da área de análise e projeto de sistemas. Esta oficina acabou por definir muitos dos conceitos que seriam aplicados ao Gnuteca.

Quem deve definir o que o sistema fará é quem irá utilizá-lo. Assim, o Gnuteca iniciou-se com o levantamento de todos os processos utilizados em uma biblioteca, desde a forma de catalogação de acervo e seus padrões até a interação com o usuário no balcão de atendimento, passando por consultas ao acervo, relatórios e o relacionamento com outros sistemas. Tudo isto foi documentado em "casos de uso", "diagramas de sequência", "diagramas de classe" e "diagramas de atividade". Além de envolvermos o cliente interno (nossas bibliotecárias e funcionários) no processo, contamos com a consultoria da Control[25] e com a participação de vários bibliotecários que começaram a contribuir com sugestões a partir da apresentação do projeto conceitual em 13 de junho de 2001 na

23 http://www.gnuteca.org.br
24 http://www.extremeprogramming.org/
25 http://www.control.com.br/

sede da Control em Porto Alegre. O código-fonte em produção foi sempre disponibilizado prematuramente no site do projeto, a fim de que a comunidade pudesse acompanhar sua evolução, ainda que em estágios não funcionais.

O Miolo foi utilizado para o desenvolvimento do módulo Web de interação com o aluno (pesquisas, renovações, reservas), assim como para os módulos de administração e catalogação, acessados pelos bibliotecários através da Intranet. Para os módulos usados no balcão de atendimento escolhemos o PHP-GTK[26].

> *Nota: Pablo Dall'Oglio, da equipe de desenvolvimento do Gnuteca, é hoje proprietário da empresa Adianti e o principal autor brasileiro sobre PHP-GTK e orientação a objetos com a linguagem PHP.*

O Gnuteca, entrou em fase de testes no campus da Univates da cidade de Encantado em outubro de 2001. Nessa fase foram realizados diversos ajustes ergonômicos e operacionais do sistema de acordo com solicitações dos nossos usuários. Em 25 de fevereiro de 2002 o sistema entrou em produção no campus central da Univates atendendo a mais de 5 mil usuários com um acervo de mais de 50 mil exemplares.

Ainda que eu retome mais adiante este assunto, já adianto que o custo de desenvolvimento do Gnuteca até a sua primeira versão não chegou a R$ 50.000,00 (dados de meados de 2002) e que o mesmo não possui nenhum pré-requisito que exija o pagamento de licenças, independente do número de usuários ou o acesso à base de dados.

26 http://gtk.php.net – Até onde tenho conhecimento, o sistema de atendimento em balcão do Gnuteca é o maior projeto já escrito em PHP-GTK

Utilizando a UML

Antes de começar este capítulo já vou dizendo que não está no escopo deste livro ensinar a utilizar a UML. Para isto há uma vasta documentação na web[27] e também impressa à qual eu não teria muito o que acrescentar. A UML também não é a única forma de se modelar o desenvolvimento de um sistema, mas é, com certeza, uma das mais utilizadas. Uma das principais vantagens da UML é a de que, através de seus diagramas, podemos evoluir desde uma documentação gráfica que os usuários do sistema entendem até um diagrama de sequência que se aproxima bastante da forma como o programa será escrito com o paradigma de orientação a objetos. Outra questão, por vezes difícil de admitir, é a de que programadores dificilmente gostam de documentar os sistemas que desenvolvem. Na verdade, quanto mais especializados se tornam os programadores, menos ainda eles gostam de documentar os sistemas. Por isto é importante que se incentive, cobre e motive uma boa documentação inicial dos sistemas a serem desenvolvidos, antes que os programadores comecem a codificar. Posteriormente, busca-se incluir processos de documentação automática e o envolvimento de uma pessoa que se encarregue de manter a documentação atualizada, mesmo que os programadores não o façam. Esta é uma tarefa bastante árdua.

Um outro benefício da UML é o fato de existirem alguns programas em software livre que permitem a modelagem de uma maneira bastante prática e até agradável. No desenvolvimento do Gnuteca optamos por utilizar o Dia[28]. No site oficial do Gnuteca (www.gnuteca.org.br) estão disponíveis alguns dos diagramas UML utilizados no desenvolvimento do sistema.

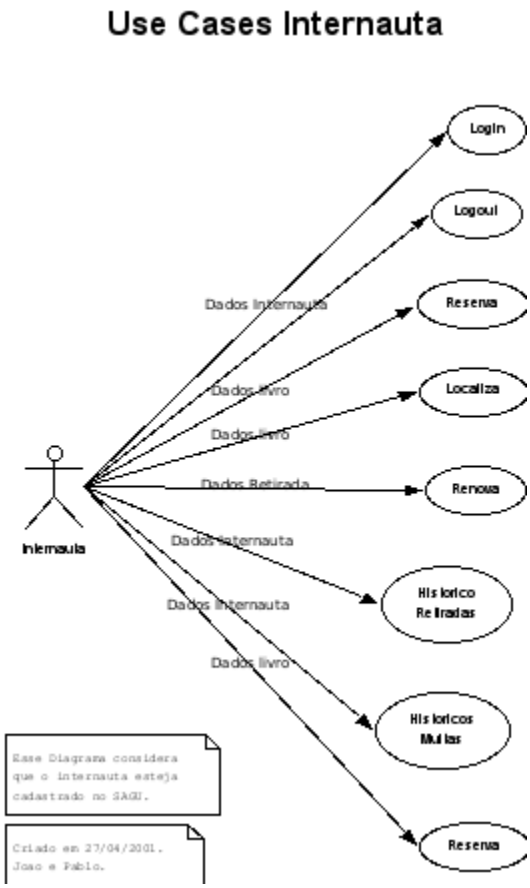
Mas o que é a UML afinal?

A UML, *Unified Modeling Language* (Linguagem Unificada de Modelagem) é uma proposta do OMG[29], *Object Management Group* (Grupo de Gestão de Objetos) visando criar um padrão de documentação para o desenvolvimento de sistemas, facilitando sua compreensão entre grupos de desenvolvedores e a interoperabilidade entre os sistemas desenvolvidos. As especificações do OMG acabaram por se tornar padrões de fato mesmo no desenvolvimento de sistemas em software livre, uma vez que as mesmas estão livremente disponíveis para download no portal da organização.

Na prática, a UML permite ao desenvolvedor uma linguagem com a qual ele pode ao mesmo tempo ir fazendo uma montagem mental do sistema a ser desenvolvido enquanto o próprio usuário pode entender o que está sendo desenvolvido e assim interagir com o desenvolvimento.

Normalmente, o primeiro diagrama com o qual trabalhamos na UML é justamente o de *Use Cases* (Casos de Uso), usado para a obtenção de requerimentos para o sistema a ser desenvolvido.

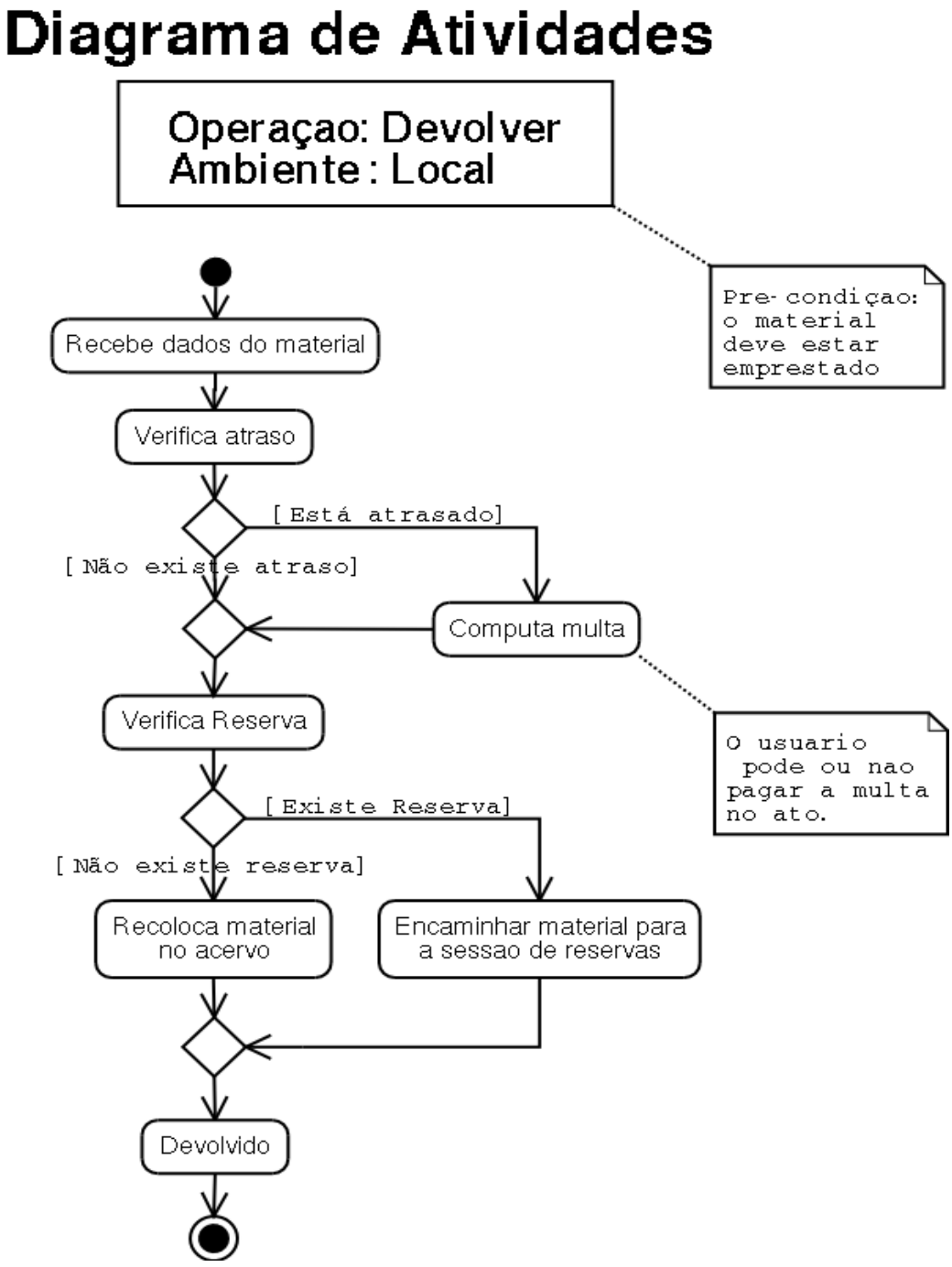
27 Comece por este link: http://www.uml.org e siga para *Getting Started With UML*
28 http://www.gnome.org/projects/dia/
29 http://www.omg.org



O diagrama acima ilustra um “ator”, no caso um usuário da biblioteca conectado ao sistema através da Internet. Dentro de cada elipse estão as operações possíveis que um usuário pode realizar e, junto a cada seta, os dados necessários para a realização de cada operação. Note que para a montagem deste diagrama basta uma conversa com o bibliotecário, perguntando o que será disponibilizado de serviços para os usuários da biblioteca através da Internet. Para cada tipo de ator nas mais variadas situações é elaborado um diagrama deste tipo. O diagrama de caso de uso está incluído nos diagramas de comportamento da UML, justamente por definir mais aspectos comportamentais mesmo do que aspectos técnicos.

O próximo diagrama que utilizamos é o de “atividades”, outro diagrama de comportamento. Ele define a forma e o fluxo de como uma operação ocorre. Nele já devemos buscar evidenciar condições e tratamentos de exceções ou erros.

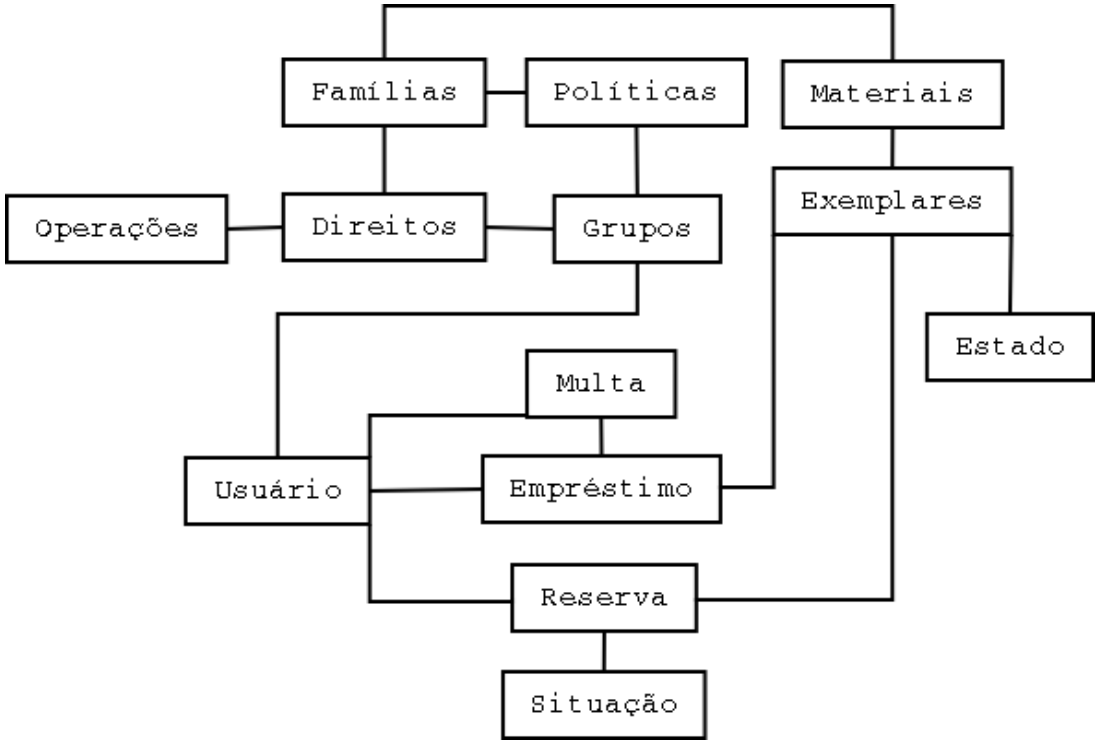
No exemplo abaixo temos uma operação de devolução. É uma operação realizada no próprio balcão de atendimento da biblioteca. O que acontece neste diagrama é que cada uma das operações possíveis, descobertas durante a elaboração dos “casos de uso”, são levantadas e analisadas à fundo. Tipicamente, nesta fase, o programador já começa a “enxergar” o programa, enquanto o usuário ainda é capaz de entender, questionar e colaborar com os diagramas.



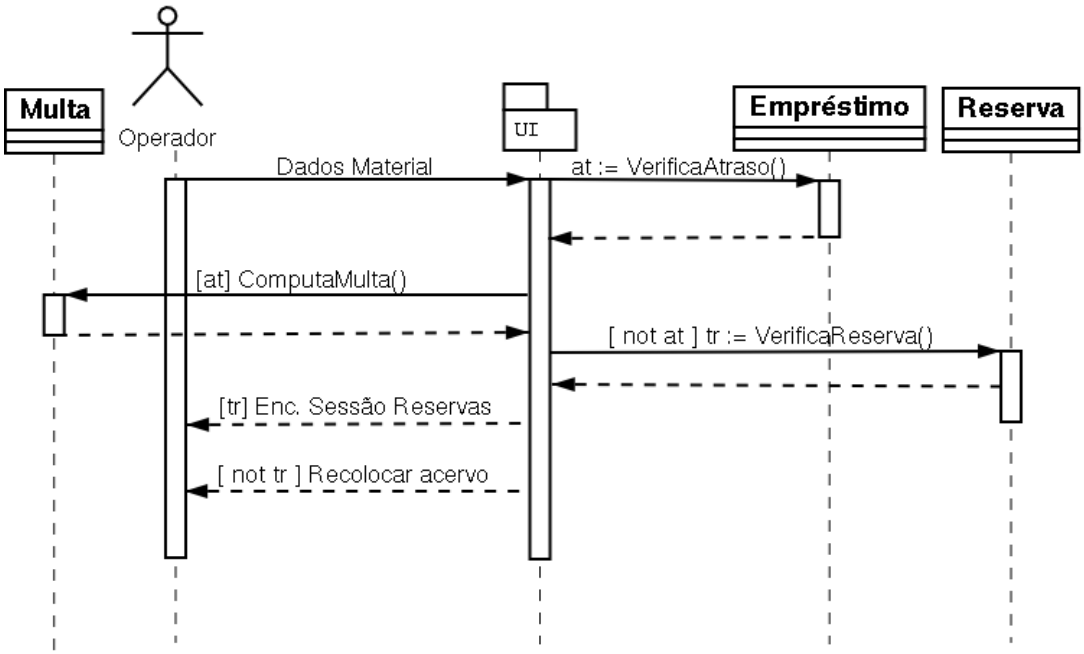
O próximo diagrama é o de “classes”, um diagrama de estrutura da UML. Nele procuramos entender quais são os objetos todos que fazem parte do sistema e sua relação uns com os outros. Abaixo temos o exemplo de um diagrama simplificado de classes do Gnuteca.

Ainda que este diagrama não seja tão fácil de entender por quem não esteja envolvido com um sistema de bibliotecas, observe que Materiais está associado à Famílias, o que leva à conclusão de que os Materiais disponibilizados na Biblioteca podem estar agrupados em Famílias (Livros, Vídeos, Periódicos). Materiais estão associados também à Exemplares, pois pode haver a disponibilidade de múltiplos exemplares de um determinado material. Exemplares estão associados a um determinado Estado: o material pode estar disponível, emprestado, reservado ou ainda em recuperação. Note que o Usuário está ligado ao Material não diretamente, mas através de um Empréstimo. O Usuário ainda pertence a um Grupo ao qual estão associadas Políticas, assim como

as Políticas também estão associadas à Famílias. Isto acontece porque cada grupo de materiais, associado a um determinado grupo de usuários, pode ter uma política diferenciada de empréstimo. Um usuário que pertença ao grupo “Professores” pode tomar emprestado um livro da Biblioteca por um período de duas semanas, enquanto um usuário do grupo “Alunos” poderá ficar apenas uma semana com o mesmo livro. A Univates disponibiliza calculadoras científicas na biblioteca, que só podem ser emprestadas a um grupo específico de usuários. É importante visualizar as classes (ou objetos) que compõem um sistema até para que algumas decisões de implementação sejam tomadas. Neste caso, uma das coisas que se evidenciaram foi a definição de que todas as políticas de empréstimo deveriam ser parametrizáveis diretamente pelo administrador da biblioteca.



O próximo diagrama que utilizamos foi o de sequência, um diagrama de interação da UML. Este já é um diagrama mais complexo e mais próximo da programação. Ele é especialmente útil quando novos programadores são integrados ao desenvolvimento do sistema, uma vez que representa visualmente a interação entre os atores através da interface de usuário, lógica de negócio e conexão com bases de dados.



Ainda assim, um diagrama de sequência permite ao analista de sistemas ou ao programador explicar para o usuário o que acontece dentro do sistema, sem a necessidade de assustá-lo com a linguagem de programação. Note que no diagrama acima, o operador, no caso um atendente da biblioteca, recolhe o material que lhe foi entregue e verifica seus dados. No Gnuteca isto é feito através da leitura do código-de-barras impresso no material, com auxílio da interface de usuário, que foi escrita na linguagem PHP-GTK. A seguir, o sistema verifica se há algum atraso na devolução e, havendo, calcula a multa (que pode ser paga na hora ou acrescida no sistema financeiro que a incorporará a outras dívidas do aluno, se for o caso). O passo seguinte é verificar se aquele material devolvido possui alguma reserva, o que é feito diretamente pelo sistema (o Gnuteca permite que os usuários da biblioteca reservem materiais pela Internet e as políticas de uso definem quanto tempo esta reserva poderá ser mantida). Havendo reserva, o material é encaminhado ao setor de reservas, caso contrário, é recolocado no acervo.

Durante o desenvolvimento do Gnuteca, buscamos tornar o sistema o mais genérico e abrangente possível, o que fez com que o mesmo acabasse por ser adotado por bibliotecas universitárias, bibliotecas independentes e mesmo para coleções pessoais.

Extreme Programming

Quando comecei a assistir à uma palestra do pessoal da Hiperlógica sobre Extreme Programming na LinuxExpo em São Paulo, em maio de 2001, cheguei a pensar: "Justo agora que adotamos a UML me aparecem com mais uma metodologia...". Logo notei, porém, que a metodologia Extreme Programming, ou XP, não chega a se opor à UML e é, em praticamente todos os casos, um bom complemento à ela. Mais do que isto, descobri que já utilizávamos no desenvolvimento de nossos projetos em software livre, mesmo que intuitivamente, muitos dos conceitos de XP.

Antes de começarmos a entender o que é XP, vamos analisar um pouco da evolução do desenvolvimento de software ao longo do tempo:

Por volta de 1960 os computadores começaram a ser utilizados cada vez com mais intensidade nas empresas, uma vez que, até então, estavam quase que exclusivamente limitados ao uso militar. As máquinas eram caríssimas, e por isso, seus recursos deveriam ser explorados ao máximo. Os programas eram otimizados ao extremo para a arquitetura do computador em que iriam rodar, e os poucos programadores que existiam não estavam muito preocupados com a legibilidade do que escreviam, até porque não tinham mesmo muita opção. Existem muitas histórias sobre programas de computadores que simplesmente tinham que ser descartados quando uma arquitetura de hardware era substituída por outra.[30]

Em 1968 Edsger Dijkstra[31] escreveu um artigo onde chamava a atenção para o perigo da utilização da declaração GOTO em um programa. Este artigo continha as idéias centrais do que hoje é chamado “Engenharia de Software”. Daí para a frente foram surgindo mais e mais regras para a escrita de programas de computadores, como que buscando colocar ordem em uma terra sem lei. No início dos anos 80 a "ordem na casa" acabou permitindo que se produzisse software de muito melhor qualidade, legibilidade e portabilidade do que o que era produzido alguns anos antes, só que a cada novo problema que surgia, novas regras eram criadas para que tal problema fosse evitado, a ponto que a partir dos anos 90 começaram a surgir questões sobre a distância colocada entre as metodologias existentes e a prática de programação.

Em 1997 Eric S. Raymond escreveu a primeira versão de seu artigo “The Cathedral and the Bazaar”[32], no qual analisava o desenvolvimento do kernel Linux de forma ao mesmo tempo cooperativa e anárquica. A máxima do artigo de Raymond é: “*Given enough eyes, all bugs are shallow*”, ou seja, com muita gente trabalhando no mesmo código, os problemas se evidenciam, e podem, portanto, ser consertados. Dentro da anarquia do desenvolvimento do Linux, surgiu uma metodologia com raízes quase que exclusivamente práticas, a qual é chamada por Raymond de “Bazaar”, em oposição ao método “Cathedral” que pressupõe um conjunto de regras “acadêmicas” para a produção de software.

O método “Bazaar” parece funcionar muito bem para projetos que são de interesse de um grande número de pessoas, como o kernel Linux e vários outros projetos em software livre. Mas quando estamos desenvolvendo um projeto que atenda um determinado nicho, cujas necessidades são de uma população restrita e o grupo de desenvolvedores é pequeno, torna-se necessário o acordo sobre a utilização de alguma metodologia, ainda mais quando se quer que outros desenvolvedores agreguem-se de forma fácil ao projeto.

O Sagu é um bom exemplo. O projeto foi desenvolvido com grande velocidade, entrando em

30 Uma história interessante é a do ITS - Incompatible Time Sharing, um sistema operacional desenvolvido no MIT pela equipe de Richard Stallman que teve que ser totalmente descartado em função da Digital ter abandonado a arquitetura PDP-10. Uma boa tradução deste relato pode ser encontrada em http://www.cipsga.org.br/sections.php?op=viewarticle&artid=61

31 A íntegra do texto de Dijkstra pode ser encontrada em http://www.acm.org/classics/oct95/

32 Leia o artigo na íntegra em http://www.cipsga.org.br/sections.php?op=viewarticle&artid=50

produção na Univates menos de seis meses após sua escrita ter se iniciado. Teve por base o modelo "ER" do sistema anterior, uma vez que o mesmo precisava ser substituído rapidamente dada sua crescente instabilidade e incapacidade da arquitetura utilizada em acompanhar o crescimento da Univates. Quando o Sagu foi entregue à comunidade como um software livre e quando começamos a agregar mais pessoas ao nosso desenvolvimento passamos a sentir cada vez mais a necessidade de trabalhar com alguma metodologia que propiciasse o fácil entendimento do que escrevíamos e colocasse alguma ordem em nossa própria anarquia - uma simples passada de olhos no código do Sagu permite acompanhar a evolução de seu estilo e padrões de codificação. Como descrito anteriormente, no início de 2001 já havíamos decidido que usaríamos a UML como padrão de desenvolvimento do Sagu2 (o Sagu orientado a objetos). Como o Sagu já estava muito grande, começamos a exercitar a UML no desenvolvimento do Gnuteca (Sistema de Gestão de Acervo, Empréstimo e Colaboração entre Bibliotecas) e do Miolo, o ambiente que atende o desenvolvimento de aplicações de bases de dados voltadas para a Web, consistindo em métodos de apresentação, conexão à base de dados entre outros.

Tudo bem até aqui, mas este capítulo não deveria ser sobre Extreme Programming? Ouso dizer que XP é mais um conjunto de regras de conduta e comportamento para desenvolvedores do que uma metodologia de desenvolvimento. Por isto minha recomendação é que se conheça primeiro a UML para depois temperá-la com XP.

Em 1990 Kent Beck[33] começou a estudar em bases práticas o que facilitava e o que dificultava a produção de software, mas sem olhar os programas que eram escritos, e sim focando-se na satisfação do cliente e na forma como as equipes de desenvolvedores trabalhavam. Kent pôde observar que o custo de desenvolvimento de um sistema estava muito mais ligado ao custo das pessoas do que ao hardware ou licenças de software empregados, e que para a boa continuidade e crescimento de um projeto era necessário envolver o usuário (cliente) no processo de produção e tornar cada programa tão claro quanto possível para a leitura por "humanos". Assim, Kent desenvolveu a metodologia XP com base em quatro princípios básicos: *comunicação, simplicidade, realimentação (feedback) e coragem*. A partir de 1996, Kent Beck passou a aplicar seus conceitos na prática na Daimler-Chrysler.

O desenvolvimento de um projeto com XP passa por quatro fases: *Planejamento, Projeto, Codificação e Testes*, cada fase contemplando um conjunto de regras.

Planejamento

O planejamento começa com a escrita de *User Stories*, uma espécie de *Use Cases* (UML) *"diet"*. O usuário ou cliente escreve histórias que representam o que o sistema deve fazer por eles, evitando usar qualquer terminologia técnica. Em conjunto com os programadores, o tempo de desenvolvimento de cada módulo do sistema que contemple cada uma das *User Stories* é estimado. Caso o tempo estimado para o desenvolvimento seja maior do que três semanas a *Use Story* deve ser subdividida em outras.

A fase seguinte é a *Release Planning*, onde a equipe de negócios e a de desenvolvimento concordam com a ordem na qual os módulos do sistema serão desenvolvidos e entregues para a produção. É importante que a equipe de negócios esteja envolvida, pois nem sempre é possível para os programadores avaliarem a importância da disponibilização de um determinado módulo do sistema, ao mesmo tempo que pode ser difícil para a equipe de negócios entender a dependência entre os módulos do sistema e a complexidade de seu desenvolvimento. Para um executivo pode ser crucial que um relatório financeiro possa estar disponível rapidamente, mas ele depende de módulos contábeis que devem ser escritos antes. Ao final do *Release Planning* deve estar claro para todos a

33 Kent Beck escreveu sua própria biografia em http://c2.com/ppr/about/author/kent.html

ordem na qual os módulos do sistema serão disponibilizados e todos devem se comprometer com isto. Para facilitar este planejamento, a equipe deve procurar levar em conta apenas quatro variáveis: *escopo, recursos, tempo e qualidade*. Às vezes pode ser possível adiantar o desenvolvimento de um módulo, por exemplo, se diminuirmos seu escopo, ou seja, a quantidade de coisas pela qual tal módulo é responsável. Sempre é possível também diminuir o tempo de realização de um projeto aumentando os recursos disponíveis para ele, mas cuidado: 100 horas de programação de um desenvolvedor não são iguais a uma hora de programação para um grupo de 100 desenvolvedores. Quanto mais desenvolvedores em um projeto mais variáveis relativas à comunicação entre eles entram em jogo[34]. Não é recomendável, porém, acelerar um projeto diminuindo os testes do sistema, comprometendo assim sua qualidade. Concentrando-se apenas nestas quatro variáveis é possível acelerar as decisões sobre a sequência de desenvolvimento.

Outra regra é *Move People Around*. O conhecimento não deve ser concentrado nas mãos de poucas pessoas. Envolva pessoas em diferentes projetos e troque-as de tempos em tempos.

A comunicação entre as pessoas e o envolvimento dos usuários (clientes) no desenvolvimento é extremamente importante. Não se deve, porém, perder mais tempo na comunicação dos resultados e na tomada de decisões do que no desenvolvimento em si. Por isto a XP propõe *Stand-up Meetings*, reuniões que devem acontecer todos os dias pela manhã, preferencialmente no ambiente de desenvolvimento. Estas reuniões informais devem tender a substituir integralmente todas as demais reuniões.

Uma das regras mais importantes do Planejamento em XP é *Fix XP when it breaks*, ou seja, se a metodologia não está funcionando, conserte a metodologia! Nem sempre pode ser possível seguir tudo o que uma determinada metodologia propõe e faz mais sentido mudar a metodologia do que querer acomodar, em todos os casos, o desenvolvimento de um sistema a regras imutáveis.

Projeto

Simplicidade é a palavra de ordem. Muitos de nós que já coordenamos ou desenvolvemos projetos conhecemos o método *KISS - Keep It Simple, Stupid* (literalmente: Mantenha a simplicidade, estúpido!). XP coloca a simplicidade como regra. Nada deve ser desenvolvido sem que seja necessário - evite prever necessidades futuras! Quando deparar-se no sistema com alguma estrutura complexa, substitua-a o mais rápido possível por outra mais simples. Estruturas complexas tendem a se tornar cada vez mais caras e difíceis de manter.

Escreva uma *System Metaphor*, ou metáfora do sistema, dando nomes simples e significativos aos objetos que o compõem. Estes nomes devem ser facilmente identificados pela sua função ou importância do sistema no negócio da empresa ou instituição à qual atendem. Na Daimler-Chrysler todos os objetos do sistema estão associados a nomes utilizados na linha de montagem dos automóveis. Faça isto de forma ingênua, mas clara. Veja no capítulo sobre UML os nomes que adotamos, por exemplo, para as classes que compõem o sistema.

As *User Stories* escritas na fase de planejamento devem agora ser transformadas em *CRC Cards* (CRC significa Classe, Responsabilidade e Colaboração). Cada *CRC Card* funciona como um "script" que representa as funções que cada módulo do sistema irá desempenhar e como ele irá se relacionar com outros módulos. Os usuários devem ser envolvidos nesta fase, e idealmente, lendo o *CRC Card* deve ser possível que uma pessoa consiga simular o papel do módulo que ele representa.

Quando surge uma dúvida sobre a forma como um determinado módulo ou função devem ser desenvolvidos, apela-se para *Spike Solutions* (Soluções Relâmpago). Antes de conhecermos o XP chamávamos as *Spike Solutions* de "Testes de Conceito" ou "Protótipos Prematuros". *Spike*

34 Um livro que aborda muito bem esta e outras questões é O Mítico Homem-Mês, de Frederick P. Brooks, com tradução de Cesar Brod para a editora Campus Elsevier

Solutions são usadas para testar uma ideia que pode ser utilizada pelo sistema, mas da qual não se tem a certeza exata de seu comportamento. Neste caso, desconsideramos o sistema como um todo e testamos a ideia isoladamente, com consciência de que ela pode não servir para nada depois e que todo o código desenvolvido para testá-la também pode não ser aproveitado para nada.

A última regra da fase de projeto é *Refactor Mercilessly*, que livremente traduzo por "Não se apaixone pelo seu código". Claro que o código pode ser reaproveitado, mas novas tecnologias e ideias surgem a toda hora e alguma coisa que fizemos no passado e que achamos sensacional pode deixar de sê-lo de uma hora para a outra. Esta regra não deve se contrapor, porém, a da simplicidade. Não fique querendo modificar tudo o que já funciona apenas porque alguém surgiu com uma nova ideia, por melhor que seja. Sempre que tiver dúvida, discuta com a equipe. Caso a nova idéia venha a simplificar algo complexo, reescreva seu código sem dó nem piedade.

Codificação

Como o usuário participou de todo o planejamento, especialmente como co-autor das *"User Stories"*, ele deve estar sempre disponível (*Customer Always Available*) durante o processo de codificação para sanar eventuais dúvidas que possam surgir e colaborar com sugestões. No caso do Sagu tínhamos na Univates, especialmente no desenvolvimento do sistema financeiro/contábil, a presença de uma pessoa de nossa contabilidade, que participa inclusive de treinamentos técnicos. No Gnuteca, as bibliotecárias da Univates estavam envolvidas no projeto desde a sua concepção e a primeira apresentação pública do projeto deu-se não para uma equipe técnica, mas para um grupo de bibliotecários.

A equipe de desenvolvimento deve concordar com os Coding Standards que serão utilizados, preferencialmente baseando-se em experiências de comprovado sucesso anterior. Nossa equipe tomou por base o "*PHP Coding Standards*" e os adaptou à nossa realidade e estilo de codificação.

Uma regra que sempre gera alguma surpresa (e até polêmica) é a *Unit Test*, que define que os testes devem ser definidos e codificados antes mesmo que o módulo que irá ser testado esteja escrito. Pensando-se nos testes que serão feitos já refina a codificação e elimina-se de início possíveis erros.

Os desenvolvedores devem trabalhar em pares (*Pair Programming*). O resultado prático é uma programação mais criativa, clara e menos sujeita a erros. Por incrível que possa parecer, a produtividade também é maior quando se tem dois programadores trabalhando ao mesmo tempo no mesmo código, além de ser mais fácil disseminar o conhecimento entre os projetos trocando-se os pares de tempos em tempos. Usamos esta técnica desde o início do desenvolvimento do Gnudata (infraestrutura em software livre para bases de dados estatístico/comparativas) e do Gnuteca.

Em XP, a propriedade do código é coletiva (*Collective Code Ownership*), assim todos compartilham do mesmo orgulho, e das mesmas críticas. A propriedade coletiva do código está totalmente alinhada com o desenvolvimento de software livre.

A otimização do código deve ser feita por último (*Optimize Last*) e apenas quando necessária. Não tente resolver gargalos antes que os mesmos apareçam. Lembre-se que o preço do hardware já está muito menor do que há dez anos, ao passo que as horas de bons desenvolvedores estão mais caras. Pense se não é mais barato aumentar o espaço em disco ou comprar processadores mais rápidos do que pagar um mês de um DBA para otimizar a base de dados.

A última regra da Codificação é *No Overtime*, sem horas-extra. Em XP o dia é de oito horas e a semana de quarenta horas. Desta forma é que os projetos devem ser dimensionados. Por mais que às vezes seja difícil afastar um programador de uma tarefa que ele faz com prazer, isto deve ser feito. A prática mostra que a produtividade diminui e a quantidade de erros aumenta quando se trabalha demais: existem limites que, antes de mais nada, são limites físicos. Mesmo que o programador

sinta-se compelido a continuar trabalhando ele deve abandonar a codificação e concentrar-se na documentação e no planejamento. Apresente para a equipe jogos como o Quake ou outros que possam ser jogados coletivamente.

Testes

Muitas das regras desta fase já foram definidas nas fases anteriores. Crie os testes mesmo antes de criar os módulos do sistema (*Unit Tests*) e efetivamente execute-os mesmo que eles não tenham condições de serem testados. Existem vários ambientes de testes que dependem da linguagem utilizada e uma série de métodos e recomendações. Adote uma delas ou crie uma metodologia que possa ser aplicada a seu sistema.

Quando aparecer um problema (*When a Bug is Found*) antes de resolvê-lo crie um teste que possa detectá-lo efetivamente, evitando assim que o mesmo tipo de problema apareça repetidas vezes. A importância disto se torna mais evidente a partir do momento que somos obrigados a resolver múltiplas vezes o mesmo tipo de problema.

Por fim, devem ser definidos junto com os usuários/clientes os *Acceptance Tests*, um conjunto de regras que irão dizer que um módulo está pronto para entrar em produção. O *Acceptance Test* é uma volta às "*Use Stories*", quando o usuário poderá comprovar que o módulo que foi desenvolvido corresponde à sua necessidade.

Miolo

Nos capítulos anteriores contei um pouco da história de alguns sistemas que foram desenvolvidos sob a minha coordenação na Univates e sobre alguma metodologia que acabamos adotando. A razão de eu ter usado esta sequência foi para poder dizer, agora, que toda a evolução de nossa metodologia de desenvolvimento se deu de forma bastante natural, orgânica. O aprendizado no desenvolvimento relâmpago do Sagu, o reaproveitamento de código no Gnudata e a conclusão de que deveríamos afinar a nossa metodologia de desenvolvimento foi algo com o que a equipe foi amadurecendo e concordando no decorrer do tempo. Sempre procuro em meus projetos cercar-me de pessoas que tenham uma grande independência, sejam autodidatas e, como costumamos falar em nossa equipe, tenham "sevirol" - que saibam mesmo buscar soluções criativas para os problemas que vão aparecendo e para as demandas vindas dos usuários. Como gestor de uma equipe de desenvolvimento procuro, antes de mais nada, atuar como um facilitador para que o ambiente seja harmônico e busco trazer para mim as decisões quando não há convergência em algum conflito, o que não é algo tão incomum. Por outro lado, trabalho muito usando um elemento da filosofia do Unix: *"No news is good news!"* - ou seja, se ninguém vem me falar nada, é sinal de que as coisas estão andando bem. Assim, procuro dar a maior independência possível aos desenvolvedores ao mesmo tempo em que eles têm a confiança de que podem buscar em mim o apoio em questões para as quais necessitam de novos elementos para uma determinada decisão ou futuro desenvolvimento.

O Miolo[35] foi um resultado bem claro desta evolução orgânica de nossa metodologia de desenvolvimento. A equipe já estava familiarizada com a linguagem PHP (mais adiante, no capítulo [XXX] volto a discutir sobre a decisão de ferramentas para o desenvolvimento) e havia concordado que o desenvolvimento utilizaria o paradigma da orientação a objetos usando a UML para a documentação. Vários elementos do Extreme Programming já estavam sendo utilizados por nós, mesmo antes de conhecermos formalmente esta metodologia. A ideia de que deveríamos ter um ambiente de desenvolvimento que contemplasse todas as coisas comuns a todos os programas, possibilitando o desenvolvimento mais rápido de soluções, era consenso. Logo após nosso primeiro workshop interno de transferência de tecnologia, que aconteceu em 2001 e que mencionei em *O Desenvolvimento do Sagu*, Vilson Gärtner e Thomas Sprietersbach começaram a desenvolver o Miolo. Mais adiante, Ely Edison Matos da Universidade Federal de Juiz de Fora tornou-se um importante colaborador, o que ilustra bem a questão anteriormente colocada que, no desenvolvimento de um software livre, pode-se contar com ajuda importante que não está diretamente abaixo de nossa própria gestão de desenvolvimento.

O nome Miolo é uma história à parte e, antes que ela perca-se no tempo, vou contá-la aqui. O Sagu é também o nome de uma popular sobremesa gaúcha, feita com vinho. Uma das brincadeiras da nossa equipe era a de que para se fazer um sagu melhor, tínhamos que usar um vinho melhor. A vinícola gaúcha Miolo é bastante famosa pela excelente qualidade de seus vinhos e, por isso, começamos a usar o nome-código de Miolo para o framework que desenvolvíamos. Com o tempo, criamos outras "desculpas" para o nome. Miolo é também aquilo que está por dentro, no meio, como o miolo de um pão. Ao menos aqui no sul chamamos também de "miolos" o que está dentro do nosso cérebro. Assim, o Miolo dá também a ideia de um kernel, algo central às aplicações, de certa forma invisível ao aspecto externo mas fundamentalmente responsável por toda a sua funcionalidade.

Com a sua evolução, o Miolo foi responsabilizando-se por várias funções comuns a boa parte das aplicações, dentre elas:

- Controles de interface com o usuário, escritos em PHP e renderizados em HTML

35 www.miolo.org.br

- Autenticação de usuários
- Controle de permissão de acesso
- Camada de abstração para acesso a bancos de dados
- Camada de persistência transparente de objetos
- Gerenciamento de sessões e estado
- Manutenção de logs
- Mecanismos de trace e debug
- Tratamento da página como um webform, no modelo event-driven
- Validação de entrada em formulários
- Customização de layout e temas, usando CSS
- Geração de arquivos PDF

Você não precisa entender o que cada uma destas funções significam, mas caso queira aprofundar-se no framework Miolo, a página do projeto oferece um bom tutorial sobre sua utilização. O fato é que, com o uso do Miolo, estimamos uma economia de mais de dois terços no tempo de desenvolvimento e um notável aumento de qualidade na escrita de código, vinda naturalmente da necessidade de um padrão de escrita exigido pelo framework.

As figuras abaixo exemplificam o código html necessário para a exibição de uma tabela e, a seguir, o código para a mesma finalidade utilizando o Miolo:

[Joice, já vou te dar um trabalhinho aqui... ao invés de usar as figuras, reescrever este código, pode ser?]

• EXEMPLO PRÁTICO - EM HTML:

[Opções de Cadastros]
[Vestibular]
[Vestibulandos]
[Inscrições]

```
<table width="100%" cellspacing="0" cellpadding="0" border="0">
 <tr>
  <td>
   <table class="themeBox" width="100%" cellspacing="0" cellpadding="0" border="0">
    <tr>
     <td class="boxTitle">[Opções de Cadastros]</td>
    </tr>
    <tr>
     <td><div class="boxContent"><a
href="index.php?module=vestibular&action=main:cadastros:novo&item=">[Vestibular]</a><br>
<a href="index.php?module=vestibular&action=main:cadastros:vestibulando&item=">[Vestibulandos]</a><br>
<a href="index.php?module=vestibular&action=main:cadastros:inscricao&item=">[Inscrições]</a><br>
     </div></td>
    </tr>
   </table>
  </td>
 </tr>
 <tr>
  <td><img src="/miolo/themes/vilson//shadow.jpg"></td>
</tr>
</table>
```

É importante salientar que a origem do Miolo está no pensamento de nossa equipe desde o princípio do desenvolvimento do Sagu, lá em 1999, ainda que ele não tivesse ainda evoluído nem para o seu formato atual e sequer tivesse recebido seu nome. Nós sabíamos que o Sagu tinha um prazo muito estreito para a sua entrada em produção e, por isso, muitas ideias sobre a elegância e portabilidade do código foram deixadas para a "versão 2", mas o germe do Miolo já estava lá. Tivesse o projeto Sagu, ou qualquer outro de nossos projetos na Univates e na Solis, sido iniciado entre cinco e dez anos depois, talvez o Miolo sequer existisse, já que outros frameworks[36] para o desenvolvimento em PHP ainda mais genéricos surgiram e passaram a contar com uma comunidade maior de desenvolvedores. Talvez por estar muito atrelado – não em sua capacidade, mas por sua história – ao desenvolvimento de aplicações de porte relativamente grande para instituições de ensino, o Miolo não teve a disseminação que poderia ter tido. Ainda assim, hoje, ele é a base de todo o desenvolvimento da Solis e teve um valor inestimável na aprendizagem de uma grande equipe, tanto em técnicas de orientação à objetos como metodologia de desenvolvimento e padronização de código.

36 Recomendo bastante aos que estão começando a explorar frameworks em PHP uma visita ao portal do CakePHP: http://cakephp.org

Plano Diretor de TI – Um modelo

Dentre as muitas coisas que aprendi com o professor Eloni Salvi está a máxima de que o trabalho de informática equivale a um trabalho de manutenção de esgoto. Enquanto tudo funciona bem, ninguém percebe o esforço que é necessário para a sua manutenção. Basta um cano estourar, porém, que todos começam a sentir o cheiro característico e a procurar pelos culpados. Por isso é importante, sempre, documentar de forma clara, inteligível por aqueles a quem nos reportamos, tudo o que fazemos em termos de gestão, desenvolvimento e manutenção daquilo que compõe os elementos de tecnologia da informação em uma empresa ou instituição. Mais do que isto, os tomadores de decisão devem saber, claramente, da importância da tecnologia da informação no crescimento e diferencial estratégico da empresa. Isto é mais fácil ou mais difícil dependendo da natureza da empresa. Durante todo o tempo em que a BrodTec prestou serviços para a Univates produzi um documento anual, depois transformado no Plano Diretor de Tecnologia da Informação (PDTI), que ilustrava os benefícios adquiridos com o uso e desenvolvimento de software livre, estado atual da área de TI, necessidades atuais e futuras relativas a usuários e infraestrutura. Este plano também servia como base para uma apresentação à reitoria e como um dos elementos para o planejamento estratégico da instituição.

Em outros trabalhos desenvolvidos pela BrodTec vários outros documentos foram desenvolvidos e fico feliz em observar que, em muitos casos, eles continuaram a ser usados como base evolutiva ou referência para outros documentos e, especialmente, ações posteriores.

Agora que o leitor está a par do que foi desenvolvido e aprendido durante a fase em que a BrodTec prestou serviços para a Univates, encerro este capítulo com algumas informações e recomendações sobre a elaboração de um PDTI, tomando como base o que foi desenvolvido para a Univates e já agradecendo a instituição, na figura de seu reitor, Professor Msc Ney José Lazzari, pela autorização de mais esta peça de conhecimento livre.

O Índice

O Índice já diz bastante coisa sobre o documento. Comento aqui, brevemente, cada um de seus componentes.

0. Histórico de Revisões	Este item deve ser obrigatório em todos os documentos que recebem contribuições de várias pessoas e passam por um processo de revisão e aprovação. Ele conta a história sucinta da evolução do documento e deve ser bem simples. Basta uma tabela que conste a alteração do documento, sua página, a razão da alteração (inclusão, deleção) e o autor desta alteração.
1. Resumo	Similar ao Resumo (ou Abstract) de um trabalho acadêmico, ele deve descrever em um parágrafo a finalidade do documento.
2. Introdução	No PDTI Univates este item é composto pelo capítulo “Ecologia do Conhecimento”, que funciona como um preâmbulo para o resgate histórico que vem no capítulo seguinte.
3. Histórico	É necessário posicionar o leitor da evolução até o momento,

	antes de partir para propostas para o futuro e pedidos de novos investimentos. No PDTI Univates ele é o capítulo “A Informática na Univates, a evolução para um Ecossistema”, composto de várias seções.
4. Economia	Qual foi a economia obtida com os investimentos em TI? Esta parte é especialmente importante para os responsáveis pelos investimentos e constitui-se em forte argumento quando da solicitação de novos recursos.
5. Plano Diretor de Tecnologia da Informação – PDTI	Aqui é a proposta para o futuro, sempre embasada naquilo que foi desenvolvido até o momento. Quanto mais fundamentados e documentados estiverem os itens anteriores, melhor a possibilidade de aprovação no que é proposto neste capítulo.
6. Conclusão	Mais um resumo, agora levando em consideração que o leitor (ou o público que assistiu a apresentação do documento) estão cientes de toda a informação até aqui exposta.
Todo List	Um documento como este é um ser vivo, sempre alimentado por ideias daqueles a quem é apresentado. Por isso ele deve, até seu derradeiro momento, contar com a lista de coisas que ainda devem fazer parte dele. Este item pode ser excluído das versões de apresentação e distribuição, mas deve ser mantido à parte como guia para futuras versões.

Repare que os itens 1 a 5 servem para nivelar e “ganhar” os leitores ou a audiência da apresentação. Eles constituem-se na defesa da ideia, no currículo da capacidade da equipe, nos sucessos alcançados. Eles devem ser escritos de forma factual e eventuais problemas e suas soluções – e também problemas sem soluções – devem ser expostos. Uma postura de transparência radical deve ser adotada e eles devem construir toda a base de justificativa para o que será proposto no item 5. Os textos a seguir foram extraídos diretamente da proposta de PDTI para a Univates no ano de 2004. Algumas informações foram editadas ou omitidas para preservar dados estratégicos ou confidenciais da instituição.

Resumo

Com o crescimento da Univates e a utilização cada vez maior da tecnologia da informação, é evidente a necessidade de que o uso desta tecnologia seja pensado de forma estratégica, alinhado ao planejamento estratégico da própria instituição. Este documento busca reunir a experiência adquirida na evolução dos sistemas de informática da Univates, no contato com outras instituições de ensino e empresas, na evolução da comunidade e das soluções em software livre e dos conceitos de “Conteúdo Aberto” (*Open Content*), e, especialmente, no contato com os usuários de nosso ambiente de TI e gestores com atribuições similares às minhas em outras instituições, formando a base de uma proposta para um Plano Diretor de Tecnologia da Informação (PDTI) para a Univates, alinhado ao seu Plano Estratégico.

Ecologia do Conhecimento

Os sistemas computacionais foram ao longo do tempo evoluindo de auxiliares operacionais para um complemento à evolução de formas do tratamento do conhecimento. Não é a toa que uma grande obsessão da Ciência da Computação é replicar o comportamento do nosso cérebro e nossos sutis métodos de decisão quando lidando com uma ampla gama de informações. Ainda assim, fora do alcance do que pode ser totalmente informatizado estão outros aspectos do conhecimento, como conversas e ideias. Esta proposta de PDTI busca tratar o conhecimento na forma de um ecossistema que integre dados, informações, documentos, conversas e ideias que compreendem – e circulam por – toda a Univates.

A ideia de uma Ecologia do Conhecimento foi proposta por Derek Keats e Madiny Darries, no documento "*Towards a Knowledge Ecology Strategy for the University of the Western Cape*", onde os autores produziram um "Estudo de Caso" da Univates que serviu como base para a adoção de software livre pela UWC. Outros documentos foram usados como referência para esta proposta, como o PDI da Unesp e a Pesquisa Nacional Sobre Gestão dos Sistemas de Informação e suas Tecnologias para Aprimoramento das IES (Lobo & Associados Consultoria). Dentre as conversas importantes, das quais surgiram boas idéias que não só foram aplicadas neste documento, mas na própria evolução da estrutura de informática da Univates, estão as que tive com o Prof. Dr. Imre Simon (projeto Tidia, FAPESP), Prof. Dr. Carlos Vogt (Presidente da FAPESP, Diretor do Laboratório de Jornalismo da Unicamp), Prof. Dr. Carlos Masiero (CIO, USP), Rubens Queiroz (Centro de Computação, Unicamp) e o Prof. Dr. Derek Keats (CIO, UWC).

> O conhecimento é a capacidade de pessoas e das comunidades de continuamente gerar e renovar, de modo a responder a novos desafios e oportunidades. As pessoas que desenvolvem conhecimentos podem ser inspiradas e apoiadas, mas dificilmente poderão ser "gerenciadas" como meras extensões de máquinas, como na Era Industrial.
>
> A ecologia do conhecimento é um campo interdisciplinar da prática gerencial que surgiu da confluência de estratégias de gestão, comunidades de prática, sistemas adaptativos complexos e gestão do conhecimento. É um corpo cada vez maior de conhecimento e práticas que focam na contínua melhoria das relações, ferramentas e métodos para criar, integrar, compartilhar, usar e alavancar conhecimento.
>
> Maria Alice Capocchi Ribeiro, Msc[37]

A Informática na Univates, a evolução para um Ecossistema

Não muito diferente de outras instituições, até 1999 a informática na Univates era usada de forma não integrada, com poucas estações de trabalho em rede utilizando aplicativos de escritório e um sistema administrativo que evoluiu de uma aplicação em Cobol para um ambiente baseado na plataforma MS-Access. Este ambiente, em 1999, serviu de base para a migração para o SAGU (Sistema Aberto de Gestão Unificada), que é usado até hoje na instituição.

37 http://www.rh.com.br/ler.php?cod=3762&org=2

Entre 1999 e 2004, o sucesso com o SAGU e com a adoção de ferramentas de produtividade livres como o Star/OpenOffice fez com que a Univates evoluísse para uma política de adoção preferencial e desenvolvimento integral de aplicações em software livre para a área administrativa e a ampliação gradual (ainda que bastante lenta) do uso de software livre também na área acadêmica.

A comparação com ferramentas e aplicativos em software proprietário que ainda mantemos na instituição nos fez comprovar ao longo do tempo que a economia (assunto sobre o qual falaremos mais no item Economia) é um efeito colateral importante de uma série de outros benefícios que temos com o uso do software livre, especialmente a flexibilidade, imunidade a vírus (e a indisponibilidade causada por eles) e a independência tecnológica.

A disponibilização de sistemas para os usuários da Univates através da rede, com a concentração de serviços na Intranet, além de criar um ambiente central-virtual compartilhado por todos, potencializou a cumplicidade dos usuários com o desenvolvimento de novos sistemas e serviços, ao mesmo tempo em que os tornou mais exigentes e dependentes da nossa estrutura de informática, que deve se tornar cada vez mais disponível, tolerante a falhas e eficiente para suprir as demandas que impactam a produtividade dos usuários e a eficiência da instituição.

Ainda em 2001 optamos por criar um ambiente de software onde pudéssemos concentrar os requisitos básicos de qualquer sistema que viéssemos a desenvolver em uma camada que pudesse servir e ser reaproveitada, gerando não só economia na escrita de código, mas permitindo que aspectos relativos à distribuição de dados, controle de acessos, segurança e outros não precisassem mais ser “programados” nas aplicações, mas fossem inerentes ao próprio ambiente. A este ambiente demos o nome de Miolo, e a primeira aplicação desenvolvida com ele foi o Gnuteca, que é responsável pela gestão de acervo, empréstimos e outros aspectos relacionados aos serviços de nossa biblioteca. O Gnuteca entrou em atividade em fevereiro de 2002.

A partir daí, todo o novo desenvolvimento de aplicações passou a ter o Miolo como base, a ponto de, em um determinado momento, os usuários passarem a confundir as aplicações disponibilizadas na Intranet com o próprio Miolo. Hoje, nossos usuários têm à sua disposição na Intranet serviços como o registro de ponto, solicitações de serviços de apoio, compras e chamados técnicos, controle de documentação na norma ISO 9002, reservas de salas e veículos, entre muitos outros. Isto também faz com que, com o uso intensivo da Intranet para o acesso aos serviços, a mesma sirva como um meio ideal de divulgação de informações aos funcionários da Univates, evoluindo para um ponto de encontro virtual.

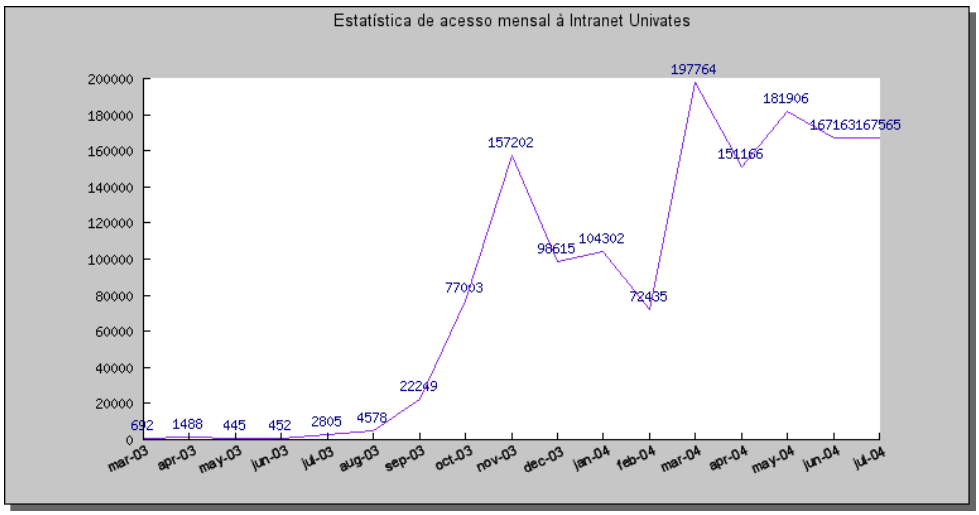


Figura 1 Evolução do acesso à Intranet

Os serviços aos alunos através da Internet também foram se intensificando, especialmente através da integração a sistemas como o Sagu e o Gnuteca. Hoje os alunos da Univates podem usar a Internet ao se inscreverem para o vestibular, administrarem sua matrícula e alterações nos períodos determinados, localizar sua sala de aula, verificar sua situação financeira, imprimir boletos para pagamentos bancários, fazer a inscrição em eventos, reservar e renovar material emprestado da biblioteca, verificar suas notas, entre outros. Os portais Inter e Intranet foram ambos desenvolvidos com o Fred, um sistema para a gestão de conteúdos que também tem o Miolo como base, o que facilita a integração de soluções e a oferta de serviços através dos portais. Apenas como exemplo, o sistema de nossa estação meteorológica, também em software livre, disponibiliza dados históricos e em tempo real sobre o clima de Lajeado através de nosso portal Internet.

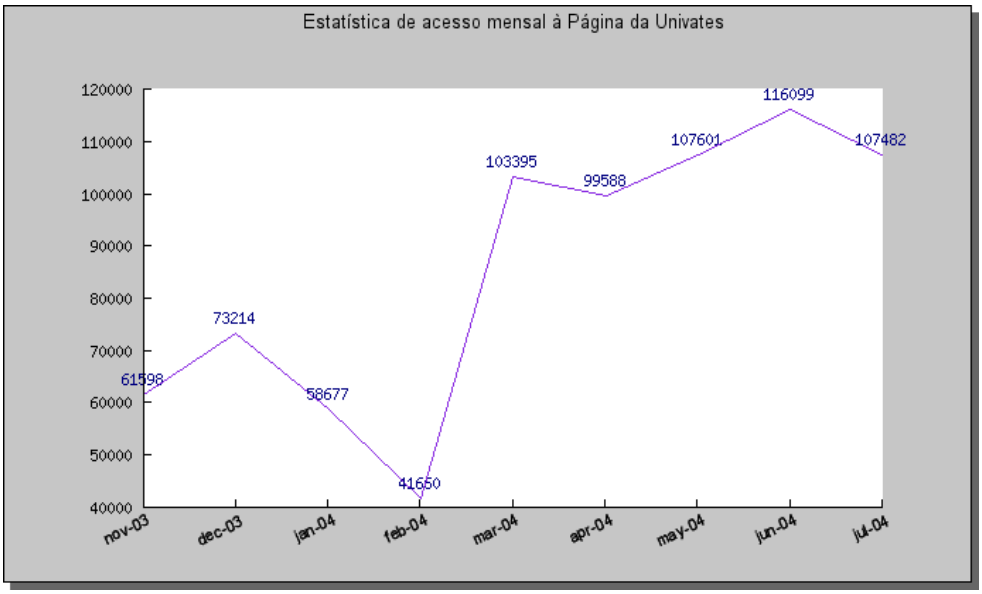


Figura 2 Evolução do acesso à Internet

Alunos, funcionários e outros colaboradores da instituição ainda têm acesso a serviços tradicionais de e-Mail, listas de discussão (quase cinquenta, reunindo turmas, grupos de interesse de alunos e funcionários, entre outras), ambientes de criação coletiva de documentos e de formação dinâmica de bases de conhecimento (Rau-Tu). Alguns professores já estão utilizando o WebDiário para o registro de freqüência e notas dos alunos. Todos estes serviços estão convergindo para um ambiente único, e são acessados de forma segura por usuários através de uma arquitetura de senhas fortes[38]. Em fase de testes e implantação ainda estão um sistema de agenda compartilhada, um sistema de armazenamento eletrônico de artigos, teses e dissertações e outros.

Acompanhamento de chamados técnicos

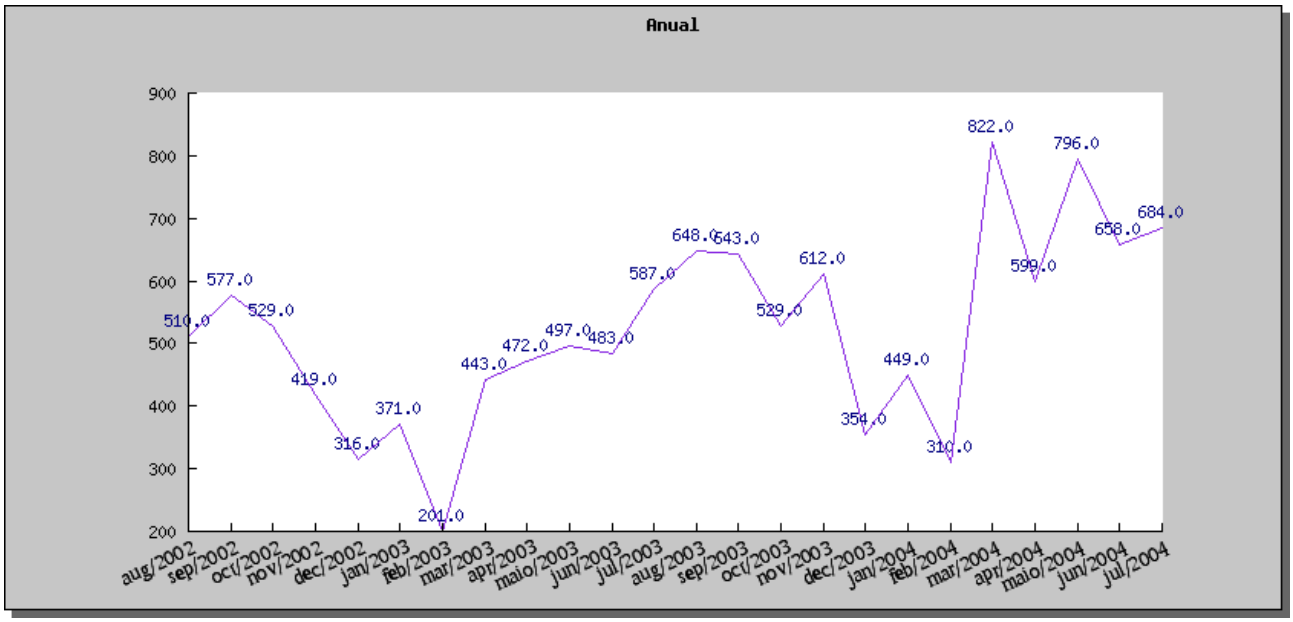


Figura 3 Evolução dos chamados técnicos, entre agosto de 2002 e julho de 2003

Em final de julho de 2002, disponibilizamos na Intranet o sistema Scotty, através do qual são feitos os registros de chamados técnicos e seu acompanhamento. Através da análise destes chamados e sua recorrência aprimoramos nossos processos de manutenção corretiva, preventiva e preditiva, especialmente com a criação de uma “imagem de instalação” para as máquinas usadas pelos funcionários e por um mecanismo de atualização e instalação automática de softwares: uma vez detectada uma falha em um programa, a correção ou a atualização é testada e instalada automaticamente em todas as máquinas. Desta forma, cada vez que um problema é detectado, ele afeta um número menor de pessoas, já que a correção é feita logo a partir da primeira ocorrência. Sistemas de suporte remoto também foram implantados, minimizando a necessidade de deslocamento da equipe. A evolução dos chamados nos permite ainda detectar aonde há falhas de treinamento de pessoal (tanto de usuários quanto da equipe de suporte) e definir treinamentos. No segundo semestre de 2004 estaremos ministrando cursos internos em Gnu/Linux Básico, OpenOffice, Sagu e Agata Report, além de vários cursos voltados ao suporte interno, nos quais, além da equipe do CPD, estão sendo envolvidos os monitores dos laboratórios acadêmicos. Outras vezes, a recorrência de um certo tipo de chamado técnico também aponta para a falta de

38 Processo de unificação de logins e senhas com o OpenLdap iniciado em meados de 2004

documentação de algum de nossos sistemas ou procedimentos. Isto nos levou a, além de aprimorar bastante os manuais do Sagu e do Gnuteca, disponíveis na Intranet, criar pequenas cartilhas de cada um dos sistemas disponíveis, apresentando-os aos usuários. A este tipo de ações que visam a diminuir o número de chamados técnicos a partir da análise do comportamento dos usuários, damos o nome de “manutenção proativa”.

Além da simples evolução do número de chamados, o Scotty ainda nos permite verificar a qualidade dos serviços de suporte, através da comparação do número dos chamados com o seu efetivo atendimento dentro dos prazos estabelecidos pelos próprios usuários, e a partir de maio de 2004 também com o registro da opinião do usuário sobre o atendimento (que é manifestada de forma voluntária).

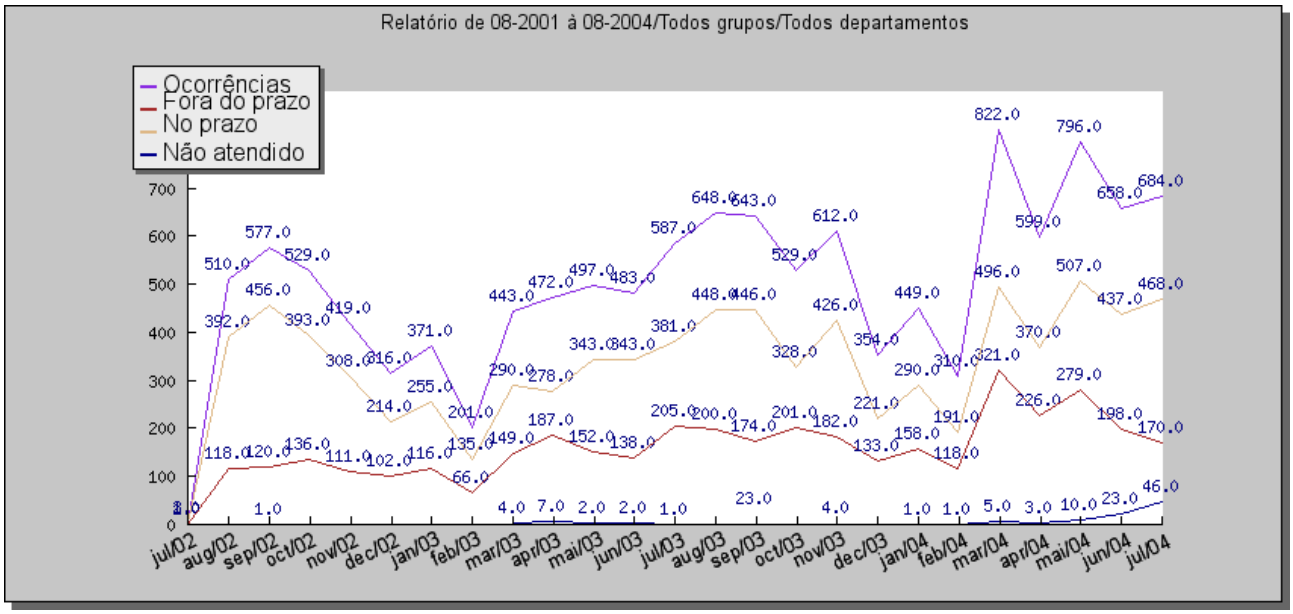


Figura 4 Evolução do atendimento dos chamados técnicos

Ainda que tenhamos conseguido manter um número de usuários crescente em relação ao número de analistas, o gráfico acima mostra que, mesmo com ações de manutenção preditiva, há uma tendência ao crescimento dos chamados não atendidos ao final de cada mês (a partir de março de 2004), o que mostra que o número de analistas está subdimensionado.

***Nota**: Repare que os dados factuais, comprovados, já servem de argumento para a solicitação de um aumento da equipe.*

Software livre na Academia

Na área acadêmica tivemos um aumento da simpatia dos docentes pelo software livre especialmente após a contratação do Prof. Marcelo Malheiros em agosto de 2002, oriundo da Unicamp, um entusiasta e desenvolvedor de sistemas em software livre que são utilizados na Unicamp, Univates e várias outras instituições de ensino. Com o Prof. Malheiros intensificamos o uso do ambiente Teleduc na nossa experiência em educação à distância, e mesmo em cursos presenciais. Mais recentemente, com a contratação da Prof.a Maria Abadia, em novembro de 2003, passamos a oferecer aos docentes um apoio mais efetivo na

busca de alternativas de ferramentas livres para a substituição de softwares proprietários.

Segundo o Prof. Malheiros: *"Ainda há muita coisa que pode ser feita em todos os cursos oferecidos pela instituição, como a adoção de ferramentas de matemática e estatística livres, programas educativos, etc. Só que o trabalho é bem maior, pois estas soluções precisam estar muito bem estruturadas, documentadas e apoiadas internamente na instituição para serem aceitas em substituição aos programas não livres que costumam ser utilizados."*

Outros aspectos bastante importantes na adoção de software livre na Univates são o incentivo aos grupos de usuários Gnurias e Taquarix e a disponibilização de um ambiente onde a comunidade pode hospedar projetos em software livre, o portal Código Livre.

Os grupos de usuários se beneficiam da estrutura da instituição para organizar encontros e eventos envolvendo nossos alunos e buscando colocá-los em contato com vários expoentes da comunidade de software livre. Um destes eventos é a "Semana do Pingüim", que acontece uma vez a cada semestre e busca trazer novidades e mostrar a nossos alunos as oportunidades que existem com o software livre. O grupo Gnurias ainda trabalha com escolas públicas, buscando despertar em alunos do ensino médio e fundamental uma "curiosidade tecnológica", que os provoca não só a buscar o acesso à tecnologia, atuando como agentes de mudança em suas comunidades, mas também permitir que eles desenvolvam capacidades que possam servir de apoio ao seu aprendizado e diferencial quando tiverem de enfrentar o mercado de trabalho. Parte deste trabalho das Gnurias inclui expor os alunos ao ambiente da Univates, em oficinas que são realizadas nos finais de semana nos laboratórios da instituição, despertando neles a vontade e ambição pelo ensino superior, pela aprendizagem contínua.

Escola Aberta e o grupo Gnurias

Graças ao grupo Gnurias várias crianças que ficavam em casa sem ter o que fazer, agora estão tendo a oportunidade de aprender a lidar com o COMPUTADOR, ter uma noção do que é um COMPUTADOR, porque hoje em dia a TECNOLOGIA está muito avançada, cada dia estão sendo criadas coisas novas, como por exemplo: computadores cada vez mais potentes com a capacidade de criar, fazer, inventar e descobrir coisas novas, coisas que os nossos antepassados nunca *imaginacem* (sic) que poderiam ser inventadas!

Para algumas crianças,apenas tocar em um computador já é maravilhoso. Neste projeto ESCOLA ABERTA, nós crianças estamos tendo a oportunidade, de mostrar o talento que temos em todas oficinas. Estamos aprendendo muito. Eu, já sabia lidar com um computador, mas resolvi me aprofundar mais, aprender mais!

Eu não poderia deixar de agradecer ao GRUPO GNURIAS tudo o que estou aprendendo a mais, mas não agradecer só por mim, mas por todos que estão tendo esta oportunidade!

A vocês o MEU MUITO OBRIGADO!

Vanessa Helena Barbon Corbelini
12 anos, 6ª série
(texto copiado e colado do original digitado pela Vanessa em uma oficina na Univates)

O portal Código Livre surgiu da necessidade de termos maior controle sobre os programas que já eram desenvolvidos na Univates, com a adoção de ferramentas para o controle de versões, controle de tarefas e sítio web com informações específicas sobre cada um dos projetos. Para isto, implantamos no final do ano 2000 o ambiente "SourceForge", um

conjunto de ferramentas que se destinam especificamente à gestão do desenvolvimento cooperativo de software. Nesta época, passamos a receber a primeira contribuição externa, voluntária, aos softwares que desenvolvíamos na Univates, de Ericson Smith (Estados Unidos) e Uwe Steinmann (Alemanha) para a psLib, uma biblioteca para a geração de relatórios impressos usada até hoje em nossos sistemas. Logo, a visibilidade de nosso sistema de gestão de projetos começou a crescer em função do acesso da comunidade ao código dos sistemas que desenvolvíamos, e logo recebemos pedidos de hospedagem de outros projetos em software livre em nosso ambiente. No início de 2001 criamos o portal Código Aberto, destinado a hospedar projetos em software livre da comunidade brasileira, e que passou a servir como uma ampla fonte de referência e aprendizagem em cima do trabalho de outros desenvolvedores. Em 2002, a pedido de Richard Stalmann, presidente da *Free Software Foundation*, mudamos o nome do portal para Código Livre. Em meados de 2003, com o Código Livre consumindo boa parte de nossos recursos de rede com mais de 200 projetos hospedados e mais de 2.000 desenvolvedores ativos, firmamos um acordo com a Unicamp, que passou a hospedar o portal, que é hoje mantido conjuntamente pela Univates e Unicamp. Hoje são mais de 840 projetos hospedados, com mais de 6.500 desenvolvedores cadastrados.

O Código Livre acabou servindo como um centro de exposição de nossos projetos e outros da comunidade, sendo onde as contribuições de código entre um projeto e outro acontecem. Mais adiante, em “Aproveitamento de Código e Colaborações”, relatamos tais contribuições, estimando o total de horas que economizamos no desenvolvimento de nossos próprios sistemas graças ao trabalho de outros, que também puderam se beneficiar de nosso trabalho.

Outra iniciativa importante e hoje nacionalmente reconhecida são os SDSL (Seminários de Desenvolvimento de Software Livre), cuja origem está na primeira edição do “Sagu Workshop” em julho de 2000, onde buscávamos novas idéias para os nossos sistemas ao mesmo tempo em que transferíamos aquilo que desenvolvemos para os participantes do evento. Como acabamos desenvolvendo novos produtos além do Sagu, o III Sagu Workshop, em julho de 2002, aconteceu em paralelo ao I Seminário Univates de Desenvolvimento de Software Livre. O evento motivou o interesse da Unisinos e Unicamp (já nossa parceira no portal Código Livre), e o IV Sagu Workshop aconteceu em julho de 2003, junto ao I SDSL, na Unisinos, em São Leopoldo (a partir daí, o SAGU Workshop deixou de existir de forma isolada e estamos incentivando um encontro de usuários do Sagu e do Gnuteca a partir dos próximos SDSL). O II SDSL, que aconteceu na Unicamp, Campinas, em dezembro de 2003, já contou também com a organização do ISTEC (Ibero-American Science and Technology Education Consortium). O III SDSL marcou a volta do evento à Univates, com um público pagante de 37 pessoas de 10 Estados brasileiros, um portal próprio na Internet (www.sdsl.org.br) e a participação da UnB na organização e que sediará também o IV SDSL em dezembro deste ano, em Brasília.

Software Livre na bagagem

O trabalho pioneiro com software livre na Univates acabou proporcionando muitas oportunidades de exposição deste trabalho em vários lugares do Brasil e do mundo, através de intercâmbios formais (três alunos que trabalhavam no CPD, e hoje na Solis, estiveram em Gelsenkirchen: Vilson Gärtner, Marcone Luis Theisen e Daniel Afonso Heisler) e da participação em eventos no Peru, Chile, Uruguai, Austrália, Estados Unidos, Costa Rica, África do Sul e outros.

Josi Petter, acadêmica do curso de História, teve a oportunidade de conhecer Machu Pichu

no Peru em novembro de 2003 graças a um convite da Unesco para apresentar seu trabalho com o grupo Gnurias na “I Conferencia Latinoamericana y del Caribe sobre desarrollo y uso del Software Libre” em Cuzco. A Univates contribuiu com a ajuda de custo para a estadia, enquanto a Unesco arcou com as demais despesas.

Joice Käfer, acadêmica do curso de Engenharia da Computação esteve na Austrália em janeiro de 2004, patrocinada pela Solis, Univates, e em grande parte pela Linux International Austrália, como “keynote speaker” da Miniconferência Educacional, durante a Linux.Conf.Au, onde, além de fazer uma palestra sobre o uso do software livre no apoio ao ensino de crianças, com base na experiência do grupo Gnurias, ainda pôde estabelecer contato com muitos desenvolvedores e trazer novas idéias, especialmente para a estrutura do portal Univates.

Na Austrália pude aprimorar meus conhecimentos em outra língua (inglês). Em todas as viagens conheci diversas pessoas importantes, com as quais troquei idéias e, nesta troca, sempre se aprende algo. Certamente me tornei mais independente: na Austrália tive que me virar sozinha... e foi bastante difícil esta sensação.

Ainda na viagem de ida para a Austrália, eu chorava de saudade, medo e preocupação. Afinal, era a minha primeira viagem internacional. Uma mulher sentada ao meu lado, colocou a mão no meu ombro e não falou nada, só sorriu como quem diz: "Vai passar" e ficou encantada que na minha idade eu já estava indo sozinha para um lugar tão distante e para falar sobre um trabalho tão legal. Eu ia apresentar os projetos das Gnurias com o uso do Software Livre no ensino fundamental e médio. A mulher era parapelégica e estava indo para o Uruguai, fazer um tratamento para tentar voltar a caminhar. Na hora, dei-me conta que eu estava chorando por problemas tão pequenos perto do dela, que estava muito confiante. Na juventude, ela tinha lutado pela causa feminista e se identificou com o nosso trabalho. Ela me disse que, uma viagem destas, na minha idade, equivalia a uma Universidade inteira. Concordo com ela.

As viagens e a participação em eventos me ajudaram principalmente na questão de falar em público. Cada vez mais os medos vão sumindo...

Joice Käfer

João Alex Fritsch, formado em Administração pela Univates, pós-graduando em Cooperativismo e presidente da Solis, esteve, sob patrocínio da Unesco, no Uruguai e no Chile participando de seminários de transferência de tecnologia sobre o Gnuteca, o que acabou, além de trazer mais idéias para o sistema, fomentando um grupo que hoje mantém a sua versão em espanhol.

Ambiente de rede

Hoje, todos os usuários de nosso ambiente de informática têm seus computadores integrados à nossa estrutura de rede, que conecta praticamente todos os prédios da Univates através de uma estrutura de fibra ótica. A administração da rede é centralizada no prédio 1, onde se concentram os servidores e sistemas de gestão, todos baseados em software livre. Os investimentos em nossa rede e infra-estrutura acompanharam com atraso a expansão de nossos sistemas, de nosso parque instalado e da necessidade de nossos usuários. Historicamente, desde a disponibilização do acesso à Internet, em 1996, temos constantemente utilizado mecanismos de limitação de tráfego, armazenagem de arquivos e outros. Os usuários possuem cotas máximas de armazenamento de informações em nossos

servidores, e o acesso à internet é monitorado e bloqueado de acordo com o sítio acessado e tipo de conteúdo. Hoje estamos revendo esta questão, e questionando se o excesso de controle não está sendo mais “caro” do que um aumento de recursos para os usuários e uma menor rigidez no controle de acesso – ainda que em recente pesquisa da Lobo & Associados pudemos comprovar que a grande maioria das instituições de ensino implementa controles de acesso à rede pelos mais diversos motivos.

Temos uma separação de nossos ambientes administrativo, acadêmico e conteúdo web público, com replicação de dados para a maior garantia de segurança em caso de um eventual ataque. Ainda são necessários muitos investimentos na replicação da estrutura onde temos pontos individuais que, em caso de falhas, indisponibilizam o acesso aos sistemas – isto ocorre tanto em servidores, como ativos de rede e interconexão entre prédios.

A Solis

Já em meados de 2000, logo após a implantação do SAGU, a equipe do CPD Univates passou a ser requisitada para prestar serviços de consultoria, desenvolvimento, implantação e integração de sistemas em software livre. Estes serviços, ainda que trouxessem recursos para a instituição, estavam fora do foco da mesma, que nunca imaginou se tornar uma software-house. Ainda assim, estava evidente uma oportunidade de negócios que podia ser potencializada para os alunos da instituição, aliada ao seu inerente compromisso e missão com o desenvolvimento regional. Após algumas discussões sobre a melhor forma de se aproveitar esta oportunidade, a Univates incentivou a equipe do CPD, formada quase totalmente por alunos da instituição, a formar uma cooperativa que, além de assumir os serviços de desenvolvimento e suporte da Univates, pudesse buscar novas oportunidades no mercado. A Solis, cooperativa de desenvolvimento e integração de soluções livres, foi formada em janeiro de 2003, e em setembro de 2003 a grande parte dos serviços realizados pelo CPD da Univates foi terceirizada para a cooperativa, que começou com 20 cooperados e hoje conta com mais de 30.

***Nota**: Mais recentemente, mesmo continuando a usar os serviços da Solis, a Univates optou por reaparelhar seu desenvolvimento interno e desenvolveu um novo sistema acadêmico e administrativo, ainda baseado em softwares livres, chamado Alfa. A Solis conta hoje com mais de 50 colaboradores.*

A Tecnologia do Ecossistema

O próprio histórico da evolução da informática na Univates, descrito acima, mostra que ela não pode ser pensada de forma destacada de uma série de outros elementos. Ela deve ser vista como a estrutura tecnológica de um ambiente de troca permanente de informações e decisões, e involucrar em uma proposta de solução e evolução tudo aquilo que é parte de um ambiente que pode ser tecnologicamente definido, e tudo aquilo que faz uso deste ambiente, mas que não pode ser tecnologicamente definido – e que muitas vezes têm ações que fogem ao controle de qualquer previsão ou mesmo descrição.

O que é Ecossistema?

Ecossistema (de *ecologycal system*, sistema ecológico) designa o conjunto formado por todos os organismos vivos que habitam numa determinada área, pelas condições ambientais dessa área, e pelas relações entre as diversas populações e entre estas e o meio.

http://pt.wikipedia.org/wiki/Ecossistema

A evolução da informática na Univates, especialmente com o uso do software livre, causou impactos que estavam fora do escopo de um planejamento inicial, mas que retornam beneficamente tanto na forma de reconhecimento e destaque ao nosso trabalho, como na forma mais efetiva e mensurável de contribuições de código para nossos sistemas, passando pela possibilidade de exposição de nossos técnicos (hoje da Solis) a um universo de outros desenvolvimentos e pessoas, com viagens apoiadas e patrocinadas não só pela Univates, mas pela Unesco, Linux International, HP, etc...

Já em sua missão, a Univates define o ecossistema em que se encontra: *"Gerar, mediar e difundir o conhecimento técnico-científico e humanístico, considerando as especificidades e as necessidades da realidade regional, inseridas no contexto universal, com vistas à expansão contínua e equilibrada da qualidade de vida."*

Assim, alinhado ao Plano Estratégico, o Plano de Desenvolvimento de Informática, pensado na forma de base a uma ecologia do conhecimento, deve involucrar em sua proposta todos os elementos da missão da Univates: o conhecimento em si, a região no contexto universal, e os atores em seus vários papéis (alunos, professores, funcionários, membros da comunidade). E os atores, vistos como usuários dos sistemas de informática e geradores das demandas para a evolução destes sistemas, também são parte deles, e centro de um ecossistema onde não serão centro em todos os momentos.

Economia

Ainda que não seja o aspecto mais importante para a adoção de uma ou outra tecnologia, há que se convir que recursos limitados para investimentos exigem a constante necessidade de se trabalhar com soluções criativas e com o menor custo possível. A adoção do software livre, especialmente na administração da Univates, permitiu que adaptássemos e desenvolvêssemos soluções com domínio da tecnologia, além da imediata economia em função de não haver a necessidade de se pagar por licenças de uso.

A seguir, o documento elaborado para a Univates apresenta o levantamento de todo o parque instalado e a economia feita ao não se utilizar softwares proprietários. A tabela abaixo apresenta um exemplo do que pode ser feito, com os números omitidos, já que os mesmo podem variar bastante. Para a economia vinda do uso do Sagu, Gnuteca, Teleduc, Agata Report e outros foram consideradas, na época, as opções equivalentes mais baratas, mesmo que não atendessem a totalidade das necessidades da Univates.

Sumário da Economia em Licenças

Descrição	*Total Economizado*
n desktops rodando exclusivamente softwares livres	
m desktops com Sistema Operacional proprietário e OpenOffice	
Sobrevida de x computadores que seriam obsoletos com software proprietário	
Uso de software livre nos servidores	
Uso de bases de dados livres	
Não necessidade de licenças de acesso às bases de dados	
Uso do Sagu	
Uso do Gnuteca	
Uso do Teleduc	
Uso do Agata Report	
Uso de estrutura em software livre para correio eletrônico	
Total	R$ 0,00

Não está considerada aqui a economia feita com a utilização de softwares livres para o ensino, que cresce consideravelmente e ainda necessita ser devidamente avaliada.

Ainda que cada tipo de software proprietário tenha sua política de atualização a prática tem mostrado que a cada dois anos os investimentos em software se repetem.

Aproveitamento de código e colaborações

Mais difícil de mensurar é a economia indireta obtida a partir da própria cultura de aproveitamento de código e colaboração que existe na comunidade de software livre. Com softwares proprietários, como nós mesmos temos o exemplo a partir de nosso ERP, ficamos na maior parte do tempo à mercê de implementações ou soluções vindas de um fornecedor único (ainda que, até neste caso, tenhamos buscado alternativas). No quadro abaixo, ilustramos os principais códigos alheios que utilizamos na construção dos nossos sistemas, assim como colaborações que recebemos.

Código da Comunidade	*O que faz*	*Onde foi utilizado*	*Estimativa de economia de tempo (um desenvolvedor)*
PEAR	Camada de abstração	Agata Report[39]	2 meses

Código da Comunidade	*O que faz*	*Onde foi utilizado*	*Estimativa de economia de tempo (um desenvolvedor)*
	de base de dados, permitindo a conexão com diferentes sistemas de bases de dados		
JPGraph	Biblioteca e aplicativos para a criação dinâmica de Gráficos	Agata Report, Scotty[40], GnuData[41], Fred[42]	6 meses
PHPDocWriter	Gera documentos no formato OpenOffice (XML) dinamicamente	Agata Report	12 meses
FPDF	Gera documentos no formato PDF dinamicamente	Agata Report	6 meses
Yala	Autenticação OpenLdap diretamente através do PHP	Internet, solicitação de conta de e-Mail	½ mês
HTMLArea	Editor gráfico em JavaScript para a autoria web	Fred	2 meses
phpNuke	Gestão de conteúdo web	Os temas (visual) do PhpNuke foram utilizados ou usados como base em vários ambientes web	1 mês
Phpopenchat	Salas de bate-papo para a web (não liberado para uso geral por limitação de recursos)	Fred	1 mês
WebMiau	Aproveitado o código para nosso WebMail	Fred	3 meses
Gettext	Estrutura que permite	Todos	-

39 O Agata Report (www.agata.org.br) é um sistema para a extração de informações e geração de relatórios a partir da conexão direta com bases de dados. Na Univates ele é utilizado para a geração de diversos relatórios a partir dos sistemas Sagu, Gnuteca e Microsiga

40 O Scotty (http://scotty.codigolivre.org.br) é o sistema de gestão de chamados técnicos e ordens de serviços utilizado através da Intranet da Univates

41 O Gnudata (http://www.bdr.Univates.br) é a base para a implementação de nosso Banco de Dados Regional na internet

42 O Fred (http://fred.codigolivre.org.br) é o gestor de conteúdo Web utilizado para os portais Internet e Intranet da Univates

Código da Comunidade	***O que faz***	***Onde foi utilizado***	***Estimativa de economia de tempo (um desenvolvedor)***
	que nossos sistemas sejam traduzidos em outras línguas		
ImageMagik	Redimensionamento dinâmico de imagens para a exibição nos portais web	Fred	3 meses
Twiki	Algoritmos, formatação e exemplo de código	MioloWiki[43]	1 mês
DocBook	Ferramenta para a geração de textos em múltiplos formatos	MioloWiki	6 meses
Diff	Classe para exibir diferenças entre versões de um mesmo texto	MioloWiki	1 mês
Total aproximado			44 meses

Colaborações da Comunidade(diretamente em nossos produtos)	***O que faz***	***Onde foi utilizado***	***Estimativa de economia de tempo (um desenvolvedor, considerado apenas o tempo que serviu diretamente à Univates)[44]***
Bruno Depero (Itália)	Testes e melhorias na conexão com bases de dados distintas, tradução para o italiano	Agata Report	1 mês
Jeffrey Buchbinder (França)	Uso de múltiplas buscas secundárias (subqueries) e melhorias no código	Agata Report	½ mês

43 O MioloWiki (http://wiki.solis.coop.br) é utilizado na criação cooperativa de documentos, e na Intranet Univates é a base para os manuais de usuário do Sagu, Gnuteca e outros

44 Note-se que, mesmo que uma colaboração na forma de uma tradução para uma outra língua (ou conexão com bases de dados diversas) possa não ter influenciado diretamente o uso de um programa pela Univates, isto acaba auxiliando no aumento da visibilidade de tal programa e permitindo o seu uso por mais pessoas, que posteriormente podem colaborar em funcionalidades que utilizamos aqui.

Colaborações da Comunidade(diretamente em nossos produtos)	***O que faz***	***Onde foi utilizado***	***Estimativa de economia de tempo (um desenvolvedor, considerado apenas o tempo que serviu diretamente à Univates)***
	fonte usado na mesclagem de documentos (merge)		
Thomas Sprietersbach (Alemanha, Brasil)	Melhoras no sistema de tradução e tradução para o alemão	Agata Report	¼ mês
Luciano Stein (Brasil)	Implementações de funcionalidades para a conexão com bases SQL Server e Oracle e testes com o Windows	Agata Report	1 mês
Johannes Schill (França)	Tradução para o Sueco	Agata Report	-
Brad McCrorey	Melhorias na geração de documentos PostScript	Agata Report	¼ mês
Christian Etuy (França)	Tradução para o Francês	Agata Report	-
Tarique Nagpur e Girish Nair (Índia)	Versão para o MySQL e testes	Agata Report	¼ mês
Steph Fox (Estados Unidos)	Auxílio com o PhpGtk e com a tradução para o inglês	Agata Report	1 mês
Frank Krommann (Estados Unidos)	Ajuda na conexão com a base de dados FrontBase	Agata Report	-
Andrew M. Yochum	Ajuda na conexão com a base de dados Sybase	Agata Report	-
Flavio Jose Fonseca de Souza (Brasil)	Ajuda na conexão com a base de dados mssql	Agata Report	-
Brice Vissiere (Brasil)	Solução de problema na quebra de coluna	Agata Report	¼ mês

Colaborações da Comunidade(diretamente em nossos produtos)	***O que faz***	***Onde foi utilizado***	***Estimativa de economia de tempo (um desenvolvedor, considerado apenas o tempo que serviu diretamente à Univates)***
Eduardo Fernandes (Brasil)	Ajuda na conexão com a base de dados SQL Server	Agata Report	-
Tomas Cox (Brasil)	Auxílio com o uso do PEAR DB	Agata Report	½ mês
Lucas Di Pentima (Espanha)	Tradução para o Espanhol	Agata Report	-
Yannick Warnier	Documentação e correção de erros	Tulip[45]	¼ mês
Unicamp (vários)	Modelagem da base de dados do sistema de ramais telefônicos e inscrições em eventos via web	Fred	1 mês
Uwe Steinman(Alemanha)	Várias contribuições na criação da PsLib	PsLib[46]	1 mês
Ericson C. Smith (Estados Unidos)	Conversão de nossa biblioteca psLib para classes em PHP, o que permitiu seu uso em mais programas	PsLib	¼ mês
Jay Haugen (Alemanha)	Alinhamento de texto e suporte a documentos colorido	PsLib	¼ mês
Joel Leon (Estados Unidos) José Paulo B. da Silva e outros	Adição de novas funcionalidades, solução de problemas diversos, versão na linguagem C e criação de extensões para outras linguagens	PsLib	2 meses

45 O Tulip (http://tulip.solis.coop.br) é um editor de código em PHP, criado para economizar tempo na escrita de programas nesta que é a linguagem de programação mais utilizada na Univates.

46 A psLib (http://pslib.codigolivre.org.br) é uma biblioteca que permite a geração dinâmica de documentos, utilizada em praticamente todos os programas desenvolvidos pela Univates/Solis

Colaborações da Comunidade(diretamente em nossos produtos)	***O que faz***	***Onde foi utilizado***	***Estimativa de economia de tempo (um desenvolvedor, considerado apenas o tempo que serviu diretamente à Univates)***
	(Python, TCL e Perl)		
Rudinei Pereira Dias, Unilassale	Geração de código de barras	Miolo[47]	¼ mês
Ely Matos[48], UFJF	Múltiplas contribuições na área de segurança, orientação a objetos e melhorias diversas	Miolo	6 meses
Total aproximado			16 meses

47 O Miolo (www.miolo.org.br) é o ambiente de desenvolvimento de praticamente todos os programas desenvolvidos pela Univates/Solis

48 O Ely Matos é o desenvolvedor externo mais ativo para o framework Miolo.

Desenvolvimento pago por outras instituições

Há alguns casos onde melhorias que implementamos em nossos sistemas, ou mesmo sistemas completamente novos que posteriormente adotamos, foram pagos por outras instituições. No caso do Sagu2[49], ainda em desenvolvimento, toda a modelagem e escrita de código inicial (cerca de dois meses de trabalho) foram pagas pela Universidade Federal de Roraima, onde os módulos acadêmicos já estão sendo utilizados.

Enquanto trabalhava como “Student Assistant” na Alemanha, o Marcone Luis Theisen desenvolveu o LdapSearch, um sistema de busca em bases de dados de autenticação de usuários no padrão Ldap que hoje é utilizado integralmente na Univates, em nosso processo de unificação de acesso aos nossos sistemas.

Muitas melhorias feitas no Gnuteca, Sagu e Agata Report surgiram a partir de serviços prestados para outras instituições de ensino, ainda que, enquanto isto ocorria, não nos preocupamos com este registro. Na quase totalidade das vezes em que algum de nossos sistemas foi implantado em outra instituição de ensino, novas necessidades acabaram também trazendo idéias que foram incorporadas ao nosso desenvolvimento.

Plano Diretor de Tecnologia da Informação – PDTI

Tudo o que já está escrito acima neste documento mostra que, ainda que a TI na Univates tenha evoluído de forma gerenciada, com resultados que, além de informatizar a maioria dos processos administrativos internos e acadêmicos da instituição, ainda foram capazes de atrair a atenção de outras instituições no país e no exterior. Esta evolução, porém, deu-se até pouco tempo de forma reativa: as necessidades surgiam, o CPD atendia a esta necessidade, mas não existia uma busca de antecipar novas necessidades que podiam surgir a partir do próprio cumprimento do plano estratégico da Univates.

Ainda assim, com o volume crescente de necessidades e a constante mudança em requisitos naquilo que já estava desenvolvido nos levou a um planejamento não de antecipação destas necessidades, mas de construção de um ambiente e processos que nos permitissem a adaptabilidade de nossos sistemas a necessidades cambiantes da instituição e de seus usuários. Deste planejamento surgiram processos como o de “manutenção proativa” anteriormente descrito; o ambiente de desenvolvimento Miolo, hoje base de todo o nosso desenvolvimento; a prática de construção de “protótipos prematuros”, auxiliando os usuários na definição de suas necessidades, e outros. A maioria destes processos estão distantes da compreensão dos usuários, a não ser que eles tenham que ser explicitamente envolvidos em algum deles. Isto é natural, pois ao usuário interessa o resultado final, e não o que leva a este resultado.

Esta proposta visa a aliar o PDTI ao Plano Estratégico da Univates, buscando não um exercício de futurologia, mas construir um guia de melhores práticas que alie de forma proativa a Tecnologia da Informação às necessidades já previstas, e para as quais soluções podem ser antecipadas de forma planejada. Além disto, esta proposta busca a manutenção do pioneirismo e destaque da Univates no desenvolvimento, integração de soluções e uso de Software Livre, e o exercício de sua latente liderança em um acordo interinstitucional e alianças estratégicas na área de TI que beneficiem a região e ampliem o mercado de atuação

49 O Sagu2 (http://sagu2.solis.coop.br) é a reescrita do Sagu utilizando o ambiente Miolo (www.miolo.org.br), o que permite uma melhor divisão de seus módulos, facilidade de manutenção e adição de novas funcionalidades.

para os alunos e egressos dos cursos afins à TI. Enfim, esperamos aqui exercer a idéia da Ecologia do Conhecimento, para a qual naturalmente evoluímos, com a ampliação de nosso Ecossistema.

Tomada de Decisão

Até hoje, a tomada de decisão sobre as ações do CPD partem da requisição por parte dos usuários (através de um projeto ou chamado técnico), sua priorização e atendimento pela equipe do CPD e terceirizados dentro da melhor percepção do que pode ser feito com os recursos existentes, da renegociação com o solicitante e do aval do pró-Reitor Administrativo/Financeiro. Isto gera alguma frustração dos usuários, especialmente quando um projeto por ele solicitado é preterido em função de outro que se tornou urgente, uma vez que, obviamente, com novas necessidades da instituição, as prioridades mudam, e os recursos para atendê-las são limitados. Há exemplos clássicos de projetos da própria Proad, como o GnuBuy (sistema de leilão reverso) que, mesmo entrando constantemente no planejamento, tem sido preterido em função de outros. Outros exemplos existem.

O avanço de nossos cursos na área de TI, e mesmo o aumento natural do uso da informática por todos os alunos da instituição, tem gerado demandas tanto de serviços quanto de estrutura, além de uma interação maior na definição de necessidades com a área acadêmica, especialmente no contato com o Prof. Marcelo Malheiros. Há um bom potencial de aproveitamento de alunos em vários projetos de TI, mas precisamos definir como isto será feito. A Solis, em sua última contratação, pontuou melhor os candidatos que já tivessem integrados em projetos da comunidade de Software Livre, e dois alunos da Univates com projetos no portal Código Livre foram admitidos como cooperados.

Comitê de Tecnologia da Informação

Com a idéia de melhorar o processo de tomada de decisão na disponibilização e planejamento futuro de recursos e sistemas de informática, sugerimos a criação de um Comitê de TI, presidido pelo Consultor de Tecnologia e formado por representantes de usuários de nosso ambiente de rede e sistemas, de tomadores de decisão (reitoria), dos principais fornecedores de serviços, dos corpos docente e discente. Inicialmente este comitê terá reuniões mensais onde serão apresentados os projetos em andamento, necessidades, limites de recursos, prioridades, e discutidas formas de atender a todos através de melhores práticas e acordos. O formato destas reuniões e o encaminhamento dos assuntos nelas tratados deve ser feito no mesmo formato já utilizado nas reuniões de gestão da qualidade.

Quem	***Resumo das Atribuições***
Consultor de Tecnologia	Preside o Comitê e com base em suas deliberações administra o orçamento destinado à TI. Administra conflitos e tem o "*voto de Minerva*" nos casos de impasse.
Coordenador do CPD Univates	Atua como secretário do Comitê, redigindo a ata das reuniões e cobrando junto aos envolvidos o encaminhamento das ações, quer envolvam compromissos da equipe interna do CPD ou terceirizados, assim como ações de usuários e demais pessoas apontadas pelos demais membros do Comitê da definição de necessidades ou

Quem	*Resumo das Atribuições*
	modelagem de sistemas.
Representante da Reitoria	Faz com que as discussões e deliberações do Comitê estejam alinhadas ao Plano Estratégico da Univates, e leva à reitoria questões do comitê e a realimentação para o Plano Estratégico.
Representantes dos grupos de usuários de sistemas internos	Traz ao grupo necessidades específicas relativas aos vários sistemas disponibilizados na Univates (Sagu, Gnuteca, Ágata, Portais, e outros) e auxilia na priorização de novos desenvolvimentos. Recomenda-se aqui que os grupos de usuários busquem ter suas reuniões em separado, com os técnicos responsáveis pelos sistemas, para que a discussão no Comitê seja sempre mais estratégica do que operacional.
Representante do corpo docente	Aponta necessidades de uso e desenvolvimento de TI necessários à área acadêmica, e busca aliar oportunidades de desenvolvimento à busca de conhecimentos que podem ser aproveitados na experiência educacional, em projetos de pesquisa (especialmente as que podem obter financiamento externo) e no envolvimento de alunos através de estágios e bolsas.
Representante do corpo discente	Traz ao grupo os interesses dos alunos, tanto como usuários dos serviços oferecidos como na busca de oportunidades de envolvimento através de estágios e bolsas. (DCE)
Representante do setor de apoio	Acompanha a evolução de necessidades de investimentos e serviços em infra-estrutura e traz informações sobre novas construções e instalações que podem influenciar a estrutura de TI, ou depender dela.
Representante da área de infraestrutura de telecomunicações	
Convidados especiais	Convidados pelo Comitê sempre que um determinado assunto ou conhecimento pode se beneficiar de ajuda externa.
Representante de fornecedores	Convidados pelo Comitê a participar de reuniões ou parte delas, respondendo e orientando em questões específicas aos produtos e serviços fornecidos.

Comitê Interinstitucional de Tecnologia da Informação

A Univates é reconhecida como referência na adoção e desenvolvimento de software livre na comunidade acadêmica, tanto nacional como internacionalmente. Uma prova disto é a expansão de nossos workshops de desenvolvimento para o SDSL, Seminário de Desenvolvimento de Software Livre, hoje organizado em parceria com a Unicamp, Unisinos, UnB e Istec, mas além disto a própria adoção de softwares por nós desenvolvidos

por instituições como a Unicruz, UFJF, UFRR e o apoio de organizações como a Unesco na disseminação do Gnuteca. Praticamente em todos os eventos dos quais participamos temos sido contatados em função da curiosidade por nosso uso de software livre. A Univates pode aproveitar este reconhecimento transformando-o em uma efetiva liderança em um Comitê Inter-institucional de Tecnologia da Informação, cujo objetivo é a adoção e disseminação de tecnologias livres e conteúdos abertos, com a adoção e definição de padrões e viabilizando com economia a informatização de processos administrativos e acadêmicos.

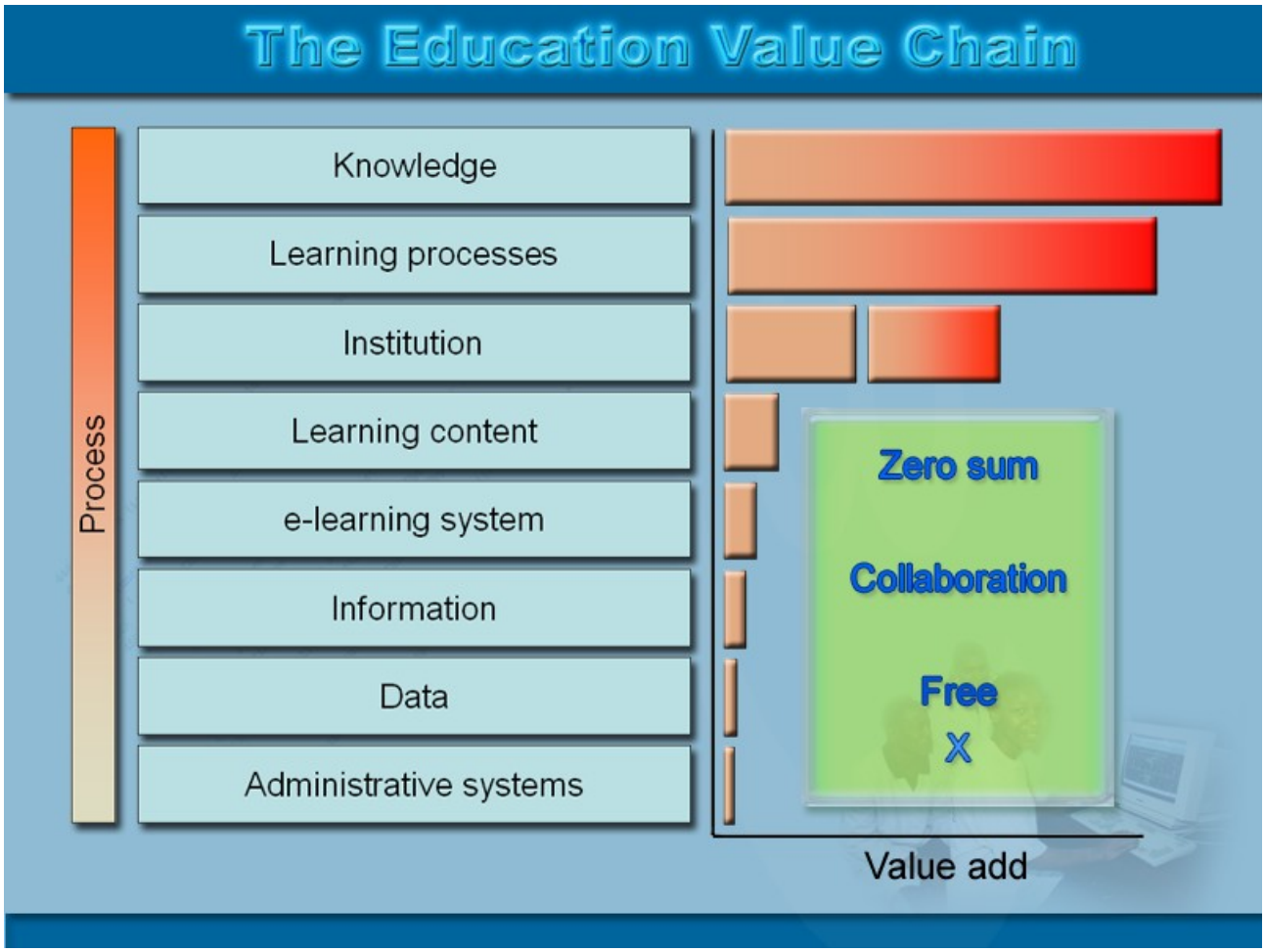


Figura 5 Cadeia de valor do processo educacional (Keats & Darries)

Segundo uma pesquisa feita por Derek Keats e Madini Darries[50] junto a estudantes de universidades sul-africanas, o valor percebido em uma instituição de ensino está no conhecimento que adquirem em sua passagem pela instituição, no processo de aprendizagem que leva a tal conhecimento (capacidade dos docentes em transmitir o conhecimento, metodologia de ensino) e no reconhecimento da própria instituição em seu meio (pesquisa, inserção comunitária, colocação de egressos no mercado de trabalho). Vários outros fatores como o próprio conteúdo usado nos cursos, dados e informações utilizados na gestão administrativa ou acadêmica e os sistemas administrativos são comuns para qualquer instituição de ensino e, por isso, tidos como uma base necessária, mas que não necessariamente agrega valor ao processo educacional. Assim, faz sentido que as instituições

50 Towards a Knowledge Ecology Strategy for the University of Western Cape

trabalhem em conjunto no desenvolvimento e integração de sistemas que atendam a esta base e diferenciem-se onde os alunos e a comunidade, na qual está inserida, realmente percebam valor.

O Comitê Interinstitucional de Tecnologia de Informação pode ser proposto a alguma das entidades das quais a Univates já faz parte, ao grupo de instituições que já utilizam os sistemas por nós desenvolvidos, ou ainda pode ter um caráter internacional já na sua formação, com a busca do apoio da Unesco e a integração à atividades como o Avoir (African Virtual Open Initiatives &Resources) e o Istec (Ibero-American Science and Technology Education Consortium). A atividade inicial deste comitê seria a definição da base tecnológica comum da qual todas as instituições de ensino poderiam se beneficiar para, a seguir, buscar, entre as instituições, desenvolvimentos e sistemas que pudessem ser padronizados, generalizados e usados por todas as demais, definindo grupos de trabalhos, investimentos, prazos e compromissos para atender aos interesses de todas as instituições participantes (ainda que os conteúdos desenvolvidos estejam disponíveis de forma livre a qualquer interessado).

Este comitê pode servir ainda para a identificação de problemas e soluções adotadas na inserção de cada instituição em sua comunidade regional, identificando quais destas soluções e modelos podem ser transplantados para outras regiões e realidades, especialmente ampliando a geração de emprego e renda e buscando ampliar oportunidades de intercâmbio para os estudantes.

Uma primeira proposta de trabalho para este Comitê Interinstitucional de TI é a definição de ações que levem ao desenvolvimento de um ERP Acadêmico.

***Nota**: Esta proposta de um Comitê Interinstitucional de TI ainda é um sonho que acredito ser realizado.*

Planejamento de Capacidade

Equipamentos

Como já exposto anteriormente, a expansão de nossa base de TI deu-se de forma reativa às necessidades da Univates. Mesmo os serviços hoje disponibilizados padecem de uma estrutura adequada para a sua oferta, e nossos usuários experimentam paradas que, se por um lado são inevitáveis em função da própria falibilidade dos equipamentos, por outro poderiam ser evitadas com o investimento em uma estrutura mais robusta e com maior capacidade de tolerância a falhas. Outros serviços ainda deixam de ser oferecidos (ou sofrem atraso em sua oferta), ainda que tecnicamente viáveis, seja por falta de equipamentos ou recursos humanos (salas de bate-papo especializadas por curso, sistema de agenda compartilhada, como exemplo). A adequação da capacidade atual de TI, com alguma folga, nos permitirá avaliar melhor as necessidades de expansão (hoje a expansão ocorre quando alguma necessidade fica na iminência do não atendimento) com medidas efetivas de sua utilização. Tais medidas hoje são usadas praticamente na identificação de gargalos que já impactam algum sistema.

Algumas ações de adequação ao consumo atual de TI pela Univates já estão sendo tomadas, especialmente com investimentos em ativos de rede. Outras merecem uma análise imediata (talvez com a rápida criação do Comitê de TI), como a aquisição de novos equipamentos para serviços eventuais, mas que já causam impacto na rede e sistemas (como relatórios,

autenticação, Teleduc) e a criação de uma estrutura de replicação mínima de ambiente para a rápida retomada de serviços em caso de pane generalizada e o aumento da autonomia no uso de energia elétrica com a aquisição de um grupo gerador (especialmente levando em conta que já sofremos por causa disto em momentos como matrícula de calouros).

***Nota**: A seguir, o documento oficial apresenta uma série de dados sobre o crescimento estimado da instituição nos próximos anos e, com base na capacidade atual, propõe a aquisição ordenada de equipamentos para os próximos três anos. Estes dados foram omitidos aqui por questões de confidencialidade.*

Há que se levar em conta ainda, especialmente com constante diminuição do custo dos computadores e do acesso à Internet, com sua crescente popularização, que observaremos nos próximos anos uma utilização maior dos serviços que oferecemos de forma remota. É de se esperar também que mais alunos venham à instituição com equipamentos que tenham capacidade de conexão à rede sem fio (pdas e notebooks wireless), e isto deve até ser incentivado. O próprio conteúdo multimídia oferecido nos cursos à distância, e que passa a ser gerado por nossos cursos na área de comunicação irá trazer impactos para a nossa estrutura de rede e servidores. Ainda neste ano estaremos testando vídeo-conferências através da web, e isto deve gerar um tráfego adicional que necessita ser medido e avaliado.

Novos serviços desejados podem entrar em calendário de testes e implementação, com seu consumo de capacidade agora devidamente medido, e novos investimentos avaliados e planejados de acordo pelo Comitê de TI. Dentre os novos serviços desejados estão (sem se limitar a eles):

- Servidor de DataWarehousing – para a extração de relatórios a partir de espelhos das bases de sistemas de produção, minimizando o impacto a eles;
- Sistema de autenticação e acesso para PDAs – para a disponibilização de serviços web como o controle de presenças e notas também para PDAs sem fio (e posterior expansão para um controle automático de ponto e freqüência com smartcards ou identificação rádio-identificada;
- Expansão da oferta de conteúdo multimídia via web em todas as salas (especialmente vídeo-conferência);
- Sala dedicada à vídeo-conferência
- Sistema de digitalização, arquivo e busca de documentos (facsímile digital)

Rede

***Nota**: As estimativas de valores relativas à expansão da rede foram omitidas aqui.*

Software

A quase totalidade do desenvolvimento de software para a Univates, assim como o suporte aos vários sistemas que utilizamos está terceirizada para a Solis, para a qual a “folha de pagamento” do CPD foi “transferida” em setembro de 2003. O acompanhamento dos projetos desenvolvidos pela Solis é feito através de documento próprio, e novas requisições de desenvolvimentos passam hoje pela aprovação da reitoria, mas podem passar pelo Comitê de TI aqui proposto. Durante o ano de 2004 a própria adequação na forma de relacionamento com a Solis e especialmente novas necessidades que surgiram na oferta de serviços via Inter/Intranet e a manutenção do desenvolvimento do Sagu em sua versão 1 fez com que realocássemos muitas horas prestadas pela cooperativa à Univates. A partir da renegociação

do contrato com a Solis em setembro de 2004 temos que rever as prioridades, especialmente no que diz respeito a destinação de horas ao desenvolvimento do Sagu2 para que não venhamos a ter problemas de desempenho com o aumento de sua base de dados e acesso a ela. Além disto, há projetos solicitados por usuários da instituição que ainda precisam ter suas horas aprovadas (sistema de avaliação 360º para o RH; um sistema para a gestão de contatos com alunos, egressos e comunidade; sistema para a criação de Cursos e Eventos com controle de fluxo; sistema para chamados de apoio das bibliotecárias para a consulta ao COMUT e consultoria bibliográfica; sistema para a rotulagem de amostras de resíduos através da web; BDI; Currículos com integração ao Lattes; portal dos professores) e novas horas a serem dedicadas a projetos como a implantação do sistema Nou-Rau (Biblioteca Digital), sua integração ao Gnuteca e ao Clara-OCR (sistema da USP para a digitalização de documentos) e a possibilidade de integração dos mesmos à bases da iniciativa Open Archives.

Suporte

O suporte ao ambiente de informática da Univates também está terceirizado em sua grande parte para a Solis, sob gestão do CPD. Ainda que muitas ações de manutenção proativa tenham sido tomadas e constantemente aprimoradas, o próprio aumento constante no número de equipamentos, serviços e usuários tem gerado uma demanda que as estatísticas de nosso sistema de chamados técnicos já mostra que está reprimida. A própria Solis deve propor um aumento no número de horas destinadas ao suporte de rede, software básico e aplicações.

Conclusão

A ideia básica desta proposta de PDTI é torná-lo um documento vivo, com ações de melhoria contínua que o realimentem e o transformem em um guia de ações e investimentos na expansão do uso da tecnologia na Univates, exercitando o conceito de ecologia do conhecimento em uma gestão que envolva e leve em conta na tomada de decisões todos os que, de alguma forma, usam a tecnologia em seu dia-a-dia. O Comitê de Tecnologia de Informação aqui proposto involucra usuários, fornecedores e tomadores de decisão e, idealmente, deve ter autonomia na negociação de um orçamento para a área de TI e sua administração. Com uma proximidade maior da área acadêmica, que possui representação no comitê, espera-se fomentar ações integradas, especialmente de desenvolvimento de software, que envolvam o corpo docente e discente. O Comitê Interinstitucional amplia a visão e ação das iniciativas de TI de várias instituições, buscando sempre a economia e a geração de oportunidades.

A proposta de continuidade dos serviços da Brod Tecnologia é passar da gestão do CPD, que auxiliou a evolução do uso de TI na Univates e seu destaque com o pioneirismo no uso e desenvolvimento de software livre, para uma gestão estratégica que evolui e integra o PDTI ao Plano Estratégico da Instituição, com a presidência do Comitê de Tecnologia de Informação e a busca de relações interinstitucionais que permitam a evolução do Comitê Interinstitucional de Tecnologia da Informação.

No ano seguinte à apresentação deste documento atuei, através da BrodTec, como consultor do Comitê de TI da Univates. A transição de minhas funções gerencias para o comitê deu-se de forma tranquila e, em 2005, deixei de atuar diretamente na instituição, mantendo-me ainda como cooperado e consultor estratégico da Solis até fevereiro de 2006, quando começa mais uma fase de aprendizagem na gestão de projetos.

O Mítico Homem-Mês

Fred Brooks Jr. confessa em seu mais recente livro, *The Design of Design*, que “tomou emprestado” o título de uma obra anterior de Gordon Glegg. Como tive a honra de ser o tradutor de duas obras de Brooks para o português, *O Mítico Homem-Mês* e o próprio *O Projeto do Projeto – da modelagem à implementação*, ambos publicados pela Campus Elsevier, sinto-me a vontade de, para este capítulo, tomar emprestado o nome de um dos livros e de um ensaio seminal de Brooks sobre Engenharia de Software.

Conheci a primeira edição, em inglês, de *O Mítico Homem-Mês* (*The Mythical Man-Month*) em 1988, quando fui pela primeira vez para os Estados Unidos ser treinado na manutenção dos processadores de comunicações NCR-Comten. Um dos instrutores, Jim Wild, percebeu em nossas conversas o meu interesse crescente por desenvolvimento de software e pela “forma correta” de fazer as coisas. De fato, desde que eu estudava eletrônica no Colégio Técnico Lavoisier, no Tatuapé, em São Paulo, eu já me interessava por software. Aprendi Basic com um curso publicado na revista Nova Eletrônica e exercitava meus programas no papel e, quando conseguia, nos poucos Sinclairs ZX-80 que o colégio havia comprado. Profissionalmente, a primeira linguagem de programação que aprendi foi o Assembler IBM/370, mas a verdade é que durante um bom tempo meu principal trabalho era com a manutenção de hardware. Verdade seja dita, porém, que ali no final dos anos 1980 havia muito software envolvido na manutenção de hardware, desde a execução e parametrização de rotinas de testes até a necessidade de, ao menos basicamente, entender o software executado pelos clientes para ter uma melhor ideia de como diagnosticar o problema.

Foi Jim Wild quem presentou-me com uma edição de *O Mítico Homem-Mês*, que até hoje é um de meus livros de cabeceira. Nele, Fred trata de algumas questões bem básicas, por vezes óbvias e por isso mesmo extremamente importantes. Dentre outras coisas, com Fred aprendi que o óbvio deve ser constantemente lembrado, exercitado, pois aquilo que está sempre à vista corre o risco de, com o costume, passar despercebido em algum momento. No artigo que dá título ao livro, Fred diz que adicionar mão de obra a um projeto que já está atrasado tem o grande potencial de atrasá-lo ainda mais, por vários fatores. O principal deles é o acréscimo da necessidade de comunicação entre os membros da equipe aumentada que, por si, já é uma sobrecarga de trabalho. Em “Não existe bala de prata” Fred fala da natureza dos problemas de software e sua individualidade, dizendo que não há uma solução mágica, como uma bala de prata para matar lobisomens, que funcione para todos os casos, mas em uma revisão de seu próprio artigo publicada na edição de 20 anos de *O Mítico Homem-Mês* (a que foi traduzida para o português), ele aponta algumas esperanças na busca desta solução.

O livro de Fred, publicado originalmente em 1975, mas contendo ensaios escritos ainda no final dos anos 1950, é surpreendentemente atual. Tive a oportunidade de perguntar a Fred a razão desta longevidade quando do lançamento da edição em português. Ele respondeu-me:

> *O livro é primariamente sobre os aspectos pessoais da engenharia de software, não sobre os aspectos técnicos. As pessoas não mudaram tanto assim desde 1975.*

Algumas das práticas que aprendi sobre Extreme Programming (e mais adiante sobre Scrum) solidificaram o que eu já havia aprendido e aplicado a partir da leitura dos ensaios de Fred. Posso dizer, com tranquilidade, que o livro que mais influenciou minha forma de gestão de projetos (e até de pessoas) foi *O Mítico Homem-Mês*. Com o tempo e a experiência, outras leituras, conversas e vivências práticas foram temperando esta influência. E não posso deixar de acrescentar aqui que sou extremamente grato ao André Wolff, editor da Campus Elsevier, por ter proporcionado a oportunidade que levou-me à tradução dos livros do Professor Brooks e a conhecê-lo pessoalmente, quando convidei-o a palestrar na edição de 2009 da Latinoware.

Na Bibliografia estão outros livros que ajudaram a formar minha base de pensamento para o exercício de minha profissão (e também para muitas atitudes de minha vida pessoal). Aqui cito apenas outros dois:

The Cathedral and the Bazaar, de Erick S. Raymond. Neste livro Eric analisa a maneira através da qual o sistema operacional Linux foi desenvolvido, sempre colocado para o escrutínio de um imenso número de usuários e desenvolvedores com seu código totalmente exposto. Segundo Raymond, dado um grande número de olhos, todos os problemas vêm à tona e, assim, podem ser mais facilmente resolvidos.

Just for Fun, de Linus Torvalds. Nele Linus dá a recomendação sobre a forma como um software deve ser liberado: logo cedo e sempre!

O portal Código Livre

Na edição de 2000 da Linux World Conference and Expo conheci mais de perto o projeto SourceForge.Net, um grande repositório de programas em código aberto disponibilizado para a comunidade de desenvolvedores. Todos podem usar o portal para distribuir seus programas, hospedar a página de seu projeto, ter o controle de versões na evolução de sua produção, discutir novas funcionalidades e prestar suporte aos usuários através de listas de discussões e utilizar uma série de outros serviços. Na noite do mesmo dia em que contatei o pessoal do SourceForge liguei para o Vilson Gartner, um dos principais desenvolvedores do Sagu e posteriormente principal autor e mantenedor do framework Miolo para que tomasse conhecimento do projeto e instalasse uma versão local, na Univates, para avaliarmos a possibilidade de usá-lo para disponibilizar os nossos projetos.

Na época já havíamos contado aos quatro cantos do mundo sobre o desenvolvimento do Sagu em software livre e já havíamos distribuído o código para algumas pessoas e instituições. Mas não tínhamos ainda um controle formal de versões e nem uma página web específica para o projeto. O SourceForge, por outro lado, não tinha uma versão em português e nós gostaríamos de oferecer nosso portal na nossa língua. O resultado disso foi a criação do portal CodigoLivre.Org.Br, que no momento da escrita deste livro hospedava mais de 2.000 projetos com mais de 15.000 usuários registrados. Hoje o portal está hospedado na Unicamp.

Quando o portal foi disponibilizado para a comunidade, em janeiro de 2001, seu nome era Código Aberto. Em novembro do mesmo ano, uma série de discussões com Richard Stallman, presidente da Free Software Foundation nos levaram a adotar o nome Código Livre para o portal.

Sinceramente confesso que, mesmo hoje, ainda acho que a discussão em cima da semântica e da importação desnecessária da duplicidade de significados da palavra "free" está aquém do real espírito de compartilhamento e liberdade de acesso ao conhecimento. Aliás, para brasileiros de uma certa idade, como eu, a "Abertura" virou quase sinônimo de "Liberdade" a partir das manifestações por uma "Abertura ampla, total e irrestrita", em voz alta, pelas ruas de nosso país, chamando por uma democracia que já tardava no final dos anos 1960 e início dos anos 1970. Hoje, ainda temos que aprender a valorizar melhor esta abertura e liberdade, não só com nosso voto, mas com a efetiva cobrança e fiscalização de nossos eleitos. Mas isto já está fora do escopo deste livro.

O Código Livre atendia a duas de minhas crenças, desenvolvidas a partir do aprendizado das leituras que citei anteriormente:

- O código que produzíamos era liberado logo de início e sempre. De fato, o controle de versões ficava aberto à visão de todos.
- Com isto, nosso desenvolvimento estava sempre aberto à críticas e sugestões, que vinham das mais variadas formas. Olhos suficientes e problemas à tona. Um verdadeiro incentivo a uma limpeza constante do código.

Uma coisa interessante que passou a acontecer foi que, cada vez mais, nossa equipe recebia solicitações de serviços e suporte daqueles que começavam a experimentar nossos sistemas, em especial o Sagu e o Gnuteca. Isto nos obrigou a pensar em uma forma melhor de estruturar estes serviços e esta foi, em grande parte, a origem da Solis.

Cooperativismo e Software Livre – Um modelo de negócios

> *Se uma estrutura de negócios cooperativa funciona para a Sunkist e para a Land O'Lakes, ela pode funcionar também para a produção de software. Depois de uma série de projetos de desenvolvimento de software em uma universidade, desenvolvedores no Brasil estão usando um plano de negócios antigo em uma forma inovadora.*

Assim foi apresentado o artigo[51] que escrevi para a revista Linux Journal de abril de 2004.

Já a partir de 2001 nós começávamos a viver uma situação, de certa forma, incômoda dentro da Univates. A divulgação do Sagu, do Gnuteca, e mesmo de nossa adoção do StarOffice mostravam claramente nosso pioneirismo na adoção de softwares livres. Especialmente os cursos que desenvolvemos para a equipe interna de funcionários sobre o uso do StarOffice acabaram sendo contratados por uma série de clientes no Rio Grande do Sul. Cursos sobre o desenvolvimento na linguagem PHP eram ministrados por nossa equipe em todo o Brasil e as implantações do Sagu e do Gnuteca começaram a acontecer e a exigir uma boa quantidade de suporte.

O incômodo vinha do fato que a Univates não era, e nunca quis ser, uma fábrica de software. Por outro lado, era inegável que havíamos desenvolvido uma fonte de receitas que deveria ser aproveitada. Na época, a Univates já começava a discutir a criação de uma incubadora (que depois concretizou-se na Inovates) e o pró-reitor administrativo, Professor Eloni Salvi, mestre em economia a quem eu me reportava diretamente, é um grande mentor em empreendedorismo. A equipe do CPD era formada, em sua quase totalidade, por alunos da instituição. Avaliando o que era gasto com o CPD em termos de salários, encargos, espaço físico e equipamentos, parecia bastante viável, economicamente, tornar toda esta equipe uma entidade terceirizada. Mas que tipo de entidade?

O Vale do Taquari tem uma tradição agrícola bastante forte, especialmente através de cooperativas de produção rural. Os alunos da Univates e, por consequência, os funcionários do CPD, eram em boa parte oriundos de famílias vindas do meio rural. Assim, o cooperativismo já estava na "veia" de todos. Analisando uma breve descrição sobre o cooperativismo, da *International Co-operative Association* (www.ica.coop), vemos que ele tem muito a ver com a forma de produção de software livre:

> *Cooperativas tem por base a ajuda mútua e o cuidado com o próximo. Uma cooperativa é um tipo de negócio ou organização. É um grupo de pessoas que trabalha em conjunto para resolver seus próprios problemas e atender a suas necessidades. As cooperativas diferem de outros tipos de organização por observar três regras principais: 1) cooperativas tratam as pessoas com justiça e respeito; 2) cooperativas encorajam as pessoas a trabalharem na solução de seus problemas mútuos; e 3) cooperativas fornecem produtos e serviços que vão de encontro às necessidades das pessoas, ao invés de apenas fazer dinheiro.*

Estava claro que as 20 pessoas que deixavam seu emprego como funcionários do CPD da Univates tinham que ter o seu sustento, mas também era claro que a forma pela qual viria este sustento era diferente da forma pela qual funcionavam outras empresas de software. A Solis, Cooperativa de Soluções Livres, não teria patentes ou propriedade intelectual alguma sobre a sua produção, já que lidava, da mesma forma que uma instituição de ensino, com o conhecimento livre que sempre estaria ao alcance de todos. Sua capacidade de criação, de solução de problemas seria a base da oferta de seus serviços. Mas a Solis nascia com o apadrinhamento da Univates, que já seria seu grande cliente desde o princípio.

O que era o valor equivalente à folha de pagamento dos funcionários do CPD tornou-se o valor do

51 http://www.linuxjournal.com/article/7081?page=0,0

contrato entre a Univates e a Solis. Como o custo trabalhista da Solis era bem menor, a diferença permitiu o aluguel de uma sede e o financiamento de equipamentos para os cooperados, coisas com as quais a Univates não precisaria mais arcar, o que foi extremamente importante para demonstrar esta economia aos tomadores de decisão da instituição.

Com o tempo, outros clientes garantiram que a Solis ficasse mais independente da Univates. A Univates, por sua vez, acabou por reaparelhar seu CPD, hoje o NTI (Núcleo de Tecnologia da Informação) para o seu suporte e desenvolvimento interno, mas mantendo-se também como cliente da Solis.

A Solis é um dos projetos da BrodTec dos quais tenho muito orgulho, mas ele jamais teria sido possível sem a visão empreendedora da Univates, em especial de seu reitor, Ney José Lazzari e do pró-reitor administrativo na época, Eloni José Salvi que, junto comigo e com a bem-vinda colaboração do Prof. Derli Schmidt definiu o projeto que deu origem ao modelo de negócio da Solis, um exemplo de compromisso de uma universidade com o desenvolvimento de sua região.

Fundada em janeiro de 2003, a Solis é a primeira cooperativa de desenvolvimento e integração de soluções em software livre. Seu exemplo foi seguido por uma série de outras cooperativas em todo o Brasil e na América Latina e, graças a este projeto, tive a oportunidade de ministrar palestras e minicursos em vários lugares do Brasil e do mundo sobre este modelo de negócios. Hoje a Solis emprega, entre cooperados, estagiários e funcionários, mais de 60 pessoas, é uma das empresas que mais arrecada impostos no Vale do Taquari e ficou em terceiro lugar no ano de 2010 como a melhor empresa para se trabalhar, segundo o Instituto Great Place to Work® (GPTW), entidade com atuação em 40 países, em parceria com a revista Computerworld, que divulgou o ranking Great Place to Work – TI & Telecom.

Três anos de Microsoft

Em novembro de 2005 fui convidado a palestrar sobre modelos de negócios para software livre em um local um pouco diferente daqueles aos quais eu estava acostumado: a sede da Microsoft em São Paulo. Roberto Prado, gerente de estratégias da Microsoft, havia preparado um evento para os executivos de vendas e parceiros da empresa, acreditando que eles deveriam entender melhor o avanço do software livre e de que maneira poderiam competir ou colaborar com o modelo, usando-o a favor dos negócios da empresa.

Este não foi o meu primeiro contato, obviamente, com a Microsoft. Além de ter trabalhado em empresas que eram clientes da Microsoft eu mesmo fui um usuário do MS-DOS e Windows até 1993, quando comecei a usar de maneira cada vez mais crescente o Linux, até passar a usá-lo, para meu uso pessoal e na minha empresa, exclusivamente, a partir de 1998. No contato com os clientes de minha empresa, porém, sempre convivi com a plataforma Microsoft (além dos ambientes operacionais da Apple, da IBM e outros).

Na Linux World de 2002 a Microsoft já estava presente com um estande e com sua equipe vestindo uma camiseta com os dizeres: “We want to talk!” Como eles queriam conversar, fui falar com eles. O que o funcionário da Microsoft disse-me foi basicamente o que a empresa seguiu dizendo nos anos que se seguiram: “Não entendemos ainda o modelo do software livre e para nós não faz sentido abrirmos mão de nossa propriedade intelectual. Já cometemos um erro no passado ao não prestar muita atenção à Internet comercial achando que teríamos nós mesmos a nossa rede. Não queremos cometer o mesmo tipo de erro, por isso estamos aqui para conversar e entender melhor o que é o código aberto e como isto pode fazer sentido para nossos negócios.” As palavras podem não ter sido exatamente estas, mas o sentido do que ouvi está preservado.

Em 2003, quando eu organizava o primeiro Congresso Internacional Software Livre Brasil, que aconteceu em Curitiba, a Microsoft quis participar como patrocinadora e fornecer palestrantes para o evento. Concordei com isso porque sempre achei – e sigo achando – que faz parte da liberdade ouvir o que todos têm a dizer para a boa formação de nossa opinião. Isto causou uma certa comoção e muitos protestos, inclusive do presidente da Free Software Foundation, Richard Stallman, que fez questão de enviar sua palestra em vídeo para o evento, já que havia quebrado o braço e estava hospitalizado.

Depois da palestra que ministrei na Microsoft em São Paulo, ainda fui convidado a participar de um evento para parceiros da empresa, em Boston, 2006, desta vez como ouvinte e com direito a um almoço com o Steve Ballmer, CEO da empresa, para o qual poderíamos fazer perguntas. Na época, a Microsoft já havia lançado seu portal Port25[52] para comunicar à comunidade suas ações relacionadas a código aberto, em especial a criação de seu Open Source Software Lab e o início de iniciativas de interoperabilidade. Perguntei ao Steve Ballmer: “Está claro para mim que a Microsoft quer seguir crescendo e ganhando dinheiro, e agora sente a necessidade de trabalhar mais próxima a projetos de código aberto. Mas qual a real possibilidade de vermos a Microsoft abrir o código de seus produtos em prol de uma maior interoperabilidade?”. A resposta do CEO da Microsoft foi ainda mais direta que a que eu havia recebido em 2002: “A Microsoft é uma grande empresa de tecnologia com compromisso de aumentar o capital investido por seus acionistas. Nossa proximidde com projetos de código aberto é, em primeiro lugar, para entendermos como o modelo funciona. Já liberamos alguns de nossos códigos e estamos avaliando o resultado disto. Aonde fizer sentido para o nosso negócio, colaboraremos com a comunidade, abriremos nosso código. Aonde não julgarmos que isto faça sentido, manteremos nosso mecanismo de licenças e competiremos fortemente sempre que necessário. E como em tudo o que fazemos, onde entrarmos para competir será para ganhar.”

52 http://port25.technet.com/

No Fórum Internacional de Software Livre de 2006, a Infomedia TV decidiu sediar um debate entre a Microsoft e a comunidade de software livre. A Fabiana Iglesias, diretora da empresa, convidou-me a organizar os temas com os desenvolvedores de projetos em software livre e pediu ao Roberto Prado que fizesse a mesma coisa do lado da Microsoft. Foi um debate muito interessante sobre métodos e ferramentas de desenvolvimento, segurança e interoperabilidade. Coube a mim e ao Prado debatermos sobre licenças. O Prado defendeu a propriedade intelectual e eu defendi que licenças restritivas e propriedade intelectual não faziam sentido algum, dizendo inclusive que, se a Microsoft abrisse todos os seus códigos, isso não causaria nenhum impacto negativo aos negócios da empresa. Durante este debate, mais uma vez o presidente da Free Software Foundation, Richard Stallman, fez-se presente gritando, à frente das câmeras, “Libertas quae sera tamen” - Liberdade ainda que tardia.

Passados alguns meses, o Prado ligou-me e contou-me que estavam iniciando no Brasil uma série de parcerias com universidades públicas, incentivando a criação de laboratórios para tratar de questões de interoperabilidade e código aberto. Ele convidou-me a participar do projeto dizendo mais ou menos assim: “Você está aí defendendo o software livre, criticando o modelo da Microsoft, mas que tal vir trabalhar conosco na produção de software de código aberto, interoperável com Linux e Windows, trazendo tudo o que você tem dito diretamente aqui para dentro?” Pareceu-me hipocrisia não aceitar esta proposta e entre 2006 e 2009 a BrodTec coordenou o desenvolvimento de software nos laboratórios de interoperabilidade da Universidade Federal do Rio Grande do Sul (UFRGS) e da Universidade Estadual de Campinas (Unicamp), com algumas atuações também junto à UNESP e à PUC-RS.

A única condição colocada pela BrodTec para assumir este trabalho é que tudo o que fosse desenvolvido fosse, de fato, publicado na forma de código aberto, com o que a Microsoft estava plenamente de acordo, mas as publicações deveriam ser feitas no Codeplex, o portal de desenvolvimento colaborativo mantido, na época, pela empresa e hoje por uma fundação da qual ela faz parte. Partindo do mesmo princípio de publicação logo de início e de manter o código sempre visível, criamos no Codeplex o portal ndos.codeplex.com, um apontador para todos os projetos que desenvolvíamos junto às universidades.

Este contato muito próximo com a academia durante quase três anos, trabalhando com tecnologias inovadoras e em contato com professores pelos quais eu nutro grande admiração, como Philippe Navaux, Nicolas Maillard, Luciana Nedel, Manuel Menezes (da UFRGS), Sandro Rigo (Unicamp) e tantos outros foi muito especial para a minha aprendizagem – e também da minha sócia, a Joice Käfer, que teve a oportunidade de conhecer muitas pessoas brilhantes enquanto concluía seu curso de Engenharia da Computação. Os bolsistas de nosso projeto foram muitos durante este período e o nome de todos eles está nos projetos listados em ndos.codeplex.com. Todos têm a gratidão profunda, minha e da Joice, pelo tempo que passamos juntos e pudemos, dentre outras coisas, exercitar a metodologia Scrum, em especial no desenvolvimento do Interop Router na Unicamp.

Enquanto na Unicamp desenvolvíamos um único projeto, com uma equipe que variava entre quatro e cinco bolsistas e o Professor Sandro Rigo como orientador, na UFRGS tínhamos vários projetos, cada um tocado tipicamente por um aluno e seu orientador. Estes projetos iam desde a criação de seres humanos virtuais com articulações anatomicamente corretas até sistemas de visualização de realidade aumentada e de processamento paralelo massivo. Todos eles desenvolvidos em código aberto e acessíveis pelo portal ndos.codeplex.com. A organização diferente nos laboratórios de cada instituição nos permitiu exercitar formas diversas de gestão de equipe. Se o laboratório de interoperabilidade na Unicamp permitiu-nos trabalhar mais diretamente com o Scrum, o da UFRGS acabou por nos levar ao contato com pessoas, a leituras e estudos (de fato!) que nos fizeram explorar outros aspectos de engenharia de software.

Engenharia de Software para Software Livre

Quando a Joice graduou-se, no final de 2008, os bolsistas com os quais desenvolvíamos projetos de interoperabilidade, na UFRGS, incentivaram-na para que logo iniciasse seu mestrado, começando como aluna especial de algumas disciplinas de mestrado da própria UFRGS. Na época eu fazia a tradução de O Mítico Homem-Mês e a Joice a sua revisão técnica. Ela gostou da disciplina de Engenharia de Software, ministrada pelo Professor Marcelo Pimenta, e convidou-me a participar como aluno ouvinte.

As aulas aconteciam todas as sextas-feiras pela manhã, durante todo o primeiro semestre de 2009. Elas tiveram grande influência na minha decisão de levar adiante a escrita deste livro. De certa forma, eu achava que faltava-me uma certa base acadêmica para tratar de temas relativos à gestão e métodos de desenvolvimento, criação de modelos de negócios, trabalho em equipe e tudo o mais sobre o que agora escrevi. A troca de experiências durante as aulas e a presença constante e positivamente crítica, ainda que dura e justa, do Professor Pimenta serviu para que, também com a ajuda da Engenheira Joice, muito de meu conhecimento prático ganhasse uma solidez que não deixou de me surpreender. A própria tradução de *O Mítico Homem-Mês* foi beneficiada por esta experiência.

Como trabalho de final de disciplina, eu e a Joice desenvolvemos o trabalho Engenharia de Software para Software Livre, cujo conteúdo está, em grande, parte reproduzido abaixo.

Introdução

Para aqueles que não estão envolvidos diretamente com o desenvolvimento de software livre e aberto pode existir a impressão de que o mesmo se dá de forma anárquica e desorganizada, com uma grande dispersão geográfica de desenvolvedores e a falta de um processo formal de desenvolvimento e documentação. Tal dispersão, porém, é um dos fatores que leva à necessidade da utilização de ferramentas e repositórios que permitam a interatividade entre os desenvolvedores e a organização do código por eles desenvolvido. Os repositórios de código livre não restringem o acesso apenas aos desenvolvedores de cada projeto, expondo o código ao escrutínio de uma comunidade extremamente ampla, evidenciando rapidamente possíveis erros. A correção destes erros pode ser feita tanto pelos desenvolvedores do projeto quanto por colaboradores eventuais.

Um dos exemplos mais expressivos é o sistema operacional GNU/Linux que, através da Internet, foi criado de forma coletiva. CASTELLS (2003) comenta sobre este fato:

> *Só uma rede de centenas, milhares de cérebros trabalhando cooperativamente, com divisão de trabalho espontânea e coordenação maleável mas eficiente, poderia levar a cabo a tarefa extraordinária de criar um sistema operacional capaz de lidar com a complexidade de computadores cada vez mais interagindo por meio da Internet.*

O envolvimento de empresas de tecnologia renomadas como a Sun e a IBM, respectivamente nos projetos OpenOffice.Org e Apache, que estão entre os discutidos neste artigo, também aproximou técnicas e processos tradicionais de Engenharia de Software a projetos de software livre e aberto.

Software aberto e software livre

No início dos anos 80, Richard Stallmann, do Laboratório de Inteligência Artificial do MIT, decepcionado com a forma pela qual as empresas de software passaram a proteger cada vez mais seus códigos através de patentes e mecanismos de direitos autorais, iniciou um movimento que desencadeou, em 1985, a criação da Free Software Foundation. Stallmann, propôs uma lógica diferente da imposta pela indústria quanto à produção e distribuição de programas de computador. A

Fundação considera os softwares como um produto científico e cultural que deve estar disponível para todos. Com esta filosofia, a FSF propiciou um entorno de desenvolvimento comunitário, incentivando a produção de programas alternativos aos comerciais que estavam disponíveis e protegendo-os legalmente através de licenças como a GPL (General Public License), que, segundo WILLIAMS (2002), garante aos usuários destes programas quatro liberdades:

- A liberdade de executar o programa para qualquer propósito;
- A liberdade de estudar como o programa funciona e adaptá-lo às suas necessidades. Acesso ao código-fonte é um pré-requisito para esta liberdade;
- A liberdade de redistribuir cópias, permitindo a ajuda ao próximo;
- A liberdade de aperfeiçoar o programa e liberar estes aperfeiçoamentos, de modo que toda a comunidade se beneficie. Acesso ao código-fonte é um pré-requisito para esta liberdade.

No final dos anos 90, empresas de tecnologia de ponta, especialmente do Vale do Silício na Califórnia, começaram a utilizar de uma forma mais ampla softwares livres na criação de seus produtos. Esta foi a época do estouro comercial das distribuições Linux (pacotes que incluíam o núcleo do sistema operacional e uma série de aplicativos que podiam substituir sistemas comerciais existentes), quando várias empresas abriram seu capital e passaram a atuar de forma mais agressiva no mercado.

Software livre é a tradução do termo, em inglês, *free software*. A palavra *free*, entretanto, na língua inglesa tem dois significados. Richard Stallmann (WILLIAMS op. cit.) distingue-os com os exemplos *free as in free beer* (*free* como em cerveja de graça) e *free as in freedom* (*free* como em liberdade). Assim software livre diz respeito à liberdade e não à gratuidade.

Em função da dualidade da palavra *free*, Eric Raymond passou a liderar a defesa do uso do termo *open source software* (software de código aberto). Este termo também era benéfico às empresas que começavam a utilizar software livre mas tinham receio de seu cunho ideológico. Até hoje o debate entre o uso dos termos *open source software* ou *free software* continua. Para o escopo deste trabalho considera-se indistintamente os termos software livre e aberto, levando-se em conta que os projetos aqui considerados têm seu código fonte aberto e o mesmo é disponibilizado de forma livre.

Modelos de desenvolvimento de software: a Catedral e o Bazar

BROOKS (1995) afirma que a adição de desenvolvedores a um projeto atrasado apenas irá atrasá-lo ainda mais, em função do aumento exponencial da complexidade e da comunicação, enquanto a quantidade de trabalho aumenta linearmente. RAYMOND (1999) observa que, se a afirmação de Brooks fosse integralmente real, o Linux seria inviável. Adiante, em sua argumentação, Raymond cita o trabalho de Gerald Weinberg, especialmente sua discussão sobre “programação sem ego”, observando que quando os desenvolvedores não são possessivos com relação ao seu código e encorajam outras pessoas a procurarem por problemas e melhorias em potencial, o avanço do software se dá dramaticamente mais rápido do que de qualquer outra forma.

A partir disto, da análise do desenvolvimento do Linux e de sua própria experiência como *hacker*, Raymond descreve dois estilos de desenvolvimento: a Catedral e o Bazar.

Nesta metáfora a Catedral corresponde a um processo de desenvolvimento unilateral comparado a de um pastor que entrega sua mensagem aos fiéis sem que esta seja contestada. O Bazar, por outro lado, corresponde a um ambiente amplo de negociação e troca de idéias, onde os comerciantes competem entre si e ouvem seus potenciais compradores que influenciam diretamente no conjunto de mercadorias que desejam comprar. A *Figura 1* ilustra esta metáfora aplicada ao desenvolvimento de software.

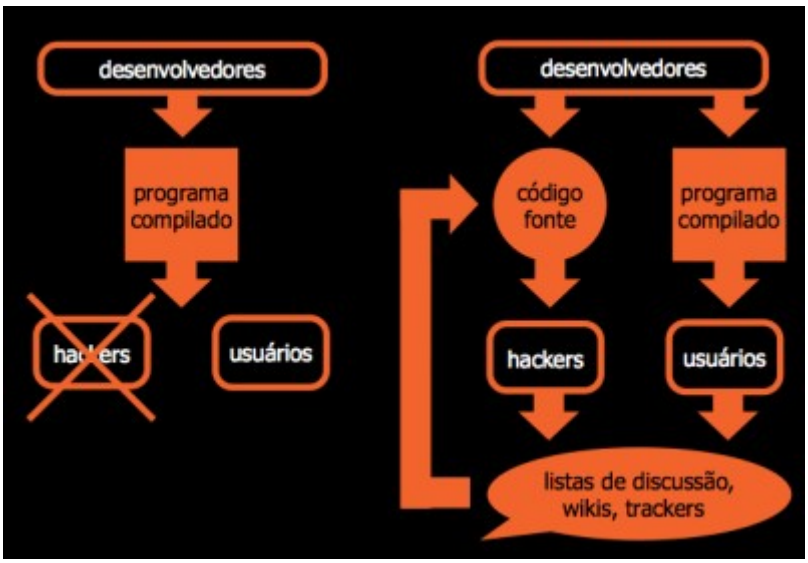


Figura 1 – Estilos de desenvolvimento Catedral e Bazar (FABERLUDENS)

No lado esquerdo da ilustração está o modelo Catedral onde uma equipe de desenvolvedores entrega um programa compilado, ou seja, sem disponibilizar o seu código fonte, aos usuários. A indisponibilidade do código fonte impede a colaboração externa à equipe. À direita está o modelo Bazar, no qual notam-se imediatamente duas diferenças fundamentais:

- o código fonte é disponibilizado para que outros desenvolvedores (*hackers*) possam conhecê-lo e modificá-lo;
- há um processo de realimentação do qual tanto os desenvolvedores quanto os usuários participam (através de listas ou fóruns de discussão, ambientes colaborativos e outros), contribuindo para a melhoria contínua do programa desenvolvido.

A popularização do Linux e de outros softwares livres acabou por mostrar o poder do modelo Bazar e muitas empresas passaram a utilizá-lo em maior ou menor grau. Alguns produtos de software migraram do controle de empresas de tecnologia para comunidades de desenvolvimento. Dois exemplos bastante significativos são o navegador *web* Firefox, originado do Netscape que pertencia à empresa AOL, e o conjunto de aplicações para escritório OpenOffice, que resultou da abertura do código do StarOffice pela Sun Microsystems.

O contrário também acontece. Produtos originados de comunidades de desenvolvimento têm hoje papel importante dentro de grandes empresas. O servidor *web* da IBM tem por base o software livre Apache e o Google é grande usuário e contribuidor de projetos livres como a linguagem Python.

A necessidade de um controle das contribuições recebidas e a utilização de software livre de forma crescente em um ambiente profissional e empresarial levou as comunidades de desenvolvimento à utilização de uma série de práticas de organização e controle, visando garantir a qualidade dos produtos desenvolvidos. Na próxima Seção serão abordadas algumas práticas adotadas por comunidade de desenvolvimento elencadas neste artigo.

Processo de desenvolvimento de software livre

De acordo com MOCKUS et. al. (2002), o estilo de desenvolvimento de código aberto tem a capacidade de competir e, em muitos casos, tornar obsoletos os métodos tradicionais de desenvolvimento de software comercial. Os autores concluem isto ao examinar os métodos de desenvolvimento de dois projetos de código aberto bem conhecidos e populares: o servidor *web* Apache e o navegador *web* Mozilla. Ainda em 2002, os autores já identificaram métodos, processos e boas práticas aplicadas ao desenvolvimento de software aberto, que são radicalmente diferentes do

estilo comercial de desenvolvimento de software:

- sistemas abertos são construídos por potencialmente centenas ou milhares de voluntários, ainda que alguns destes "voluntários" tenham seu trabalho patrocinado por suas empresas;
- o trabalho não é designado, os desenvolvedores escolhem as tarefas que querem executar;
- não há um projeto detalhado do sistema;
- não há um plano de projeto, cronograma ou lista de entregas.

Com o amadurecimento da utilização e do desenvolvimento de softwares livres também em ambientes comerciais, porém, é possível encontrar um melhor detalhamento de projetos, incluindo sua especificação, documentação, cronogramas e listas de entregas.

O desenvolvimento de software livre, em sua maioria, se dá em comunidades de desenvolvedores e usuários. Alguns projetos, porém, têm empresas como suas mantenedoras, com maior ou menor envolvimento da comunidade. O projeto MySQL[53], por exemplo, tem como sua principal mantenedora, hoje, a Sun Microsystems, com forte envolvimento da comunidade. A brasileira Solis[54], uma cooperativa de desenvolvimento e integração de softwares livres, possui uma série de produtos que, por sua especificidade, mesmo distribuídos sob licenças não restritivas, não agregam uma grande comunidade de desenvolvimento em seu entorno.

Nas Seções seguintes serão abordadas as práticas e ferramentas utilizadas em algumas comunidades de desenvolvimento de software livre.

Práticas das comunidades de desenvolvimento de SL

Com o objetivo de verificar as práticas utilizadas em projetos de naturezas diversas, mas todos desenvolvidos em software livre, foi feita uma análise sucinta baseada na documentação disponível nos próprios portais *web* das comunidades e, sempre que possível, em trabalhos de outros autores.

As comunidades elencadas foram as seguintes:

- kernel Linux;
- servidor *web* Apache;
- linguagem de programação Python;
- sistema gerenciador de banco de dados PostgreSQL;
- framework de desenvolvimento Ruby on Rails;
- gestor de conteúdo *web* Drupal;
- conjunto de aplicativos de produtividade para escritório OpenOffice.Org.

Linux

Em agosto de 1991, Linus Torvalds enviou um email para o grupo comp.os.minix (que discutia e trocava informações sobre o sistema operacional Minix), dizendo o seguinte:

> *Olá a todos vocês usando o minix. Estou construindo um sistema operacional livre (apenas como um hobby, não será tão grande ou profissional como o gnu) para o 386 (486) e seus clones. (...) Qualquer sugestão é bem-vinda, mas não prometo que as implementarei. (TORVALDS, DIAMOND, 2001).*

O tempo mostrou que o desenvolvimento, estilo Bazar (RAYMOND op. cit.), do Linux foi capaz de aglomerar um grande número de desenvolvedores, usuários e mesmo empresas que exploram comercialmente o sistema.

53 http://www.mysql.com
54 http://www.solis.coop.br

TUOMI (2000) lembra que, até o Linux ser possível, entretanto, uma série de outros atores, eventos e metodologias já existiam, e procura resumir, graficamente (*Figura 2*) os principais elementos que permitiram o surgimento do Linux.

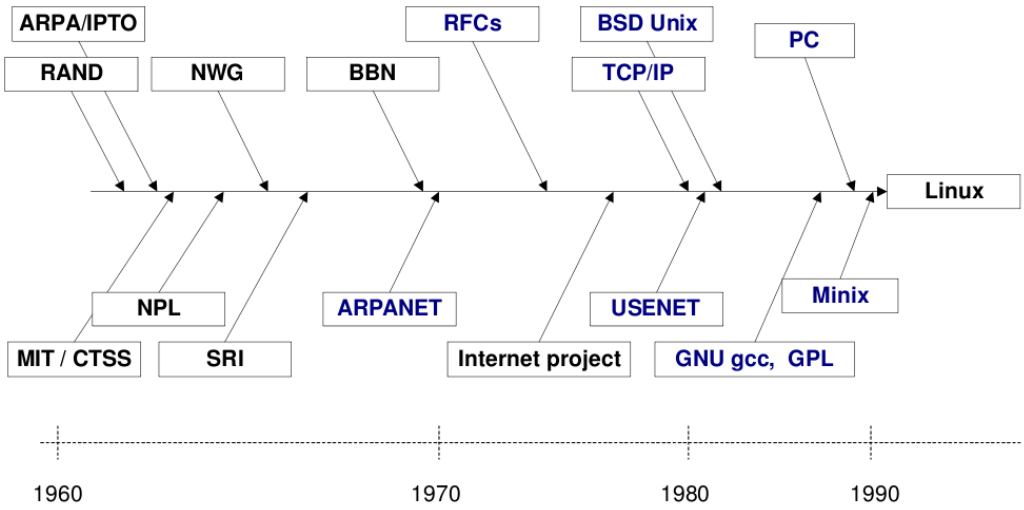


Figura 2 - Origens do Linux (TUOMI, 2000)

De fato, a grande maioria dos projetos em software livre tem sua origem em projetos anteriores, de maneira direta ou indireta, sendo possível observar elementos destes ancestrais no novo projeto. O desenvolvimento do Linux ainda é fortemente baseado em listas de discussões, mantidas inicialmente no Usenet[55]. Elas foram e ainda são a base do desenvolvimento do Minix, do Gnu e, por sua vez, derivaram dos RFCs (*Request for Comments* – Pedidos de Comentários), documentos que formaram a base de toda a estrutura da Internet.

O modelo de desenvolvimento do Linux foi herdado da cultura do Unix, na qual existe um conjunto de ferramentas, que podem ser facilmente reutilizadas e combinadas como componentes de novas ferramentas.

O anúncio do Linux na lista de discussões deu início a uma comunidade virtual que, em sua grande parte, não tinha interações sociais diretas e confiava totalmente na Internet como a base de suas ferramentas de desenvolvimento. Com o número de contribuições e registros de problemas, vindos de todas as partes do mundo, era claro que um controle maior sobre isto era necessário. No início dos anos 90 ferramentas como JitterBug e CVS eram usadas, respectivamente, para o registro de problemas e o controle de versões. Mais recentemente elas foram substituídas pelo BugZilla[56] e o Git[57]. Outra ferramenta adicional é o PatchWork[58], usado para o controle de contribuições oferecidas pela comunidade de desenvolvedores e administradas por cada um dos mantenedores dos vários módulos do kernel Linux que têm o direito de colocar o código diretamente no sistema de controle de versões.

A distribuição do kernel Linux inclui dois arquivos que informam sobre os mantenedores dos vários módulos (MAINTAINERS) e sobre os autores das várias contribuições recebidas (CREDITS) (LEE, COLE, 2003). Estes arquivos mostram a “hierarquia” do desenvolvimento do Linux. A rigor, qualquer pessoa pode contribuir para o código do Linux, mas ele precisa passar pelo crivo e aprovação do mantenedor do módulo em questão e, posteriormente, ser incluído no ramo estável da distribuição (depois de passar pelos estágios de desenvolvimento).

55 Usenet (do inglês *Unix User Network*) é um sistema de discussões distribuído disponível na Internet.
56 http://www.bugzilla.org/
57 http://git.kernel.org
58 http://ozlabs.org/~jk/projects/patchwork/

No momento da produção deste documento, eram 546 os mantenedores dos vários módulos do Linux listados no arquivo MAINTAINERS e 498 contribuidores no arquivo CREDITS. Os comandos utilizados, em uma console Linux, para obter o número individual de mantenedores e contribuidores foram, respectivamente, os seguintes:

```
cat MAINTAINERS | grep "P:" | sort | awk '!x[$2]++' FS="P:" | wc
cat CREDITS | grep "N:" | sort | awk '!x[$2]++' FS="N:" | wc
```

Apache

O projeto Apache[59] teve início em fevereiro de 1995 com a formação de um consórcio para a solução de problemas existentes no programa httpd, desenvolvido por Rob McCool no NCSA (National Center for Supercomputing Applications)[60]. O nome do servidor Apache veio do fato de que ele era o mesmo programa httpd com remendos (do inglês *"a patchy" server*). Posteriormente, o Apache foi todo reescrito e há muito pouco ou nada do seu código original (CONLON, 2007).

O projeto é mantido pela Apache Software Foundation, que cuida de outros projetos além do servidor *web*, contando com mais de 800 colaboradores diretos com acesso ao código fonte.

O enfoque do desenvolvimento do Apache é manter o projeto pequeno. Ele é liderado por um comitê gestor cuja participação está limitada àqueles que contribuem com o código do projeto. Toda a evolução deve ser resultado de um consenso técnico. Qualquer funcionalidade que ultrapasse a do servidor *web* básico é tratada como um projeto auxiliar, que deve integrar-se ao principal através de sua interface bem definida.

Interessados em avançar na hierarquia do projeto devem começar utilizando-o, depois contribuir com o código e eventualmente ganhar acesso à base do código fonte (controle de versões). A partir daí podem candidatar-se a membros do comitê gestor.

A metodologia usada pela comunidade do Apache é baseada em listas de discussões específicas para usuários, desenvolvedores, testadores, anúncios de novas funcionalidades, entre outras. O registro e controle de problemas são feitos através do Bugzilla que também serve para a submissão de pequenos consertos ao código fonte (*patches*). Qualquer código submetido deve estar de acordo com o guia de estilo[61] disponibilizado no portal do projeto.

A documentação oficial, também disponibilizada no mesmo portal, é mantida em formatos XML bastante estritos e os colaboradores devem segui-los. Existem, porém, várias páginas wiki[62] mantidas pela comunidade.

O controle de versões é feito com o software Subversion.

Python

A linguagem de programação Python foi lançada para a comunidade em 1991, o que significa que ela é contemporânea do kernel Linux. Seus desenvolvedores, porém, procuraram, desde o princípio, documentar o estilo, a cultura e as ferramentas de desenvolvimento. A maior fonte de referência para este documento foi o portal oficial da linguagem[63].

A cultura Python tem muito de humor e leveza. O próprio nome da linguagem deriva do grupo de humor britânico Monty Python e o criador da linguagem, Guido van Rossum, é chamado de Benevolente Ditador Vitalício. Em 1999, Tim Peters, junto com Guido, publicaram na lista de discussão comp.lang.python os princípios de projeto da linguagem pedindo, na mesma publicação,

59 http://httpd.apache.org
60 http://hoohoo.ncsa.illinois.edu/
61 http://httpd.apache.org/dev/styleguide.html
62 http://wiki.apache.org/httpd/
63 http://www.python.org

que os mesmos não fossem levados tão a sério (os autores não recomendam que a lista de princípios seja usada como tatuagem, por exemplo). Ainda assim, tais princípios ilustram bem a cultura e o estilo de desenvolvimento do Python:

- belo é melhor que feio;
- explícito é melhor que implícito;
- simples é melhor que complexo;
- complexo é melhor que complicado;
- plano é melhor que aninhado;
- esparso é melhor que denso;
- legibilidade conta;
- casos especiais não são especiais o suficiente para violar as regras;
- ainda que a praticidade vença a pureza;
- erros nunca devem passar silenciosamente;
- a não ser que sejam explicitamente silenciados;
- em caso de ambiguidade, resista à tentação de adivinhar;
- deve haver uma - e apenas uma - maneira óbvia de fazer algo;
- mesmo que tal maneira não seja tão óbvia à primeira vista, a não ser que você seja holandês;
- agora é melhor do que nunca;
- embora nunca seja frequentemente melhor do que exatamente agora;
- se a implementação é difícil de explicar, a ideia é ruim;
- se a implementação é fácil de explicar, talvez a ideia seja boa;
- espaços de nomes (*namespaces*) são uma ideia fantástica - vamos fazer mais deles!

Estes princípios, associados à forma de programação orientada a objetos da linguagem (que exige uma identação formal em sua sintaxe), levam a um código limpo e legível.

As ferramentas usadas no desenvolvimento da linguagem são descritas por Brett Cannon em "Guido, Some Guys, and a Mailing List: How Python is Developed"[64]. Para o registro e controle de problemas é usada a ferramenta RoundUp[65], que também é usada para a submissão de sugestões de código e solicitação de novas funcionalidades; para o controle de versões, o Subversion[66]; listas de discussões distintas são usadas para a comunicação dos grupos de desenvolvimento, gestão de problemas e para anúncios diversos. Sugestões de melhorias para a linguagem passam por um processo de análise e aprovação mais formal, iniciado pela submissão de um PEP (*Python Enhancement Proposal* – Proposta de Melhoria do Python), que além de passar pela equipe de desenvolvimento, deve ter o aval do mantenedor da linguagem (o Benevolente Ditador Vitalício).

PostgreSQL

O sistema gerenciador de banco de dados PostgreSQL[67] é um dos mais antigos projetos desenvolvidos em código aberto (CONLON, op. cit.). Michael Stonebreaker, que havia abandonado o projeto Ingres na Universidade da Califórnia, em Berkeley, no final dos anos 70, retornou, em 1985, para trabalhar em sua evolução, inicialmente batizada de Post-Ingres e depois reduzida para Postgres. O Postgres apenas ganhou um interpretador SQL[68] em 1994, graças ao trabalho de

64 http://www.python.org/dev/intro/
65 http://roundup.sourceforge.net/
66 http://subversion.tigris.org/
67 http://www.postgresql.org/
68 *Structured Query Language* – Linguagem Estruturada de Pesquisa

Andrew Yu e Jolly Chen, quando foi lançado com o nome de Postgres95 sob uma licença de código aberto.

Por ser o primeiro gerenciador de banco de dados relacional distribuído sob uma licença de código aberto, o Postgres logo foi adotado por uma crescente comunidade. Em 1996 teve seu nome trocado para PostgreSQL e passou a ser mantido pelo PostgreSQL Global Development Group. Sua equipe nuclear é formada por sete desenvolvedores, responsáveis pela direção do projeto; por 24 contribuidores principais, que se destacam por seu envolvimento com o projeto e a importância de suas contribuições; e por mais 36 desenvolvedores ativos[69].

Para coordenar o trabalho dos desenvolvedores, o projeto mantém páginas específicas[70] com o plano para a nova versão (ainda que a data de lançamento seja mais um desejo do que um compromisso), listas de tarefas, matriz de funcionalidades a serem disponibilizadas em cada versão, lista de perguntas mais frequentes e instruções para novos colaboradores. As colaborações devem sempre ser enviadas através de uma lista de discussão específica, o que garante que elas serão revistas e consideradas por desenvolvedores experientes e, caso aprovadas, submetidas à equipe nuclear. Há também uma lista específica para a discussão de bugs (que devem ser submetidos através de um formulário padrão) e outras para os demais componentes do projeto. Após o aceite de um registro de bug, o mesmo passa para uma lista de tarefas, na estrutura de um wiki público, para que os usuários e desenvolvedores possam conferir se um determinado problema que encontram é conhecido e já está com a sua solução encaminhada, ou se é algo novo que deva ser relatado.

A comunidade PostgreSQL utiliza o PGFoundry[71], uma implementação completa do repositório GForge[72], para o controle de colaborações e versões para todos os seus projetos auxiliares, mas mantém um repositório CVS[73] à parte para o próprio PostgreSQL, ainda que espelhos com o Subversion sejam externamente disponibilizados.

Ruby on Rails

Ruby on Rails (RoR)[74], um *framework* para o desenvolvimento na linguagem Ruby, começou como um projeto de David Heinemeier Hansson em 2004 e hoje envolve uma comunidade de mais de 1.400 contribuidores. De forma similar aos mantenedores do Linux, o Ruby on Rails conta com uma "equipe nuclear" (*core team*), que tem acesso direto ao código fonte do projeto, além de atuar em seu direcionamento futuro e filtrar sugestões de novas funcionalidades e melhorias vindas da comunidade de usuários.

Michael Koziarski, um dos membros da equipe nuclear, mantém um portal de práticas para a criação de aplicações com o Ruby on Rails, o TheRailsWay.Com[75], que serve para a divulgação de métodos de refatoração, reuso e outras técnicas. É comum observar perto de uma centena de comentários em cada um dos artigos publicados no TheRailsWay.Com.

David Heinemeier Hansson, em conjunto com Sam Ruby e Dave Thomas, escreveu o livro "Agile Web Development with Rails". Este livro, artigos e documentos espalhados pela comunidade de RoR mostram um forte compromisso dos usuários e desenvolvedores do projeto com metodologias ágeis.

Da mesma forma que o Linux, o RoR utiliza o Git como o repositório de código e controle de

69 Em12/05/2009, http://www.postgresql.org/community/contributors/
70 http://wiki.postgresql.org/wiki/Development_information
71 http://pgfoundry.org/
72 http://gforge.org/
73 http://www.nongnu.org/cvs/
74 http://rubyonrails.org
75 http://www.therailsway.com

versões. O sistema para o registro e controle de problemas é o LightHouse[76]. As listas de discussões, para o público em geral, a equipe nuclear, os anúncios de segurança e outras, são mantidas no Google Groups. Há ainda um wiki[77] para a produção colaborativa de documentos por parte da comunidade, um agregador de blogs[78] e uma coletânea de publicações pequenas[79] (*twits*).

Drupal

Como muitos projetos de software livre, o Drupal[80] surgiu para resolver um problema específico de um grupo pequeno de pessoas, no caso estudantes da Universidade de Antuérpia, na Bélgica. No ano 2000, conexões de alta velocidade à Internet ainda eram um luxo, mas Hans Snijder possuía uma conexão ADSL em seu alojamento na universidade. Ele e seu colega Dries Buytaert decidiram ampliar este acesso para mais oito colegas, o que fizeram com facilidade. A questão é que ainda faltava uma simples ferramenta para que os estudantes compartilhassem informações entre si. Dries resolveu este problema criando um pequeno quadro de avisos, baseado na *web*, onde o grupo poderia colocar informações sobre materiais de aula, o estado da rede ou simplesmente onde iriam jantar.

O software apenas recebeu um nome quando Dries concluiu sua graduação e ele e seus colegas decidiram abrir o quadro de avisos para o mundo externo a fim de que, mesmo deixando a Universidade, todos pudessem manter contato. Assim, nasceu o drop.org que deveria chamar-se dorp.org, em função do nome dado em holandês – nacionalidade de Dries – para um pequeno vilarejo. Um erro de digitação fez com que *dorp* virasse *drop*. Em janeiro de 2001 Dries decidiu dar o nome de Drupal para o software que era a base do quadro de anúncios e lançá-lo sob uma licença livre, de forma a que outros o conhecessem e viessem a contribuir com ele. Logo o Drupal tornou-se um popular gestor de conteúdo *web*, com uma grande comunidade de colaboradores e uma série de conferências e seminários organizados em todo o mundo.

A forma como o Drupal foi concebido, desde o seu princípio, buscando aplicar os conceitos de padrões de projeto[81] (*Design Patterns*), permitiu que o mesmo fosse facilmente estendido através de uma infinidade de módulos desenvolvidos e mantidos pela comunidade externa ao núcleo principal através de uma API bem documentada (Drupal API). Dentre os padrões explicitamente utilizados no Drupal estão os seguintes: *Singleton*, *Decorator*, *Observer*, *Bridge*, *Chain of Responsibility* e *Command* (GAMMA et. al. 1995). A última versão estável do Drupal (6.10) conta com mais de 2.300 módulos desenvolvidos pela comunidade, além de mais de 260 temas. Os módulos ampliam a funcionalidade para a gestão de conteúdo *web*, ferramentas de comunicação e mesmo aplicações bastante complexas como sistemas de gestão de contatos e controle de chamados técnicos. Os temas permitem que um portal desenvolvido com o Drupal tenha muitas aparências diferentes para o seu usuário final.

A base de comunicação da comunidade do Drupal é o próprio Drupal, com seus fóruns de discussões, sistema de registro de problemas e boletins de notícias. Para o controle de versões e repositório de código fonte, o Drupal utiliza o CVS, mas um repositório Git foi também disponibilizado em janeiro de 2009. Ainda que seja difícil estimar o total de contribuidores para o Drupal, em maio de 2009 o número de indivíduos que escreveram alguma coisa no Drupal Handbook, a base de documentação do projeto, era de 1258.

Qualquer nova funcionalidade ou código incluído no núcleo do Drupal deve passar pela aprovação

76 https://rails.lighthouseapp.com
77 http://wiki.rubyonrails.org/
78 http://planetrubyonrails.com/
79 http://search.twitter.com/search?q=rails
80 http://drupal.org
81 http://api.drupal.org/api/file/developer/topics/oop.html/6

daqueles com acesso à manutenção do código fonte (*core commiters*). Hoje, uma pessoa é responsável por isso: Dries Buytaert. Ainda assim, Dries conta com o apoio dos mantenedores dos ramos estáveis e de desenvolvimento da versão atual, duas imediatamente anteriores e a versão futura. Há ainda 23 mantenedores[82] responsáveis por módulos que são considerados parte integral do núcleo do Drupal e que estão junto ao pacote do sistema distribuído para a instalação. Todos estes mantenedores devem ser indicados por Dries, ou aprovados por ele após a manifestação individual do interessado ou recomendação de outros.

OpenOffice.Org

O OpenOffice.Org surgiu a partir do StarOffice, um produto proprietário da empresa alemã Star Division que foi comprada pela Sun Microsystems em 1999. Assim, o StarOffice passou a ser um produto da Sun e após ter o código, pertencente a outras empresas, removido, foi lançado com o nome de Open Office e em código aberto. Mas como Open Office era a marca registrada de outra empresa, o mesmo foi renomeado para OpenOffice.Org (CONLON, op. cit.).

O OpenOffice.Org atraiu um grande número de usuários por ser a primeira suíte de produtividade interoperável com arquivos escritos no Microsoft Office e de código aberto.

Para contribuir com o projeto, os interessados devem concordar que seu código seja de propriedade compartilhada com a Sun, tendo que preencher um documento de aceitação (*Sun's Joint Copyright Assignment).* Além de desenvolver código fonte, a comunidade é incentivada a participar das listas e fóruns de discussões sobre qualidade, marketing, ajuda a usuários, interface, entre outros. As principais fontes de documentação para os desenvolvedores estão reunidas em um *wiki*[83] e recomenda-se, para garantir a interoperabilidade entre linguagens de programação, modelos de objeto e arquiteturas de hardware, a utilização do UNO (Universal Network Objects) que é o modelo de interface baseada em componentes para o OpenOffice.Org. O wiki apresenta um capítulo sobre o uso de *Design Patterns* (*Singleton, Factory, Listener, Element access*, *Properties*, *UCB comments* e *Dispatch comments*) e estilos de codificação para o projeto.

Organizado na forma de um comitê, o projeto OpenOffice.Org é similar ao Apache, onde existem membros, contribuidores, desenvolvedores e líderes do projeto, tendo a Sun como mantenedora. Além disso, a CollabNet hospeda e colabora com a gestão do projeto.

Para o registro e controle de bugs é utilizada uma versão modificada do Bugzilla, chamada IssueTracker. O CVS é utilizado para o controle de versões e repositório de código.

A Engenharia de Software e o Software Livre

De acordo com FIORINI (1998):

> *A Engenharia de Software visa sistematizar a produção, a manutenção, a evolução e a recuperação de produtos intensivos de software, de modo que ocorra dentro de prazos e custos estimados, com progresso controlado e utilizando princípios, métodos, tecnologia e processos em contínuo aprimoramento. Os produtos desenvolvidos e mantidos, seguindo um processo efetivo e segundo preceitos da Engenharia de Software asseguram, por construção, qualidade satisfatória, apoiando adequadamente os seus usuários na realização de suas tarefas, operam satisfatória e economicamente em ambientes reais e podem evoluir continuamente, adaptando-se a um mundo em constante evolução.*

Para entender melhor quais práticas de Engenharia de Software estão evidenciadas no desenvolvimento de SL, a Figura 3 resume a cadeia de valor formada pelas comunidades, práticas executadas e ferramental de apoio que resultam em projetos de SL.

82 http://cvs.drupal.org/viewvc.py/drupal/drupal/MAINTAINERS.txt em maio de 2009
83 http://wiki.services.openoffice.org

De maneira geral, observa-se que as comunidades de desenvolvimento de SL, por mais que funcionem de maneira independente umas das outras, utilizam um conjunto de ferramentas que instrumentalizam as práticas que aplicam.

Tomando por base o documento eletrônico (*ebook*) Gestão de Projetos de Software Livre: Uma Abordagem de Práticas, publicado pela organização Via Digital, e REIS (2001), a seguir são listadas algumas práticas de Engenharia de Software relacionando-as com as práticas utilizadas pelas comunidades anteriormente citadas.

Especificação de Requisitos

Não foram encontradas evidências de especificações de requisitos desenvolvidas anteriormente ao início de cada projeto. Acredita-se que isto deva-se ao fato de que grande parte dos projetos surge de motivações pessoais (os requisitos estão na cabeça do desenvolvedor original), o novo software replica funcionalidades de algum produto existente (a análise de requisitos já existiu para este produto existente) e a natureza do SL é evolutiva (a interação e contribuições da comunidade fará com que novas necessidades possam ser atendidas na medida em que se manifestem).

Gerência de Configuração

A gerência de configuração permite que os desenvolvedores trabalhem de forma paralela e eficiente em um mesmo projeto. Para isto as comunidades utilizam repositórios de software com sistemas de controle de versão que permitem que os vários desenvolvedores submetam seu código de forma compartilhada e controlada, permitindo retornos à versões anteriores em caso de problemas. Estas ferramentas permitem a manutenção simultânea de ramos de produção (estáveis), de teste e desenvolvimento de um mesmo software.

Observou-se que todas as comunidades possuem práticas de gerência de configuração, ainda que nem todas utilizem as mesmas ferramentas em seus projetos. A Tabela 1 sumariza as ferramentas para a gestão de configuração de cada um dos projetos aqui mencionados.

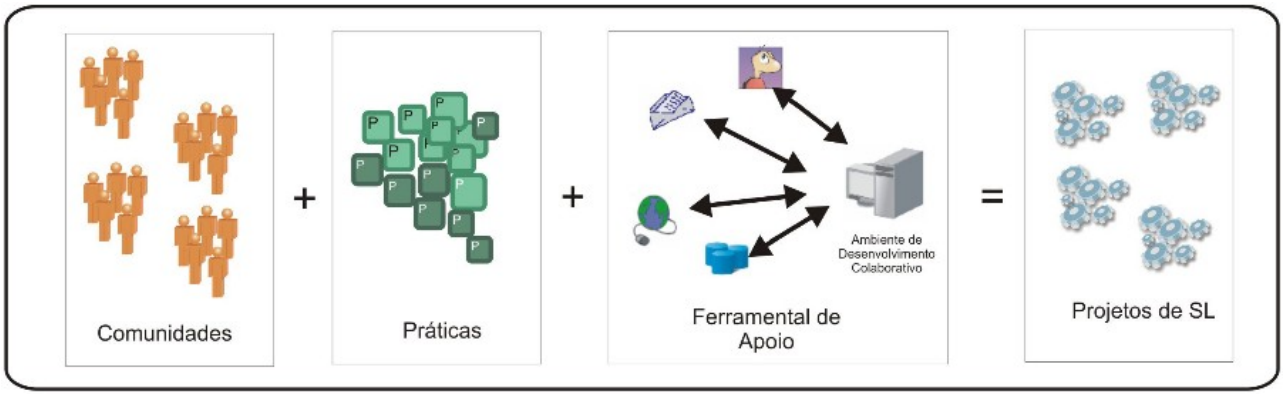


Figura 3 - Corrente de valor para produção de projetos de SL (VIA DIGITAL)

Projeto	**Ferramenta**
Kernel Linux	Git
Apache	Subversion
Python	Subversion
PostgreSQL	CVS
Ruby on Rails	Git

Drupal	CVS e Git
OpenOffice	CVS

Tabela 1 – Ferramentas de Gerência de Configuração

Coordenação

Tipicamente, no início de projetos de SL não existem procedimentos definidos de coordenação. Eles surgem e evoluem organicamente na medida em que mais desenvolvedores agregam-se aos projetos. Como observado nas comunidades de desenvolvimento, um projeto pode surgir a partir da iniciativa de um único desenvolvedor (como ocorreu com Linux, Python, Ruby on Rails e Drupal), a partir de um grupo tentando solucionar um problema comum (Apache) ou através de uma iniciativa empresarial ou acadêmica de abrir o código de um determinado produto para o desenvolvimento comunitário (OpenOffice.Org, PostgreSQL).

Os projetos que surgem através da iniciativa individual possuem, normalmente, uma coordenação centralizada, fortemente focada na figura de uma pessoa que define os rumos do projeto e é o juiz final em eventuais conflitos. Linux, Python, Ruby on Rails e Drupal apresentam este tipo de gerência.

Os projetos que surgem através de um grupo ou a partir de empresas tendem a se organizar através de comitês. Estes são formados por desenvolvedores eleitos internamente na comunidade ou pelos membros fundadores do projeto. Os projetos que apresentam este tipo de coordenação são Apache, PostgreSQL e OpenOffice.

Nota-se, porém, independentemente da forma de organização de cada comunidade, a prevalência das listas ou fóruns de discussões na coordenação e distribuição das tarefas de desenvolvimento.

A Tabela 2 resume as ferramentas usadas na coordenação de cada projeto.

Projeto	Ferramenta
Kernel Linux	Listas de discussões CVS Patchwork
Apache	Listas de discussões Subversion Bugzilla
Python	Listas de discussões Subversion RoundUp
PostgreSQL	Listas de discussões CVS Fluxo de submissão de erros (formulário, lista e wiki)
Ruby on Rails	Listas de discussões Git LightHouse
Drupal	Drupal (módulos Fórum e Project)
OpenOffice	Listas de discussões CVS

	IssueTracker

Tabela 2 – Ferramentas de Coordenação

Gerência de Evolução e Manutenção

Na medida que os projetos de SL evoluem, é natural que sejam encontrados problemas (*bugs*) que necessitam ser priorizados e resolvidos de acordo com seu grau de severidade. Todas as comunidades observadas utilizam algum método de gerência de evolução. Esta gerência ocorre da seguinte forma: qualquer usuário ou desenvolvedor, ao detectar algum problema, faz o registro do mesmo em ambiente disponibilizado pelo projeto, já atribuindo a sua percepção de severidade do problema (alta/média/baixa). Em seguida, o grupo ou desenvolvedor, responsável pela parte do projeto onde o problema foi detectado, captura diretamente este registro ou recebe sua notificação através de algum intermediário, prioriza e agenda sua solução.

Algumas comunidades utilizam o próprio sistema de registro de problemas também para o registro de solicitações de novas funcionalidades e sugestões de melhorias. Observam-se, porém, estilos diferentes na forma de tratamento destas solicitações. No PostgreSQL, por exemplo, há uma matriz de funcionalidades a ser implementada a cada versão e um processo de submissão de melhorias que devem ser avaliadas por desenvolvedores mais experientes e aprovadas pela equipe nuclear. Já no Drupal as ideias para as novas funcionalidades podem surgir a partir dos fóruns e solicitações formais (*feature requests* - enviadas a partir do mesmo sistema de submissão de erros), mas as mesmas só farão parte de uma próxima versão do sistema se isto for da vontade do líder do projeto.

A Tabela 3 resume as ferramentas utilizadas pelos projetos em sua gerência de evolução e manutenção.

Projeto	Ferramenta
Kernel Linux	Bugzilla Listas de discussões
Apache	Bugzilla Listas de discussões
Python	RoundUp Listas de discussões
PostgreSQL	Formulário próprio Lista de discussões Lista de tarefas (*wiki)*
Ruby on Rails	LightHouse Listas de discussões
Drupal	Drupal (módulo Project)
OpenOffice	IssueTracker (baseado no Bugzilla) Listas de discussões

Tabela 3 – Ferramentas de Gerência de Evolução e Manutenção

Reuso e Componentização

A filosofia do Unix que defende a criação de ferramentas pequenas e autocontidas que podem ser

ligadas umas às outras para solucionar problemas mais complexos permeou integralmente o desenvolvimento do Linux e, em um grau maior ou menor, todos os projetos aqui citados. Isto está fortemente ligado às técnicas de reuso e componentização.

O reuso manifesta-se pelo hábito da comunidade em, antes de desenvolver qualquer coisa, verificar se esta já não existe.

A componentização está clara nas bibliotecas compartilhadas e disponibilizadas entre os vários projetos de SL. Apenas para ficar em um exemplo, a biblioteca Curl, utilizada para a transferência de arquivos via FTP, HTTP e outros protocolos, é utilizada pelas linguagens PHP (com a qual o Drupal foi desenvolvido), Python e Ruby, dentre outras. Ou seja, estas linguagens valem-se de um componente externo para implementar uma funcionalidade que nativamente não possuem, ao invés de desenvolvê-lo.

Teste e Garantia da Qualidade

Por ser desenvolvido de forma distribuída e colaborativa, o SL utiliza mecanismos de garantia de qualidade como: o uso de linguagens de alto nível (que segundo Brooks é um dos principais fatores para a melhoria da qualidade e produtividade no desenvolvimento de software); o uso de controle de versões (que permite um controle do desenvolvimento distribuído e o fácil retorno a uma versão funcional em caso da submissão de um código com erro) e a exposição do código a um imenso número de pessoas (de acordo com Raymond, “dados olhos suficientes, os problemas são evidentes”).

Além disto, práticas de liberação de versões de testes (*alpha* e *beta*) antes da publicação da versão final (*release*) também são comuns à todos os projetos observados.

Refatoração

Como observado anteriormente, o código fonte de qualquer projeto em SL está disponível ao olhar de todos. A primeira forma como algo é escrito certamente não será a última na composição de um projeto e existe uma boa chance que sequer seja do mesmo autor. Ou seja, o software é naturalmente refatorado pelo simples processo de interação entre os desenvolvedores e o código.

Além disto, os desenvolvedores, por saberem da exposição de seu código, preocupam-se com a elegância do mesmo. Isto é evidente nos princípios que ilustram a cultura e o estilo de desenvolvimento do Python, nas práticas recomendadas para a criação de aplicações com o Ruby (que explicitamente mencionam técnicas de reuso e refatoração) e nos fóruns do Drupal onde é comum ver um desenvolvedor refatorar o código de outro em uma simples troca de mensagens.

Práticas de desenvolvimento ágil

Ainda que não seja possível inferir no desenvolvimento de cada projeto qual a totalidade das práticas utilizadas pelos desenvolvedores, observa-se um forte envolvimento dos mesmos com práticas de desenvolvimento ágil (em parte já evidenciado nas Seções sobre Refatoração e Reuso).

O criador e líder do projeto Ruby on Rails é autor de um livro sobre desenvolvimento ágil com o *framework* e os blogs da equipe do projeto também mostram seu envolvimento com o uso e disseminação com práticas ágeis.

O Drupal possui módulos específicos para o desenvolvimento baseado em testes (*Test Driven Development*) e automação dos mesmos.

O Python é considerado por muitos uma linguagem de programação ágil (YOUNKER, 2008).

Além disto, algumas fases e práticas do Extreme Programming, por exemplo, podem ser aplicadas diretamente às comunidades de SL:

- o conhecimento não deve ser concentrado nas mãos de poucas pessoas, que devem ser trocadas de tempos em tempos. A observação dos mantenedores dos vários projetos que existe uma rotatividade e a própria abertura do código evita a concentração da informação;
- a simplicidade é uma palavra de ordem. As estruturas devem ser mantidas simples. Isto evidencia-se na filosofia do Unix, na cultura de desenvolvimento do Python e de outros projetos;
- o cliente está sempre disponível. Nada pode ser mais verdadeiro do que para um projeto de SL, cujos clientes estão, na maior parte das vezes, online utilizando e testando cada nova versão dos programas dos projetos;
- a propriedade coletiva do código. As licenças de SL, na prática, tornam naturalmente o código de propriedade coletiva.

Padrões de Projeto (*Design Patterns*)

Dos projetos observados, o Drupal e o OpenOffice.Org evidenciam a utilização de *Design Patterns* em seu desenvolvimento. Isto era esperado pois estes projetos constituem-se em aplicações. Os desenvolvedores podem optar por utilizar Python ou Ruby on Rails para desenvolver aplicações utilizando *Design Patterns* ou não. Não foi possível identificar se *Design Patterns* foram utilizados no desenvolvimento do Linux e do Apache.

Conclusão

Foi possível notar nos vários projetos em Software Livre aqui observados que, independentemente da forma como se iniciaram, todos eles evoluíram para a adoção de práticas de Engenharia de Software. Estas práticas evidenciaram-se em diferentes graus para cada um dos projetos. Em alguns casos, referências explícitas a estas práticas estão disponíveis nas áreas destinadas a desenvolvedores nos portais dos projetos, como nos casos específicos do Drupal, OpenOffice.Org, Ruby On Rails e Python. Nos demais casos, mesmo não estando tão explícitas, são possíveis de serem encontradas em fóruns e listas de discussões.

O Grande Contrato

Trabalho com várias formas de integração de sistemas e desenvolvimento de soluções desde o final dos anos 80, sempre usando novas tecnologias. Isto continua acontecendo na BrodTec. Fui apresentado ao livro "O Mítico Homem-Mês", do Fred Brooks Jr. em 1988 e ele foi fundamental na formação do meu pensamento sobre a forma através da qual projetos deviam ser desenvolvidos. Quase dez anos depois comecei a estudar Extreme Programming, que usei junto com a UML (Unified Modeling Language) na gestão de desenvolvimento do projeto Gnuteca a partir de 2000, assim como em outros projetos que vieram depois. Mais adiante, o Scrum passou a fazer parte não só da vida da nossa empresa como da minha forma de pensar.

O mais novo livro do Fred, "O Projeto do Projeto – da modelagem à implantação", em seu capítulo quatro, trata da questão do estabelecimento de contratos entre fornecedores e clientes. Já na abertura do capítulo, Fred cita um texto de Pahl e Beitz:

> *Qualquer tentativa de formular todos os requisitos possíveis no início de um projeto irá falhar e causará atrasos consideráveis.*

Mais adiante, o próprio Fred conclui:

> *A pressão por um conjunto de requisitos completo e acordado provém, em última instância, do desejo - com frequência uma demanda institucional - por um contrato de preço fixo para uma entrega específica. Ainda assim, esta demanda está baseada frontalmente na evidência concreta (?) de que é essencialmente impossível especificar um conjunto completo e preciso de requisitos para qualquer sistema complexo, a não ser através da interação iterativa com o processo de modelagem.*

Parece-me que todos aqueles que desenvolvem alguma experiência com gestão de projetos acabam chegando a conclusões parecidas. Jeff Sutherland, em sua apresentação As Raízes do Scrum[84] comenta que é importante levar em conta que as metas de um projeto são alcançadas a partir de um "espaço de navegação" dinâmico, onde uma série de coisas - como mudanças de tecnologia e requisitos - irão causar, inevitavelmente, desvios no rumo desta navegação, que devem ser constantemente considerados.

Eu defendo sempre o desenvolvimento de protótipos prematuros, mesmo que não totalmente funcionais, que permitam ao cliente experimentar seus próprios requisitos e seu atendimento. Desta forma, junto com a equipe de desenvolvimento, ele é capaz de avaliar, refinar o que está sendo desenvolvido, descobrir o que realmente deseja e, em especial, quais destes desejos realmente trarão as funcionalidades e vantagens realmente importantes para o sistema, todas alinhando-se cada vez mais a seu negócio, dentro do espaço de navegação que está sendo explorado. Protótipos de papel que simulam a interface de um sistema são muito eficazes.

Vou além. Piamente acredito que contratos de desenvolvimento são absolutamente inúteis e apenas amarram o cliente a definições que ele fez sem o total conhecimento do que ele realmente desejava. Infelizmente, hoje, os contratos mais penalizam o cliente do que o auxiliam. Eles dão a desculpa aos fornecedores de entregar, protegidos por um contrato, exatamente aquilo que o cliente não conseguirá utilizar na forma em que foi entregue.

Este é um problema cuja solução não é fácil e, ainda que exista, ela não pode, simplesmente, ser replicada em todas as situações em que ocorre. Idealmente, deve existir uma relação de confiança absoluta entre cliente e fornecedor, de forma que o cliente não tenha receios em mudar seus requisitos ao descobrir novas necessidades na medida em que o sistema é desenvolvido e que o fornecedor não fique em desvantagem ao ter que modificar o que está desenvolvendo - muitas vezes sendo obrigado a jogar fora parte de seu código e considerar novas alternativas.

84 http://www.brod.com.br/ra-zes-do-scrum

Há uma cultura muito forte, baseada na compra e venda de produtos finalizados. Infelizmente, tais produtos, especialmente tratando-se de software, existem cada vez em menor quantidade.

Será que uma empresa que adquiriu uma solução de gestão de relacionamento com seus clientes, há três anos, imaginou que estes clientes passariam a utilizar o twitter como a sua forma preferencial de elogiar ou reclamar dos produtos da empresa? Mais do que isto, eles esperam que a empresa manifeste-se também através do twitter. Mas é possível (ou mesmo vantajosa) a integração do sistema atual de relacionamento, de alguma forma automática ou semiautomática, com o twitter? Aí há uma série de outras questões e críticas relativas a graus de automação e integração entre sistemas. Eliminando humanos de certos processos, coisas importantes passarão desapercebidas.

Isto dá muito pano pra manga, mas só pra citar uma coisinha, eu sou totalmente contra a geração automática de boletins a partir de material que é colocado em sistemas de gestão de conteúdo. Por outro lado, acho legal avisar aos seguidores da empresa, no twitter, sobre a publicação de um conteúdo novo em sua página - desde que se tenha o pleno conhecimento de que estamos tratando de formas diversas de comunicação.

Mas divaguei. A oferta de uma solução que atenda a um cliente deve passar por uma fase de aquisição de conhecimento de seu negócio. Não total, claro. O cliente sempre dominará seu negócio e qualquer ferramenta tecnológica que entregarmos a ele deve auxiliá-lo a dominar ainda mais. Ter a pretensão de que entenderemos totalmente o negócio do cliente é a mesma coisa que imaginar que o cliente virá a dominar a linguagem de programação, frameworks e métodos que usaremos para desenvolver uma solução. De novo, devemos navegar, em conjunto com o cliente, no espaço do projeto, da modelagem e criação de seus sistemas.

Uma forma de se começar a migrar dos grandes contratos para uma solução de plena confiança mútua é, talvez, usar o passo intermediário de minicontratos. Identifica-se, junto com o cliente, uma área específica de seu negócio para a qual possa ser desenvolvida (ou melhorada) alguma solução, com segurança e compromisso de ambas as partes e um limite de tempo (e consequente limite de funcionalidades) bastante grande. O limite mágico de tempo, a meu ver, é de três meses. Ainda há muitas empresas que estão começando a explorar melhor seus portais de conteúdo, sistemas de relacionamento com clientes e muitos outros para os quais há uma infinidade de sistemas plenamente customizáveis, modulares, que darão o espaço necessário para uma compreensão melhor de outras necessidades do cliente. Ao final deste período, sempre em conjunto com o cliente, explora-se novas oportunidades. No decorrer do tempo, a navegação pelo espaço de projetos e soluções torna-se um processo contínuo e de confiança, onde o fornecedor tem a tranquilidade de sua remuneração e o cliente reconhece que esta remuneração é justa e traz benefícios a seu negócio. Tal confiança suplantará, então, a necessidade de um contrato.

Scrum

Caso você não tenha pulado diretamente para este capítulo, deve ter percebido a construção, a evolução orgânica de uma metodologia de gestão e desenvolvimento que está embutida em cada um dos projetos aqui descritos. Minha intenção com os dois últimos capítulos, quando falei sobre Engenharia de Software para Software Livre e sobre a evolução de um modelo de contrato entre fornecedor e cliente foi a de “criar o ambiente” para falar mais sobre o Scrum. Esta metodologia, forma e atitude de trabalho é a da qual mais tiramos elementos para o desenvolvimento de nossos projetos, sem esquecer o aprendizado anterior com Extreme Programming, UML e todos os ensinamentos do Fred Brooks Jr. e das várias pessoas que fizeram parte da vida da nossa empresa (e da nossa vida) até agora.

Os textos abaixo são adaptados e estendidos a partir de seus originais publicados para o portal Dicas-L[85].

A ordem nascida do Caos

Depois da excelente apresentação do Alexandre Marcondes no encontro do grupo SOLIVRE-PR em Ivaiporã, 2006, parei para pensar há quanto tempo eu já estava envolvido com gestão de projetos e o quanto pude aprender com a experiência alheia. Já cheguei a fazer apresentações sobre o que chamei de "evolução de uma metodologia orgânica de desenvolvimento", contando especialmente a história que levou ao desenvolvimento de vários dos sistemas da Solis. Posso dizer que, especialmente a partir de 1993 (mesmo ano em que conheci o Linux) fui me envolvendo de maneira crescente com a gestão de equipes e projetos. Como vim da área de suporte de hardware e as primeiras equipes que gerenciei trabalhavam com o suporte a clientes, eu estava acostumado com o imprevisível quando, gradualmente, fui migrando para a gestão de ambientes e desenvolvimento de software.

Talvez por ser virginiano (até demais, como alguns dizem), sempre me interessei por metodologias de trabalho que levassem a uma maior satisfação e produtividade, com liberdade de ação e valorização de talentos. Programação é, antes de tudo, uma arte. Mas antes disto, todo o trabalho bem feito é uma obra de arte. Assim, devemos conciliar nosso compromisso de "entregas" para nossos clientes com alguma forma de darmos a devida "liberdade criativa" aos que estão produzindo.

Com uma frequência maior do que eu gostaria, ouço pessoas dizerem que, muitas vezes, não existe a "maturidade" suficiente para que se permita um trabalho sem um absoluto controle, que as pessoas não podem trabalhar "soltas". Eu aprendi que, devidamente motivada, a pessoa trabalha dentro dos horários aos quais se impõe e entrega seus projetos dentro dos prazos (mesmo que, por vezes, tenha que renegociá-los) com a alegria de que fez algo porque quis e não porque foi obrigada a fazê-lo.

Clientes, gestores e equipes envolvidos em um projeto devem concordar em uma visão, um plano de ação e serem todos cúmplices de sua execução. A transparência deve existir do início ao fim do projeto, da sua negociação comercial à sua entrega e aceitação. A motivação para a realização do trabalho se constrói através da visão da produção de um bem comum, que servirá a todos os envolvidos não só como um produto final, mas como uma oportunidade de aprendizagem e interação com outras pessoas. Cada um que trabalha no projeto deve sentir-se automotivado pela vontade de aprender, crescer, ser remunerado de acordo e fazer parte de um ambiente que valoriza seu potencial e respeita sua liberdade.

85 http://www.dicas-l.com.br/brod

Meu livro de cabeceira sobre engenharia de software ainda é *O Mítico Homem-Mês*, escrito originalmente em 1975 por Fred Brooks, Jr. Ele acaba passando pouco tempo na minha cabeceira pois sempre o empresto quando há a necessidade ou quando eu sinto que o livro pode ajudar. Este livro influenciou tanta gente que uma busca no Google sobre o tema já praticamente permita que você absorva as ideias do autor sem necessariamente ler o livro. Se o inglês não é uma língua familiar para você, tente os resultados em português. Mas se a falta do conhecimento da língua inglesa é realmente o seu caso, você deveria se preocupar. A falta de conhecimento da língua inglesa é um dos principais pontos de bloqueio no avanço em uma carreira na área de informática.

Mas voltando ao livro, Fred observa que, quando um projeto está atrasado, adicionar pessoas ao projeto servirá apenas para atrasá-lo ainda mais. Ele também diz que devemos considerar o tempo que perdemos em gestão e comunicação quando temos pessoas demais trabalhando em um projeto e que ao calcular o tempo de desenvolvimento de qualquer coisa, temos que dobrá-lo, pois é muito fácil esquecermos que o programador precisa de "tempo para pensar" além do "tempo para programar".

A forma simples e objetiva com a qual Fred via as coisas já em 1975 caem como uma luva para as, hoje, chamadas metodologias ágeis de desenvolvimento. As metodologias ágeis destacam-se por reconhecer que mudanças acontecem no decorrer do desenvolvimento de um projeto e que o cliente deve estar envolvido e presente durante sua execução. Esta presença é necessária porque a interação entre as pessoas será constante e o produto final deve ser amigável a ponto de, praticamente, prescindir de documentação.

A metodologia Scrum complementa as práticas de Extreme Programming e, a meu ver, acaba servindo tanto como uma evolução para aqueles que já usam Extreme Programming como um excelente ponto de partida para a aplicação destas práticas.

No jogo de rugby, o "scrum" é a forma de reiniciar o jogo após uma falta acidental ou outro incidente onde não é possível determinar quem cometeu a falta. No basquete acontece de forma similar quando o juiz não consegue determinar para qual time deve ir a bola e a joga para cima à frente de um jogador de cada time. Só que, no rugby, todos os jogadores se posicionam em um bolo humano para competir pela bola no reinício de jogo. Não é tão simples como eu descrevi, há regras quanto à forma como a bola deve ser retomada, mas já dá pra ter uma ideia. O termo foi associado ao desenvolvimento pela primeira vez no artigp *The New New Product Development Game*, de Hirotaka Takeuchi e Ikujiro Nonaka. Neste artigo, os autores dizem que deve ser adotada uma forma de desenvolvimento onde toda a equipe trabalha como uma unidade para atingir um objetivo comum. Exatamente como é feito quando se tem que recuperar a bola em um "scrum" no jogo de rugby.

Previously on Lost

Tirando quem acaba de voltar de uma longa viagem fora do sistema solar, todos os demais já assistiram ou ao menos ouviram falar do seriado Lost: a história dos sobreviventes de um desastre aéreo que acabam em uma ilha não exatamente deserta. Nela, sempre que parece que a ordem é estabelecida, alguma surpresa acontece. O personagem John Locke - que casualmente, ou não, tem nome de filósofo - é o ScrumMaster da ilha. Ainda na primeira temporada, com o ataque surpresa de javalis, o homem teve a ideia de caçar e carnear os bichos. No seriado isto não é explícito, mas é evidente que Locke é gaúcho!

Quem já trabalhou em qualquer projeto em que a especificação resultou em algo que era exatamente o que o cliente queria, parabéns! Na maioria dos casos isto não acontece. Não por culpa do cliente, mas porque os projetos são desenvolvidos ao longo de um tempo onde as necessidades podem mudar e a própria interação entre a equipe de desenvolvimento e o cliente pode mostrar que existem

soluções melhores do que as que foram pensadas inicialmente - e é um erro ater-se a especificações iniciais quando uma possibilidade melhor, mais simples ou mais econômica de atender o problema aparece. As metodologias ágeis levam isto em conta. Tudo muda a todo o tempo! É o caos! Como colocar ordem neste caos? No princípio era o caos, diz a bíblia, mas em seis dias o criador deu um jeito e ainda descansou no sétimo. Deus também é ScrumMaster certificado.

Já falamos sobre o artigo *The New New Product Development Game*, de Hirotaka Takeuchi e Ikujiro Nonaka. Outros nomes estão envolvidos na conceituação e desenvolvimento da metodologia, dentre eles Peter DeGrace e Leslie Hulet Stahl, autores de *Wicked Problems, Righteous Solutions: A Catalog of Modern Engineering Paradigms* (onde o termo Scrum foi especificamente associado a software). Jeff Sutherland, Ken Schwaber e Mike Beedle são outros nomes pelos quais você pode procurar no Google se quiser aprofundar seu conhecimento no assunto.

O Scrum é um método de trabalho para equipes pequenas e tempos curtos de desenvolvimento de projeto. Você trabalha com uma grande equipe e projetos que duram anos? Sem problemas, divida as pessoas em equipes menores (entre cinco e dez pessoas) e seu projeto em subprojetos. O Scrum trabalha com o conceito de "sprints", que é o progresso do projeto no período de um mês (ou menos). Os requerimentos dos projetos são trabalhados em uma lista de tarefas a serem cumpridas (*product backlog*). As reuniões com as pessoas da equipe são diárias e não devem durar mais do que quinze minutos. Nelas são discutidas o acompanhamento das tarefas (*sprint backlog*) e, preferencialmente, cada tarefa deve ser realizada dentro de um dia (se levar mais de um dia, deve ser dividida em mais tarefas). Isto se faz para manter as coisas o mais simples possíveis. Quando aumenta a complexidade, aumentam as dúvidas e o ruído na comunicação, o que atrasa o projeto e arrisca os resultados finais. O coordenador geral do projeto é o ScrumMaster, responsável por garantir a aplicação da metodologia e atuar como o representante do "cliente do projeto" quando ele não está presente. A principal tarefa do ScrumMaster, porém, é a de remover obstáculos, independente de sua natureza.

Uma equipe Scrum terá membros com especialidades variadas, de acordo com a necessidade do projeto. Todos trabalham na equipe em tempo integral (talvez com a exceção do ScrumMaster, que pode estar coordenando mais de uma equipe, ou membros que executem tarefas acessórias ao time mas que devam estar comprometidas com o projeto). Os membros da equipe discutem entre si e se auto gerenciam. Não há níveis hierárquicos dentro de uma equipe. Durante cada sprint (o período de um mês de projeto) os membros não serão trocados.

A principal razão de se dividir as entregas de um projeto em sprints mensais é justamente a questão de manter-se o controle sobre as surpresas. Dentro de um período de um mês, uma parte do sistema será projetada, codificada, testada e entregue ao cliente. Neste período não serão admitidas mudanças de requisitos, pois isto ampliaria o tempo de desenvolvimento. Ao final do sprint, porém, podem ser revistos os requisitos e uma nova lista de tarefas pode ser criada para a adequação do produto dentro de um novo sprint. Isto tende a fazer com que os requisitos passem a ser cada vez melhor definidos e a codificação para o atendimento dos mesmos cada vez mais refinada. As partes acabam chegando em acordos de requisitos mínimos e imutáveis, com os quais todos se comprometem pelo período de um mês.

A lista de tarefas (product backlog) é a lista de tudo o que é necessário no projeto (independente do número de sprints que o irá compor). Ela deve contemplar o que no Extreme Programming chamamos de User Stories (se não lembra disto, volte ao capítulo sobre este assunto). Se no Extreme Programming dissemos que as User Stories eram Use Cases "diet", na lista de tarefas elas são ainda mais enxutas. Na medida do possível, os itens da lista devem contemplar o que o cliente (ou o usuário) deseja junto com as tarefas necessárias para atender estes desejos, de forma bastante sintética. As tarefas são priorizadas de acordo com o que o patrocinador (aquele que está "pagando" pelo produto) deseja. Isso deve ser feito da forma mais simples possível, permitindo a busca textual

por tarefas, em um documento que possa ser acessado e alterado por todos os membros do projeto. Desde um quadro branco até um wiki ou planilha no Gogle Docs servem para isto. O importante é que esta lista seja clara para você e para os membros de sua equipe. A forma como ela é implementada não interessa.

Com o Product Backlog em mãos, agora o mesmo será subdividido em Sprint Backlogs. Como você já imaginou, o Sprint Backlog são listas de tarefas que ocorrerão dentro de cada sprint, para uma determinada equipe. Nada impede que você subdivida as tarefas entre equipes distintas e tenha sprints diversos ocorrendo simultaneamente. Com o Product Backlog devidamente priorizado, busque subdividi-lo em temas que darão nome aos sprints. Imagine que você está montando um portal dinâmico para a sua empresa. Uma das tarefas é a disponibilização de um serviço de acompanhamento de pedidos para os clientes da empresa. O sistema já existe mas os clientes só podem acessá-lo através do telefone. Este sprint pode se chamar "Acompanhamento de pedidos através da web". Sua lista de tarefas incluirá o design da interface para o cliente, a conexão com o sistema ou a base de dados existente para o acompanhamento, testes, documentação e o que mais for necessário. Estas decisões são tomadas na reunião de planejamento do sprint, com a presença da equipe scrum, do ScrumMaster, o patrocinador do cliente, representantes dos usuários e gestores envolvidos no processo. Em sua apresentação *An Overview of Scrum*[86] a Mountain Goat Software apresenta exemplos do Product Backlog, Sprint Backlog e dá boas dicas sobre a reunião de planejamento do sprint. Melhor ainda, a empresa permite que a apresentação seja usada e modificada livremente por qualquer interessado, desde que seja mantida a referência de autoria.

O que você fez ontem?

Stephen Covey em seu livro *O Oitavo Hábito* diz (e muitos concordam com ele, inclusive eu) que a melhor forma de aprender alguma coisa é tentando ensiná-la. Foi por esta razão que escrevi, originalmente para o portal Dicas-L, esta série de textos sobre Scrum. Até começar a aprender sobre o Scrum eu utilizava basicamente o e-mail para minha organização diária. Era um método que funcionava bem para mim mas que, sou obrigado a reconhecer, pode não ser o ideal para todos. Minha sócia e colega de vários projetos, a Joice Käfer, chegou a olhar para mim de um jeito "eu te disse!" quando aprendemos que no Scrum os e-mails não substituem as reuniões diárias de 15 minutos. Praticamente a partir do início de nossa aprendizagem sobre esta metodologia já começamos a praticá-la, não sem fazer algumas adequações.

Uma coisa que chamou-me a atenção no Scrum é que há uma série de práticas recomendadas para reuniões, mas não encontrei muita coisa sobre o que seria a reunião inicial, onde se define qual será o produto desejado ao final do projeto. A ficha começou a cair quando assisti uma pequena introdução ao Scrum disponível no portal *Scrum for Team System*[87]. Nesta apresentação de Ken Schwaber, autor do livro *Agile Project Management with Scrum*, ele diz que apesar do Scrum ser muitas vezes chamado de "metodologia", na verdade ele não o considera como tal. Uma metodologia deveria dizer "o que fazer". Comparado a um esporte ou a um jogo, o Scrum é um conjunto de regras para eles. Por isto, para a definição do projeto que deve levar ao "product backlog" inicial, vale o que dizem outras metodologias que explicam melhor como fazer isto. É neste ponto, mais uma vez, que o Extreme Programming casa muito bem com o Scrum. Em reuniões com o cliente, construa as "user stories" e tudo o mais que for necessário para definição do projeto. Depois, com seu time, comece a dividir a execução do projeto em sprints mensais.

A prática e muitas leituras ensinaram-me que um projeto é bem controlável se ele dura até três meses. Caso dure mais do que três meses, é melhor dividí-lo em mais projetos (ou em fases que

86 http://www.mountaingoatsoftware.com/scrum-a-presentation – Procure pela tradução para o português, feita por Cesar Brod

87 http://scrumforteamsystem.com/

durem até três meses). Como cada sprint dura um mês, ao final de cada três sprints podemos concluir cada projeto (ou suas fases). As ferramentas que o SCRUM recomenda, porém, não servem só para controlar projetos bem definidos e delimitados, mas até tarefas constantes e diárias. Isto deve tornar-se mais evidente quando mais adiante falarmos sobre as reuniões diárias e exemplificarmos algumas planilhas de controle e acompanhamento de projetos.

O SCRUM começa a ser aplicado no product backlog, que irá traduzir em uma planilha que define as prioridades, a descrição breve e genérica de cada tarefa a ser executada, quem as executará e a estimativa de tempos. O product backlog pode ser alimentado a partir das user stories do Extreme Programming. Com isto em mãos, parte-se para a primeira reunião de planejamento do sprint.

Cada sprint deve ter um "tema" que identifique as principais tarefas que nele serão executadas. "Acompanhamento de pedidos através da web" foi o exemplo que utilizamos anteriormente. A reunião de planejamento deve contar com a presença do patrocinador do produto, da equipe scrum, dos gestores de projeto e do cliente ou usuários do produto que está sendo desenvolvido. Nesta reunião serão discutidos e avaliados o que já está disponível (se esta é a primeira reunião, muito pouco ou nada), quais as capacidades da equipe e o que cada pessoa estará fazendo, tecnologias adotadas e quais os pontos do product backlog serão atendidos. O resultado da reunião será o objetivo do sprint (o que será produzido) e o sprint backlog (a lista de tarefas específicas deste sprint). Esta reunião pode ter um tempo de preparação, com o gestor de projeto (ScrumMaster) fazendo perguntas aos integrantes, buscando saber de suas necessidades e habilidades.

A partir daí, ocorrem reuniões diárias, extremamente curtas (entre 15 minutos e meia-hora, sempre buscando chegar mais perto dos 15 minutos), onde são feitas as seguintes perguntas para cada membro do time:

1. O que você fez ontem?
2. O que você fará hoje?
3. Quais os obstáculos que impedem (ou podem impedir) seu trabalho?

Caberá ao ScrumMaster (não se preocupe, ainda falaremos mais sobre o ScrumMaster) remover estes obstáculos. Esta reunião deve ser presencial e não pode ser substituída por uma lista de discussões ou outra forma de encontro (ainda que eu não tenha encontrado nada sobre isto, creio que na absoluta impossibilidade de um encontro real, algum método de interação em tempo real - videoconferência, por exemplo - é aceitável). A idéia de reuniões diárias vem de nosso amigo Fred Brooks Jr. Em seu livro *O Mítico Homem-Mês* ele diz que os projetos devem ser conduzidos "um dia de cada vez". Através das reuniões diárias, todos os membros da equipe têm acesso ao cenário completo do estado do desenvolvimento, ao ouvir seus pares responderem às três perguntas acima (isto fecha também com a ideia de equipes pequenas. Imagine uma reunião durar 15 minutos com mais de cinco ou dez pessoas). Além disto, a presença ajuda a criar a pressão do compromisso de cada um fazer o que se comprometeu a fazer para todo o sprint e todos os dias. Reuniões diárias e curtas acabam por tornar desnecessárias outras reuniões.

Outro aspecto importante das reuniões diárias é que elas não são destinadas à solução de problemas. Como não existe hierarquia entre os membros de um time scrum, cada pessoa deve procurar resolver os problemas entre seus pares, acessando-os diretamente. Assim, se um desenvolvedor tem alguma dúvida sobre como implementar um determinado recurso, ele não precisa passar pelo ScrumMaster ou qualquer outra pessoa: deve perguntar diretamente ao usuário, cliente ou ao patrocinador do produto qual a interpretação deles de determinada tarefa ou função.

Product Backlog

Desde que trabalhei em um projeto financiado pelo governo finlandês, que buscava fortalecer a

associação do uso de software livre à geração de emprego e renda, tive vontade de criar um portal ibero-americano de colaboração na adoção, integração de soluções e desenvolvimento de softwares livres e de código aberto. A ideia, sobre a qual cheguei a conversar com várias pessoas, buscaria o financiamento de governos, instituições e empresas de vários países para manter um grupo pequeno (entre cinco e dez pessoas) que trabalhasse tanto na pesquisa de necessidades quanto na articulação de contatos que tornassem viáveis projetos que pudessem ser úteis à várias geografias. Em resumo, se existisse um projeto na Argentina que pudesse atender a alguma necessidade específica do México, Espanha e Brasil, seria criada uma rede entre os potenciais usuários e financiadores para que o projeto se tornasse economicamente viável e, mais do que isto, que pudesse gerar receita para empresas envolvidas no seu suporte e desenvolvimento - com isto criando também novos postos de emprego. A ideia não era tão inédita, mas buscava agregar boas práticas e vontades que desenvolvemos no projeto sobre o qual falei inicialmente.

Enquanto pensava em como exemplificar as ferramentas de acompanhamento e controle de tarefas do SCRUM concluí que o melhor era partir direto para a prática e usar o "Portal Ibero-americano de Colaboração e Desenvolvimento Tecnológico" como base para este exercício. Vamos ao projeto, que foi escrito muito antes de eu pensar no uso do Scrum:

Descrição do Projeto

> Este portal não pretende criar novamente o que já existe, mas, ao usar o máximo possível estruturas e sistemas existentes, quer que sua própria implementação já sirva como uma experiência de integração entre projetos da comunidade. Assim, alguns elementos clássicos e de sucesso entre a comunidade de software livre serão devidamente reconhecidos, valorizados e agregados a este projeto.
>
> Para o repositório de software, será usada a estrutura já implementada no CodigoLivre, o maior repositório latino-americano de software livre. Inicialmente, um espelho integral do portal CodigoLivre será montado na estrutura do Portal Íbero-Americano, com treinamento à equipe que estará envolvida nesse projeto.
>
> Implementada a estrutura do CodigoLivre, trabalharemos na criação de uma nova interface para o seu acesso, agregando ao próprio CodigoLivre funcionalidades que serão de interesse do portal ibero-americano (para o qual ainda devemos criar um nome atrativo e significativo). Esta interface deve permitir uma série de funcionalidades:

1. **Registro e hospedagem de projetos**: será utilizada a infraestrutura do CodigoLivre, expandindo-a para a seleção da linguagem em que o registro do projeto será feito (inicialmente português e espanhol) e facilitando a hospedagem de projetos com um "template básico" para a criação de um sítio web para o projeto, fóruns de discussão e outros itens que podem ser agregados posteriormente à medida em que o desenvolvedor se torna mais experiente com a ferramenta.

   Nota: Aqui imagino um "auto-piloto", disponibilizando uma página básica e uma seleção guiada da inclusão de ferramentas ou não, explicando uma a uma. O projeto Losango pode servir como base ou referência para isto.

   Exemplo: Deseja criar listas de discussão para o seu projeto? Recomendamos, caso seja usado este recurso, que se criem ao menos duas listas, uma para usuários e outra para desenvolvedores de seu projeto. Se você desejar, criaremos agora uma lista chamada projeto-user e outra chamada projeto-devel, caso prefira escolher outro nome para as suas listas ou acessar a configuração avançada, clique aqui...

   Este "auto-piloto" poderá ser chamado várias vezes dentro da página de gestão do projeto,

mas não pode ser destrutivo. Ele sempre deve detectar o que o usuário já possui e apenas oferecer novas ferramentas a serem agregadas.

Ao classificar um projeto, seu criador pode escolher categorias às quais o mesmo pertence, mas um administrador do portal pode "recategorizar" o projeto de forma a garantir sua correta exposição na busca por categorias e mesmo para "alinhar" projetos de natureza similar, buscando a interação entre eles.

Projetos hospedados em outros sítios podem ser categorizados também aqui, servindo este portal, então, como um apontador para outros repositórios como o SourceForge, Incubadora Virtual da Fapesp, Savanah e outros. No caso do CodigoLivre, porém, isto será feito de forma automática, garantindo já a exposição de todo o volume de projetos do CodigoLivre no portal íbero-americano.

2. **Notícias**: As notícias podem ser relativas aos projetos, de responsabilidade dos desenvolvedores e publicadas com destaque na página principal (de todos os projetos), na página do projeto (apenas as relativas a ele) e devem poder ser exportadas para outros sítios em formatos como o RSS.

   Notícias postadas pela comunidade ibero-americana também serão aceitas de várias formas, com níveis de mediação. Notícias postadas anonimamente sempre passarão pelo crivo do gerente de conteúdo. Notícias postadas por usuários registrados terão prioridade, mas também passarão pelo crivo de um gerente de conteúdo. O gerente de conteúdo pode autorizar alguns usuários a postarem notícias no portal sem a necessidade de mediação.

   Notícias que vêm de outros portais podem ser incluídas neste através de mecanismos como RSS.

3. **Multi-idiomas**: Inicialmente, todo o portal terá sua interface em português e espanhol. Os conteúdos podem ser disponibilizados em apenas uma língua, havendo neste caso, quando da exibição do conteúdo em uma língua para o qual não foi traduzido, a opção para que o leitor ou colaborador "envie uma tradução". Por exemplo, um projeto com desenvolvedores brasileiros irá postar suas notícias e demais informações em português. Quando alguém de língua espanhola acessar uma informação não traduzida, a lerá na língua original, mas terá a possibilidade de colaborar com uma tradução. Isto pode ser expandido para outros idiomas.

4. **Gestão do ambiente e estrutura de suporte**: Este portal visa ser referência e ponto de integração no desenvolvimentos de software livre para a comunidade íbero-americana. Além da ação prática de hospedagem de projetos (ou apontadores para outros portais), ele será a base de uma ação de integração e busca de recursos para desenvolvimentos que estejam ligados à geração de emprego e renda, inclusão social e resolução de conflitos.

   Desta forma, a fim de que o portal atinja o sucesso em suas metas, propomos a montagem de uma estrutura que pode ser ampliada à medida em que seu uso se intensifique:

Comitê Gestor

Formado por membros da comunidade, mediante indicação dos financiadores deste portal e com vagas abertas para a eleição democrática de outros membros da comunidade íbero-americana.

Equipe de suporte

- Tradutor português-espanhol e vice-versa;
- Gerente de conteúdo (para aprovar as notícias e negociar relações com as fontes);
- Analistas de suporte (2) - (para resolver questões técnicas e cuidar da saúde operacional do sistema);
- Diretor Executivo (representante do portal perante a comunidade, ligação entre o comitê-gestor e a equipe de suporte).

Ideias para o Cronograma de implantação

Fase 0: Implantação do espelho do CodigoLivre no portal íbero-americano, treinamento dos analistas de suporte

Fase 1: Criação das interfaces de registro de projetos baseadas no Fred, Losango interagindo de forma transparente com o CodigoLivre, conforme leiaute definido pela equipe de arte contratada à parte

Fase 2: Sistema de notícias

Fase 3: Multi-idiomas

Testamos toda a interface em português e depois faz-se o porte para o espanhol, deixando aberta a possibilidade de tradução para outras línguas.

Fase 4: Suporte continuado

Numa estimativa inicial, cheguei a oito semanas de desenvolvimento para as fases 0 a 3, com o envolvimento semi-integral de um gestor e o uso de pessoas contratadas para a escrita de código.

Com estes dados, chegamos a um Product Backlog, exemplificado abaixo:

Brod Tecnologia - Product Backlog				
Produto:	**Portal Ibero-Americano de Colaboração e Desenvolvimento Tecnológico**			
Data:	**30/10/2006**			
Prioridade	***Item***	***Descrição***	***(horas)***	***Quem***
		Startup	**242**	
Alta	1	Captação de recursos para o projeto	160	CB
Alta	2	Contratação da equipe	80	CB
Alta	3	backlog	2	CB
		Design	**40**	
Alta	4	necessidades e ferramentas a serem adotadas	40	CB
		Setup	**42**	
Alta	5	Divisão de tarefas	2	CB
Alta	6	domínio, implantação do espelho do CL	40	TBD
		Desenvolvimento	**240**	
Alta	7	do mesmo	80	Devel1
Média	8	projeto e gerais, incluindo publicação por RSS	80	Devel2
Alta	9	notícias	80	Devel3
		Lançamento e Marketing	**42**	
Alta	10	suporte	2	
Média	11	Lançamento para os meios de comunicação	40	
		Suporte continuado		

Sprint Backlog

Note que enquanto no Product Backlog fazemos a melhor estimativa das horas que gastaremos nas tarefas, é no Sprint Backlog que fazemos o registro destas horas e podemos confrontá-las com as estimativas. Sprints podem ocorrer em paralelo, desde que tenhamos como segmentar as tarefas. No nosso caso, poderíamos ter as tarefas de definição de leiaute ocorrendo em paralelo às tarefas de definição e aquisição de equipamentos. Após a reunião que irá definir as tarefas do Sprint, o ScrumMaster apenas acompanha o projeto, buscando eliminar qualquer obstáculo que a equipe tenha. O Sprint Backlog é o guia que a equipe usa para sua auto-gestão. Ela mesma o preenche. Por isto é útil tê-lo no formato de um documento compartilhado.

Brod Tecnologia - Sprint Backlog									
Produto:	**Portal Ibero-Americano de Colaboração e Desenvolvimento Tecnológico**								
Sprint:	**Captação de recursos**								
Período:	**30 dias a partir de 04/12/06**								
		Preencha esta coluna com a data em que a tarefa foi submetida	*Esta é a data limite para a entrega da tarefa, e deve ser menor que a do final deste sprint*	*A data em que a solução foi entregue*	*Para cada linha de tarefas, o total de horas gastas naquela tarefa específica dentro da semana especificada na coluna*				*Total de horas do mês*
Quem	**Descrição**	**Data de entrada**	**Data Limite**	**Data de solução**	**Semana 1**	**Semana 2**	**Semana 3**	**Semana 4**	**Total de horas**
Total de horas utilizadas na semana (Preencha para cada semana o total de horas utilizados na execução das tarefas)					60	40	40	20	160
CB	Desenvolvimento de Material para a Apresentação	4/12/2006	8/12/2006	8/12/2006	40				
FI	Contato com clientes e agenda de reuniões	4/12/2006	8/12/2006	8/12/2006	20				
CB/JK	Apresentação aos clientes	11/12/2006	22/12/2006	26/12/2006		40	40	20	

Burndown Chart

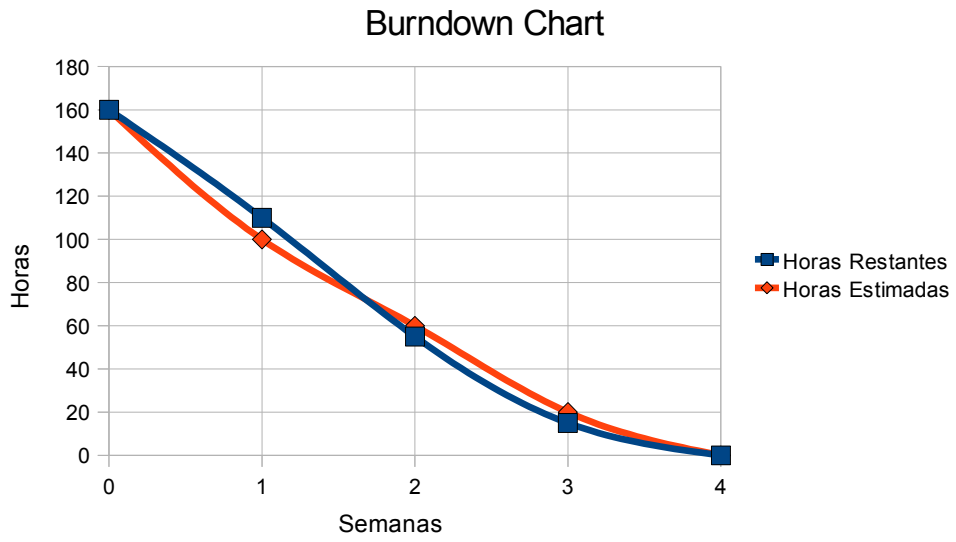
Outro artefato importante do Scrum é o Burndown Chart, complementando os backlogs e atuando como seu elemento de realimentação e controle de qualidade. A forma mais simples de implementá-lo é simplesmente adicionar uma coluna a mais na estimativa de horas do backlog, colocando ao lado das horas estimadas as horas efetivamente realizadas na execução de uma determinada tarefa. Com isto é possível verificar o quanto a nossa estimativa inicial estava correta ou não, melhorando-a continuamente a cada novo projeto.

De forma simples, para o nosso Sprint de quatro semanas, temos um total de 160 horas, que vão sendo consumidas ao longo do tempo de acordo com a tabela abaixo:

Semana	Horas Restantes	Horas Estimadas
0	160	160
1	110	100
2	55	60
3	15	20
4	0	0

Aqui foram consideradas as horas totais de todas as pessoas envolvidas no projeto. A coluna relativa à horas estimadas corresponde ao Sprint Backlog acima. Começamos a semana com 160 horas e, idealmente, consumiríamos 60 horas na primeira semana, 40 na segunda, mais 40 na terceira e,

finalmente, 20 horas na quarta semana. De fato, tivemos uma boa surpresa e consumimos apenas 50 horas na primeira semana. Precisamos averiguar a razão disto, já que nossa estimativa estava errada. Superestimamos o esforço? A tarefa não era tão complicada quanto pensávamos? Ocorreu algum fato novo que fez com que não precisássemos gastar todas as horas previstas? Este exercício de crítica das estimativas deve ocorrer constantemente, servindo como ferramenta de ajuste para estimativas futuras. O gráfico abaixo, montado a partir da tabela acima, mostra o quanto desviamos de nossa previsão, mas felizmente cumprindo o prometido dentro do prazo total para o Sprint.



A estimativa de um trabalho em horas, mesmo parecendo ser a mais direta, não é, na maioria dos casos, a ideal. A hora de um desenvolvedor nunca será igual a de outro e a qualidade do resultado do trabalho também varia. Um desenvolvedor competente pode resolver um determinado problema em oito horas. Outro desenvolvedor, talvez mais criativo, pode ficar simplesmente pensando na solução por algumas semanas e, ao visualizá-la, escrever seu código em alguns minutos e com muito menos linhas que o primeiro. Em uma equipe de desenvolvimento procura-se nivelar a equipe com padrões de codificação e guias de melhores práticas mas, ainda assim, conta-se com desenvolvedores com os mais variados graus de experiência. Mais adiante falaremos mais sobre formas de estimar o trabalho de desenvolvimento.

ScrumMaster

Ao contrário de um gestor de projetos tradicional, que é responsável por planejar, instruir e coordenar os trabalhos de sua equipe, nas metodologias ágeis de desenvolvimento este gestor deve principalmente ser um facilitador e motivador. O Scrum visa ser um conjunto simples e eficaz de regras e ferramentas para maximizar resultados, gerando o mínimo possível de sobrecarga administrativa. O ScrumMaster deve garantir que estas regras e ferramentas sejam usadas e que sempre estejam diretamente relacionadas à melhoria dos processos, melhor retorno de investimento e liberdade criativa para a equipe.

User Stories

> *Sócrates: Homero não diz muitas vezes e muito sobre as técnicas? Por exemplo, sobre a técnica do auriga – se me recordares o verso, eu te direi.*
>
> *Íon: Mas eu recitarei pois eu me recordo.*
>
> *Sócrates: Dize-me, então, o que diz Nestor ao seu filho Antíloco, quando o aconselha ficar atento a respeito da*

baliza, na corrida de cavalos em honra a Pátroclo.

Íon:

“Inclina-te, diz , no carro bem polido

docemente para a esquerda dos dois: o cavalo da direita

estimula com a voz, cede-lhe as rédeas com as mãos.

Na meta, certo, o cavalo da esquerda se lance,

a fim de que o cubo da roda bem feito

pareça tocar a meta: mas evita tocar na pedra.”

Sócrates: Basta! Esses versos épicos, Íon, se Homero diz corretamente ou não, quem conheceria melhor, um médico ou um auriga?

Íon: Um auriga certamente.

Sócrates: E é porque ele possui essa técnica ou por algum outro motivo qualquer?

Íon: Não, mas porque ele possui essa técnica.

Sócrates: Então a cada uma das técnicas foi dada por Deus uma função de ser capaz de conhecer? Pois não conhecemos pela técnica do piloto o que conheceremos pela técnica médica.

Íon: Não, certamente.

Sócrates: E nem conhecemos com a técnica médica essas também que conheceremos na arquitetura.

Íon: Não, certamente.

Sócrates: Portanto é assim também segundo todas as técnicas, aquilo que conhecemos através de uma técnica, não conheceremos através de outra? Mas, responda-me isso primeiro: afirmas que as técnicas diferem umas das outras?

Íon: Sim.

Sócrates: Ah, assim como eu estou conjecturando, quando uma ciência trata de umas coisas e outra trata de outras, assim eu chamo uma técnica de uma maneira, a outra de outra; e tu também fazes o mesmo?

Íon: Sim. Sócrates: Se houvesse uma ciência das mesmas coisas, por que haveríamos de dizer de uma maneira diferente da outra, cada vez seria possível saber as mesmas coisas através de ambas? Por exemplo: eu conheço que esses dedos são cinco, tu também sabes, e, como eu, tu conheces as mesmas coisas a respeito deles. E se eu te perguntasse se é pela mesma técnica que nós conhecemos isso, isto é, pela aritmética que eu e tu conhecemos as mesmas coisas ou por outra técnica, tu dirias certamente que é pela mesma.

Íon: Sim.

Sócrates: Dize-me agora, então, o que a pouco eu estava a ponto de te perguntar: se te parece que ocorre assim em todas as técnicas, isto é, que conhecemos uma mesma coisa necessariamente com uma técnica, porém nunca essa mesma coisa por outra técnica; uma vez que se trate de uma técnica diferente, é necessário que seja outro o objeto do seu conhecimento.

Íon: Assim me parece, Sócrates.

Sócrates: Portanto, aquele que não possui uma técnica não será capaz de conhecer bem nem o que se diz nem o que se faz dessa técnica?

Íon: Dizes a verdade.

DIÁLOGOS DE PLATÃO, ÍON
Tradução original de Humberto Zanardo Petrelli

Todo bom projeto começa com o pleno entendimento, por parte de uma equipe de desenvolvimento ou de um fornecedor de soluções, daquilo que o cliente realmente deseja. Isto deveria ser trivial, simples, mas deixou de ser a partir do surgimento e domínio de uma série de tecnologias aplicada ao desenvolvimento de soluções. Passamos a ter, de um lado, o usuário e, do outro, o conhecedor da tecnologia.

O usuário precisa de uma solução, mas não domina a tecnologia. Quem domina a tecnologia, por outro lado, muitas vezes desconhece a real necessidade do usuário. Um dos textos de Platão, descreve um diálogo entre Sócrates e Íon, o rapsodo (artista popular ou cantor que, na antiga Grécia, ia de cidade em cidade recitando poemas). Em certa parte do texto, Sócrates discute com Íon um poema de Homero no qual o pai recomenda ao filho cuidado em uma corrida de bigas. Um cocheiro entenderia muito melhor este texto do que um médico, ou um arquiteto, concordam os dois.

Ao dominarmos uma tecnologia não podemos ter a pretensão de entender integralmente o negócio ou as necessidades de nossos clientes. Ainda que nos esforcemos ao máximo para isto, é o cliente que detém esta compreensão, devendo ser o soberano na tomada de decisões sobre a solução que deseja, mesmo que não entenda nada da técnica a ser usada na construção desta solução.

Já mencionei, anteriormete, User Stories, ferramentas que envolvem o cliente e o fornecedor/desenvolvedor na descrição da solução. Mike Cohn, autor da clássica apresentação sobre Scrum que traduzi, é também autor de um dos melhores livros sobre User Stories: *User Stories Applied*. Resumindo ao extremo, as User Stories permitem àqueles que conhecem a técnica da construção de uma solução (linguagem de programação, ferramentas, bases de dados) guiar quem necessita desta solução no exercício de descrevê-la de forma simples e concisa.

As User Stories devem ter o usuário, aquele que necessita da solução, como o foco da história. Não se deve mencionar na história algo como "melhorar a indexação da tabela de pedidos". Isto será grego para um usuário leigo, mas um analista de base de dados entenderá desta forma quando ler, na linguagem do cliente: "aumentar a velocidade na impressão dos relatórios de pedidos". User Stories devem ser perfeitamente explicáveis em 30 segundos. Cada história deve "caber" no trabalho a ser executado em uma semana pela equipe de desenvolvimento, e deve ser facilmente testável: o cliente deve poder acompanhar o desenvolvimento do sistema lendo as User Stories que ajudou a escrever.

Usando dois acrônimos em inglês, o pessoal do xp123.com completa com mais boas dicas. *INVEST in Good Stories and SMART Tasks*.

INVEST vem de *Independent, Negotiable, Valuable, Estimable, Small, Testable*. Cada User Story deve ser, o máximo possível, **independente** de outras histórias. Aquilo que ela descreve deve ser perfeitamente **negociável** entre o fornecedor e o cliente (de fato, ela deve poder ser traduzida, posteriormente, em um plano de trabalho ou contrato de serviços). O cliente deve perceber o **valor** da User Story tanto como apoio à compreensão na construção da solução quanto na percepção real da **estimativa** de seu investimento na mesma. E como já mencionei, cada história deve ser **pequena** e **testável**.

SMART vem de *Specific, Measurable, Achievable, Relevant, Time-boxed*. Quando auxiliamos o cliente na construção de User Stories, devemos ter em mente que elas serão atendidas, na construção da solução, com a realização de uma série de tarefas. Ao saber como devem ser as tarefas, saberemos orientar melhor cada história. Cada tarefa deve ser **específica**, contida em si, bem definida e também independente. A tarefa deve poder ser **medida** para que seu custo possa ser efetivamente avaliado. Obviamente, temos que definir tarefas que possam ser **realizadas**. Cada tarefa deve ser **relevante** à história e, se na definição das tarefas, uma delas parecer não estar relacionada com a história (uma tarefa assessória não prevista inicialmente), então a tarefa ou a história devem ser revistas. Por fim, um período de **tempo deve ser definido** para cada tarefa.

Muito bem, mas que tal um exemplo? O que considero mais simples, de fácil entendimento e visualização, é o desenvolvido pela Universidade Estadual da Carolina do Norte, descrevendo o projeto de uma cafeteira.

Título: Estado de Espera		
Teste de Aceite: verificaOpções0	Prioridade: 0	Story Points: 2
Quando a cafeteira não está em uso, ela fica aguardando a seleção do usuário. Há seis diferentes possibilidades de seleção: 1) Adiciona uma Receita; 2) Apaga uma Receita; 3) Edita uma Receita; 4) Adiciona Itens ao Estoque; 5) Verifica o Estoque; e 6) Adquire uma Bebida.		

Título: Adiciona uma Receita		
Teste de Aceite: adicionaReceita1	Prioridade: 1	Story Points: 2
Apenas três receitas podem ser adicionadas à cafeteira. Uma receita consiste de um nome, preço, doses de café, doses de açúcar e doses de chocolate. Cada nome de receita deve ser único na Lista de Receitas. O preço deve ser tratado como um número inteiro. Uma mensagem de estado é exibida para mostrar se a receita foi adicionada com sucesso, ou não. Ao final, a cafeteira retorna a seu estado de espera.		

Título: Apaga uma Receita		
Teste de Aceite: apagaReceita1	Prioridade: 2	Story Points: 1
Uma receita pode ser apagada da cafeteira se ela existir na sua Lista de Receitas. As receitas são listadas por ordem de nome. Ao final, uma mensagem de estado é exibida e a cafeteira retorna a seu estado de espera.		

Título: Edita uma Receita		
Teste de Aceite: editaReceita1	Prioridade: 2	Story Points: 1
Uma receita pode ser editada se ela existir na Lista de Receitas da cafeteira. As receitas são listadas por ordem de nome. Após selecionar uma receita para editar, o usuário digita a nova informação da mesma. Ao final, uma mensagem de estado é exibida e a cafeteira retorna a seu estado de espera.		

Título: Adiciona Itens ao Estoque		
Teste de Aceite: adicionaEstoque1	Prioridade: 1	Story Points: 2
Itens de estoque podem ser adicionados à cafeteira a partir do menu principal, sendo adicionados ao estoque atual da mesma. Os tipos de itens são café, leite, açúcar e chocolate. O estoque é medido em números inteiros. Itens de estoque só podem ser removidos da cafeteira na aquisição de uma bebida. Ao final, uma mensagem de estado é exibida e a cafeteira retorna a seu estado de espera.		

Título: Verifica o Estoque		
Teste de Aceite: verificaEstoque	Prioridade: 2	Story Points: 1
O estoque pode ser verificado a qualquer momento a partir do menu principal. Ao final, a cafeteira retorna a seu estado de espera.		

Título: Adquire uma Bebida		
Teste de Aceite: adquireBebida1	Prioridade: 1	Story Points: 2
O usuário seleciona uma bebida e insere uma quantidade de dinheiro. A quantidade de dinheiro deve ser um número inteiro. Se a bebida consta do Livro de Receitas e o usuário inseriu uma quantidade de dinheiro suficiente, a bebida será entregue junto com a quantidade de troco (se aplicável). O usuário não poderá adquirir uma bebida se não depositar dinheiro suficiente na cafeteira. O dinheiro do usuário será devolvido se não houver estoque suficiente para fazer sua bebida. Ao final, uma mensagem de estado sobre a compra é exibida e a cafeteira retorna a seu estado de espera.		

Na página do projeto da cafeteira[88] da Universidade Estadual da Carolina do Norte estas User Stories transformam-se em Casos de Uso, Diagramas de Sequência e em códigos de programas em várias linguagens. Recomendo fortemente esta leitura.

O jogo da estimativa

> *Estou bastante convencido que, uma vez que ele foi além das questões que podem ser respondidas através de um inquérito razoável, o projetista deve chutar ou, se você preferir, postular um conjunto completo de atributos e valores, com uma frequência imaginada de distribuições, a fim de desenvolver modelos completos, explícitos e compartilhados de usuário e usos.*
>
> > *Um chute calculado é melhor do que uma premissa não verbalizada.*
>
> *Muitos benefícios provêm deste procedimento "ingênuo":*
>
> *Chutar os valores e frequências força o projetista a pensar muito cuidadosamente sobre a composição esperada de usuários.*
>
> O PROJETO DO PROJETO – DA MODELAGEM À IMPLEMENTAÇÃO
> FRED BROOKS, JR

88 http://agile.csc.ncsu.edu/SEMaterials/tutorials/coffee_maker/

Você deve ter observado nas User Stories da cafeteira, acima, uma célula contendo *Story Points*. Os Story Points, ou Pontos de História, correspondem a um "número mágico" que reflete o esforço necessário para o desenvolvimento de cada uma das partes específicas que irão compor a solução completa. A estimativa deste esforço é uma das partes mais difíceis da elaboração de um projeto – se é que não é a parte mais difícil. Dela irão depender a alocação de pessoas, materiais, equipamentos e também é ela que definirá, em grande parte, o valor necessário para o investimento no desenvolvimento da solução.

Já falamos sobre a dificuldade de se estabelecer a quantidade de horas necessárias para completar uma determinada tarefa, já que cada pessoa e equipe trabalham de formas distintas, com níveis diferentes de elegância e qualidade. No fim das contas, é o chute calibrado do projetista que irá definir a quantidade necessária de esforços para a conclusão de cada uma das tarefas. Este chute é tão mais calibrado quanto maior for o seu histórico anterior de projetos. É a experiência que acaba "calibrando" a percepção para o bom chute. Segundo Mike Cohn[89]:

> *O Scrum não prescreve uma forma através da qual as equipes estimam seu trabalho. Entretanto, ele incentiva as equipes a não fazer estimativas em termos de horas mas, ao invés disto, a usar uma métrica mais abstrata para quantificar o esforço. Métodos comuns de estimativa incluem escalas numéricas (1 a 10), números de camisetas (PP, P, M, G, GG, GGG, XG, XXG), a sequência de Fibonacci (1, 2, 3, 5, 8, 13, 21, 34, etc) e até mesmo raças de cachorros, na qual o chihuaua representa as histórias menores e um dinamarquês a maior. O importante é que a equipe compartilhe o entendimento da escala que usa, de forma que cada um de seus membros esteja confortável com seus valores.*

Isto não é fácil. Uma grande parte dos clientes, acostumados ao Grande Contrato, ainda tem sua percepção de esforços necessários ao desenvolvimento fortemente atrelada a um número de horas. Mesmo quem está gerenciando o desenvolvimento de um projeto terá, na maioria das vezes, que remunerar seus desenvolvedores em alguma forma de pagamento por dias ou horas de trabalho. Há aqui um paradigma estabelecido e que precisa ser mudado, já que o número de horas para o desenvolvimento de um projeto tem muito pouco significado e uma relação muito frágil com a qualidade final do projeto.

Uma linha de código escrita em dois minutos por um desenvolvedor com muitos anos de experiência e enorme criatividade tem mais valor do que 100 linhas de código escritas em dez dias por um outro desenvolvedor com muito menos experiência? Considere que o resultado produzido pelo trabalho dos dois desenvolvedores é idêntico e que você não consegue identificar, sem olhar o código, quem produziu uma solução ou outra. Agora, imagine que você não tem dez dias em seu cronograma para entregar o resultado da tarefa. Há uma subjetividade muito grande na estimativa de esforços, recursos para a execução da tarefa e valores que serão cobrados do cliente. Pense também em um profissional que dedica cinco horas de seu dia ao seu trabalho de desenvolvimento e outras três a seu aprimoramento profissional através de cursos, leituras, conversas com seus pares. Como embutir estas três horas diárias no valor que cobra de seus clientes? Somado a todos estes fatores está a concorrência. Independente de qualquer estimativa, o valor cobrado de um cliente deve ser competitivo o suficiente para que, considerando a relação entre custo e benefício, ele invista em nosso trabalho e não no trabalho de outros. Isto inverte completamente a equação da estimativa: temos que fazer com que o que estimamos "caiba" no orçamento disponível para o investimento do cliente.

Não existe bala de prata. Por isso, a melhor saída é realmente termos um método de estimativa de esforços com o qual a equipe esteja confortável e, a partir dele, criarmos a nossa própria relação entre o orçamento do cliente de um lado e o que pagamos a desenvolvedores e investimos em recursos (maquinário, capacitação, etc) de outro.

Uma forma de estimativa que eu acho especialmente útil é atribuir duas propriedades a cada tarefa:

89 http://scrummethodology.com/scrum-effort-estimation-and-story-points

o valor do negócio (BV – Business Value) definido pelo cliente do projeto; e a carga de trabalho (W – workload), que é definida pelos desenvolvedores. A prioridade de cada tarefa é o valor da divisão BV/W. BV e W não são valores exatos, eles não representam horas ou dias, mas os números 1, 2, 3, 5, 8, 13, 21 (a seqüência de Fibonacci). “É muito difícil avaliar exatamente uma tarefa. Então, para que seguir usando horas? É mais fácil dizer que a tarefa A é mais fácil que a tarefa B e atribuir cargas de trabalho proporcionais”, segundo os autores do Scrinch[90], uma ferramenta auxiliar ao desenvolvimento de projetos com o Scrum.

Já dissemos que o cálculo de horas para uma determinada tarefa é, sempre, muito complexo. Ele exige que se conheça muito bem a tecnologia com a qual se trabalha e a real produtividade de cada membro da equipe. Como temos que estar sempre de olhos abertos a novas tecnologias e como os membros de uma equipe podem não ser os mesmos no decorrer de todo o tempo da realização de um projeto, que também pode ter seus requisitos ajustados, as variáveis para o cálculo de horas são tantas que essa ideia de usar a sequência de Fibonacci como um fator de relação entre tarefas, ao invés da atribuição inexata de um número de horas, não parece tão absurda. "É um tanto incômodo no início, mas bastante útil no final", ainda segundo os autores do Scrinch.

Essa prática de atribuir números da sequência de Fibonacci a uma tarefa tem suas origens no Poker do Planejamento do Scrum, um jogo - e ao mesmo tempo um exercício - de estimativa de desenvolvimento de um projeto a partir da informação que se tem do cliente: as User Stories.

No Poker do Planejamento, cada membro da equipe fica com um conjunto de cartas numeradas de acordo com a sequência de Fibonacci. Todos analisam as User Stories, uma por vez, e jogam na mesa uma carta de acordo com o que acreditam ser o grau de complexidade de seu desenvolvimento. Quanto mais complexa a tarefa, maior o número de sua carta. A média dos valores das cartas corresponde aos pontos para aquela User Story, seus Story Points. Caso, por exemplo, a maioria dos membros da equipe jogue uma carta de valor alto e apenas um coloca uma carta de valor baixo (ou vice-versa), a carta discrepante é descartada da média. Há ainda uma carta com o sinal de interrogação (?), significando que a User Story não está suficientemente explicada ou não foi completamente entendida.

Isto é extremamente subjetivo, mas dá a real dimensão comparativa entre a complexidade de desenvolvimento entre uma história e outra. Com a prática é possível estabelecer relações entre os Story Points e o valor a ser cobrado do cliente e o que será pago aos desenvolvedores.

Mike Cohn criou o PlanningPoker.Com, que permite que se faça a estimativa via internet, de graça, desde que se faça o registro no site.

O Planning Poker é legal para fomentar a interação entre a equipe de uma maneira divertida, servindo também para o ScrumMaster e o Product Owner terem uma ideia dos perfis de estimativa de cada um dos membros. Sempre existirá aquele que é mais ousado e acha possível fazer tudo muito fácil, assim como, no outro extremo, terá aquele que sempre vai encontrar pontos que devem ser melhor esclarecidos em cada história. Na prática, o Planning Poker funciona como um nivelador para a equipe, para aumentar o entrosamento e para ensinar o Scrum. Depois de alguns exercícios, o Planning Poker torna-se intuitivo e as cartas acabam se aposentando.

O pessoal bacana na Blusoft disponibilizou o conjunto de cartas do Poker do Planejamento[91] para download, inclusive em um formato editável no Corel, para que cada um possa modificar as suas. Há também o formato em pdf, para os que não querem ter esse trabalho e podem fazer uma merecida propaganda para a empresa.

Caso você tenha um bom histórico de seus projetos anteriores, experimente fazer uma retrospectiva deles, tratando-os como novos e estimando-os novamente com o Poker do Planejamento. Revise ou

90 http://scrinch.sourceforge.net

91 http://bluesoft.wordpress.com/2007/11/07/scrum-planning-poker/

crie as User Stories respectivas e estabeleça os Story Points para cada uma delas. Avalie a remuneração dos desenvolvedores, o quanto foi cobrado do cliente e o lucro obtido. Faça isto para uns três ou quatro projetos e tente estabelecer uma relação entre os valores financeiros e os Story Points. Aplique esta relação a um outro projeto conhecido e verifique se ela é real. Repetindo o que diz o pessoal do Scrinch: "É um tanto incômodo no início, mas bastante útil no final."

Ah, o cheirinho de Scrum pela manhã!

É justamente no dia-a-dia que o Scrum vai tornando-se mais importante. Depois de algum tempo praticando, as ferramentas vão tornando-se um hábito. Em nossa empresa é muito comum, já em uma primeira reunião com um cliente, começarmos a pensar nos termos do Product Backlog, imaginando como será a formação da equipe e parceiros de desenvolvimento e nos Sprints que ocorrerão até a entrega do projeto. Quando a proposta é feita, muitas vezes em conjunto com um ou mais parceiros de negócios, ela traz embutida o Product Backlog e um cronograma que dará origem ao Burndown Chart. O perigo de uma coisa que se torna cotidiana, porém, é que essa coisa pode começar a se deteriorar sem que a gente perceba. É preciso prestar atenção aos cheiros do SCRUM pra ver se não há algo de podre no reino da Dinamarca, como diria Hamlet.

Quem usou pela primeira vez o termo *Code Smells* (o fedor do código) foi o Kent Beck, criador do Extreme Programming, enquanto ajudava o Martin Fowler em seu livro sobre *Refactoring*. Segundo Martin, "um método muito extenso é um bom exemplo - apenas de olhar para um código meu nariz começa a se contorcer se eu vejo mais do que uma dúzia de linhas em Java".

A analogia com o "cheiro" é bastante simples. Muitas vezes sabemos que há algo errado, algo está cheirando mal, mas nem sempre é fácil identificar a origem. Mike Cohn, que muitos já notaram ser uma das minhas fontes preferidas, propôs um catálogo de cheiros do SCRUM em 2003, buscando auxiliar na identificação de alguns problemas:

- **Perda de ritmo**: acontece quando os Sprints começam a ter duração variável. É muito importante para o sucesso do Scrum que a equipe adquira a naturalidade em seu ritmo, concluindo cada Sprint com algum incremento ao projeto. Se, por qualquer razão, um Sprint tiver a duração de uma semana, outro de duas, mais outro de uma e assim por diante, será impossível que o ritmo adquira uma naturalidade. Eu vejo a tendência disso acontecer quando o time é formado por "porcos" que têm muitos outros compromissos. Porcos são aqueles que entram com o "toucinho" no projeto, os que devem estar verdadeiramente comprometidos. O ScrumMaster deve considerar a duração do Sprint algo sagrado e, se for o caso, conversar com o patrocinador do projeto exigindo a participação do devido porco ou até sugerindo a troca de porcos.
- **Galinhas falantes**: na reunião diária do projeto, apenas os porcos podem falar. As galinhas, que entram com os ovos, estão envolvidas, mas não comprometidas com o processo. O Mike é totalmente contra permitir que as galinhas se intrometam, mas concorda que elas podem assistir às reuniões. Eu concordo, mas também aceito que, na informalidade, uma galinha tente transmitir a um porco suas ideias, desde que fora das reuniões. Em raros casos, e sempre entre o fim de um Sprint e o começo de outro, aceito a indicação de um porco para promover uma galinha a porco também.
- **Porcos que faltam**: cada vez mais desenvolvedores trabalham em horários flexíveis. Isto é um problema quando se quer estabelecer um ritmo e tem a ver com o primeiro dos problemas aqui colocados. As reuniões diárias devem ser rápidas, de quinze minutos, aproveitando ao máximo o tempo dos participantes. Os porcos escolhidos devem saber disso e estarem de acordo logo no início do projeto. O porco que faltar estará sujeito às decisões dos demais, sem poder negociar nada posteriormente. Mas, e com equipes que trabalham

remotamente? Aí eu sou mais flexível, mesmo reconhecendo que essa flexibilidade é, às vezes, contraproducente e que o ScrumMaster acaba tendo a sobrecarga de ficar dando puxões de orelha aqui e ali. Nesses casos, o que faço é ter vivo um documento compartilhado, onde as decisões têm seu horário de fechamento. Passado esse horário, quem não contribuiu sujeita-se às decisões dos demais.

- **Hábitos persistentes**: é normal que cada grupo comece um projeto trazendo para ele seus hábitos de trabalho individuais e em equipe mas é importante que, quando um mau hábito é identificado, ele seja imediatamente corrigido. Um dos termômetros do Sprint é o Burndown Chart, o quadro de registro do consumo de horas do projeto. Quando as horas efetivamente consumidas flutuam muito com relação às horas previstas, há algo de errado no planejamento ou na execução do projeto.
- **O ScrumMaster delega trabalhos**: especialmente em equipes ainda não habituadas ao Scrum, é normal que exista uma expectativa que o ScrumMaster, como um gerente de projetos, distribua os trabalhos a serem feitos. Isto é errado. O ScrumMaster auxilia a equipe, removendo os obstáculos à execução do projeto. Cada membro da equipe deve escolher as tarefas que executará, de acordo com a sua competência e negociando com os demais membros do grupo.
- **A reunião diária é para o ScrumMaster**: também um sintoma clássico da falta de familiaridade com o Scrum, manifesta-se com a equipe fazendo para o ScrumMaster o relato dos trabalhos. A reunião tem que ser para a equipe. Quando um membro compromete-se em entregar uma tarefa, ele compromete-se com a equipe e não com o ScrumMaster. Não é papel do ScrumMaster reclamar da entrega ou não de tarefas, mas entender os obstáculos que impediram a sua realização e removê-los.
- **Cargos especializados**: acontece quando as pessoas entram na equipe com cargos de "testador", "chefe de desenvolvimento" e outros. Quando alguém entra com o papel de "testador", por exemplo, os demais podem ter a impressão de estar "liberados" de participar de testes do projeto. Assim como um "chefe" de qualquer coisa pode dar uma falsa impressão de hierarquia. Dentro de uma equipe Scrum, todos devem ter o espírito "estamos nessa juntos" e assumir responsabilidades idênticas perante o projeto. Claro que vai ser impossível montar uma equipe formada apenas por generalistas, que saibam tudo de tudo, mas o espírito é esse. O que eu procuro fazer é eliminar os títulos e cargos na apresentação da equipe, observar a escolha de tarefas de cada um (tipicamente, dentro das áreas de maior conforto) e tentar incentivar a colaboração especialmente quando há a identificação de obstáculos para os quais sinto que "o olhar de um outro colega" pode trazer nova luz às dificuldades. Claro que sempre tomando o cuidado de não delegar tarefas.

Perguntas e Respostas

Durante o período em que adquiríamos familiaridade com o Scrum durante o projeto do Interoplab, com a equipe da Unicamp, os membros da equipe (em especial o Raul Kist e o Bruno Melo) compilaram dúvidas que foram discutidas ao longo do projeto. Destas, algumas têm se apresentado com mais frequência em outras implementações do Scrum e, por isso, as reproduzo aqui.

Pergunta 1: Para utilizar o Scrum você precisa saber aonde quer chegar. E quando pegamos aqueles clientes que não sabem o que querem e constantemente tiram e colocam as mesmas coisas em um projeto? Outra pergunta tão inquietante é: como precifica-se um projeto em algo que não sabemos o que vai virar o fim, sem saber direito o tamanho do projeto? (já que prioridades e requisitos são desbravados aos poucos, em fases).

Um projeto novo, sendo interno ou externo, começa com os "user stories", com foco total

em quem usará o sistema (ou uma nova funcionalidade dele) a ser desenvolvido. A partir destas "histórias", estima-se o tempo de desenvolvimento e os recursos necessários, que entram no Product Backlog e no Burndown Chart.

A partir do Product Backlog é que estima-se o preço do projeto. Mas aí há uma questão: O Scrum (assim como o XP e metodologias ágeis em geral) são para projetos de curta a média duração. Três meses é o período mágico máximo. Se um projeto durar mais do que três meses, ele deve ser dividido em dois ou mais projetos. Ficando nestes três meses, a precificação e a "visão" do que será feito no período é relativamente fácil. Se passar disto, começa o exercício do "achismo". Em metodologias mais clássicas, independente do tempo que o projeto levará, uma fase extensa de planejamento irá determinar recursos e cronogramas, com o efeito colateral de que a própria fase de planejamento irá adicionar tempo e custos ao projeto, coisas que as metodologias ágeis buscam evitar – na metodologia ágil, o planejamento e controle não devem ser sobrecarga no projeto. Eu busco resolver isto da seguinte forma: começo, sim, com os "user stories", tratando-os de forma mais ampla, como uma "lista de desejos". Com base também em projetos anteriores (ajuda se eles existirem), estimo o tempo de desenvolvimento e multiplico por dois (sim! e isto é uma recomendação do Fred Brooks Jr. em *O Mítico Homem-Mês*). Faço um Product Backlog prévio, que incluirá, de forma geral, o que irá compor os demais Product Backlogs, e aí precifico. Isto antes de começar o projeto e o primeiro Product Backlog e Burndown Chart. Mas claro que o melhor é sempre manter o projeto no escopo visível de uns poucos meses.

O Burndown Chart simplesmente lista as horas a serem consumidas pelos recursos. Pode ser montado de forma global (prefiro) ou para cada recurso a ser consumido. O ideal é que as horas previstas sejam equivalentes às consumidas. Se as consumidas forem menores, menos mal. Se maiores, o projeto está consumindo recursos além dos que foram propostos. Ele serve, historicamente, para avaliar nossa capacidade de prever os recursos, e como guia para que a ajustemos.

Temos que saber o que vai dar no fim sim, independente de mudanças de prioridades posteriores. O "fim" tem que estar descrito no Product Backlog inicial.

Pergunta 2: O Scrum utitiliza uma teoria de membros auto gerenciáveis, onde os esforço do ScrumMaster é quase unicamente facilitar e dar subsídios para a equipe (e não necessariamente resolver os problemas existentes). Mas existem aqueles casos em que a equipe, nem por reza brava, entra em acordo. O Scrum tem alguma saída nesse caso, dado que o ScrumMaster não deve "gerenciar" estas pessoas?

Se for um conflito "técnico", o ScrumMaster resolve, em último caso, ditatorialmente mesmo. Se for algo relativo à funcionalidade, aí é o dono do projeto (patrocinador, pagante) quem decide.

Pergunta 3: O Scrum leva em consideração equipes que tem contato constante. Mas se isso não for tão ideal? Se os contatos (por horários, por exemplo) não forem tão acopláveis?

O contato "físico" imprescindível são as "stand up meetings", que deveriam ser diárias e com a duração máxima de 15 minutos. Em alguns casos, isto pode ser inviável. Aí vale-se de meios eletrônicos mesmo, como um diário de bordo em que, assincronicamente, um possa verificar e interferir no trabalho do outro. Um "wiki" ou um documento no Google Docs podem ajudar neste sentido. Aí já não é da metodologia, mas da aplicação prática dela.

Eu e a Joice, minha sócia, trabalhamos, quase sempre, um turno por dia juntos. Ao menos um dia por semana reservamos para o nosso “dia integral” no qual, além de fazer uma discussão mais ampla de nossas atividades, estado dos clientes e planejamento futuro e trabalharmos naquelas tarefas que requerem a presença física de ambos ainda exercitamos

nossos dotes culinários na “cozinha experimental da BrodTec”. Tenho notado que uma atividade “extra-curricular” como a culinária, um trabalho voluntário conjunto ou até mesmo um passeio (quando aproveitamos a visita à clientes remotos para um eventual “turismo acidental”) são importantes para aumentar os laços e compromissos em qualquer equipe.

As "stand-up meetings" na BrodTec ocorrem naturalmente e mais do que uma vez por dia. Por vezes até pensamos em normalizar isto, oficializando um horário, mas como está sendo produtivo assim, mantemos desta forma. Além disso, temos uma "Lista de Tarefas" no Google Docs, acessível por ambos, onde registramos as coisas mais importantes a serem feitas, quer estejamos trabalhando juntos ou separados. Na nossa "lida", essa lista acabou virando um borrão eficaz de um Sprint Backlog. E sempre vale a máxima do Extreme Programming: se a metodologia não couber ao trabalho, adequa-se a metodologia e não o trabalho que está dando certo. Claro que a metodologia ajuda a avaliar se o trabalho está, ou não, dando certo.

Pergunta 4: Em uma empresa que desenvolve projetos internos (para si mesma) e externos (para clientes), cheguei a uma conclusão: Scrum é mais eficiente em projetos internos. Digo isso pois, os riscos de um projeto interno normalmente são menores (nem sempre, admito) mas principalmente porque cliente (própria empresa), ScrumMaster e time de desenvolvimento estão juntos, com contato mais próximo. Estou errado? Até porque um sprint pode não sair se demasiadamente uma atividade depender do cliente, e o cliente resolver sumir por algum tempo.

Quando se adota o Scrum, o cliente deve estar envolvido e estar sempre presente. No mínimo, ao alcance de um e-Mail ou telefone. Não pode "sumir". Por isto deve estar ciente disto. Quanto a funcionar melhor para projetos internos ou externos, acho que não faz diferença, mas isto não com base em minha experiência, mas na documentação de outros autores.

Ferramentas

Não há uma prescrição específica de ferramentas a serem adotadas com o Scrum, mas especialmente para a sua prática e a implementação de seus artefatos há algumas disponíveis na web que podem ser de alguma ajuda. Uma ferramenta, porém, é imprescindível: o quadro branco. Adiante falarei sobre o quadro branco e outras ferramentas que fomos utilizando ao longo do tempo.

Quadro branco e kanban

Uma equipe Scrum deve poder visualizar, o tempo inteiro, a evolução de suas tarefas e a sua agenda. Um *kanban* (a palavra significa “registro” em japonês) consiste em uma tabela simples com as tarefas planejadas na coluna à esquerda, tarefas em execução na coluna do meio e tarefas completas na coluna à esquerda. Uma implementação típica de um kanban é um conjunto de etiquetas adesivas coloridas que são coladas e movidas entre as colunas de um quadro branco de acordo com a evolução de cada tarefa. As cores das etiquetas podem ser usadas para indicar a prioridade das tarefas, o responsável por ela ou ambos. Os demais espaços do quadro branco podem conter a lista de obstáculos ou impedimentos a serem levados ao ScrumMaster na próxima reunião, a agenda da equipe e outras informações pertinentes ao projeto. Uma “sala Scrum” pode ter uma parede pintada com tinta impermeável, onde as colunas do kanban possam ser desenhadas e as anotações possam ser feitas e apagadas. Outras paredes podem ter as cópias dos backlogs para o acompanhamento da equipe.

Caso a equipe trabalhe remotamente, uma simples tabela compartilhada no Google Docs pode fazer às vezes de um quadro branco compartilhado entre todos os membros, mas há ferramentas livres específicas para este tipo de implementação, como o heijunja-kanban (http://heijunka-

kanban.sourceforge.net). Através dele é possível acompanhar as tarefas desde que são solicitadas, até a sua configuração, execução, teste e entrega ao cliente.

Google Docs

O Google Docs (http://docs.google.com) tem se mostrado, na nossa empresa, uma excelente ferramenta para o acompanhamento e compartilhamento de informações em um projeto Scrum, tanto que não limitamos seu uso a trabalhos realizados com equipes remotas, usando-o também em nossas tarefas diárias. As várias folhas de uma planilha podem ser usadas para a criação do Product Backlog e dos vários Sprints Backlogs. Uma folha adicional pode servir como um kanban e mais outra como um Burndown Chart.

Abaixo o exemplo de um Burndown Chart, com seus dados, em uma planilha compartilhada no Google Docs, que pode ser acessada neste endereço: http://miud.in/e2V

Burndown Chart

Tarefas | Resumo | Chart

Colaborador	Tarefas	Semana 1 (Estimada)	Semana 1 (Realizada)	Tarefas	Semana 2 (Estimada)	Semana 2 (Realizada)	Tarefas	Semana 3 (Estimada)	Semana 3 (Realizada)	Tarefas	Semana 4 (Estimada)	Semana 4 (Realizada)
Vinícius de Moraes	Desenho da interface	40	60	Aplicação do leiaute à interface	40	40		0		Adequação do leiaute de acordo com o feedback do cliente	20	20
Tom Jobim	Obtenção dos elementos que compõem o leiaute e adequação aos formatos padrão	20	15	Criação do paper prototype para testar com o cliente	40	20	Testes do paper prototype junto ao cliente. Relatório de adequação para o leiaute	20	30	Entrega ao cliente da versão funcional para testes e breve capacitação	40	35
Baden Powel	Montagem da infra (servidor, base de dados, core do CMS)	20	20	Criação da estrutura para customizações no CMS	10	15		0		Conclusão da estrutura necessária à entrega ao cliente (publicação e testes)	40	35
Total		**80**	**95**		**90**	**75**		**20**	**30**		**100**	**90**

Repare nas abas Tarefas, Resumo e Chart. Em Tarefas você têm a descrição das tarefas para cada desenvolvedor, com suas horas estimadas e efetivamente realizadas. Como vimos antes, ao invés de horas poderíamos ter usado Story Points. Repare nas fórmulas usadas na planilha Tarefas, abaixo:

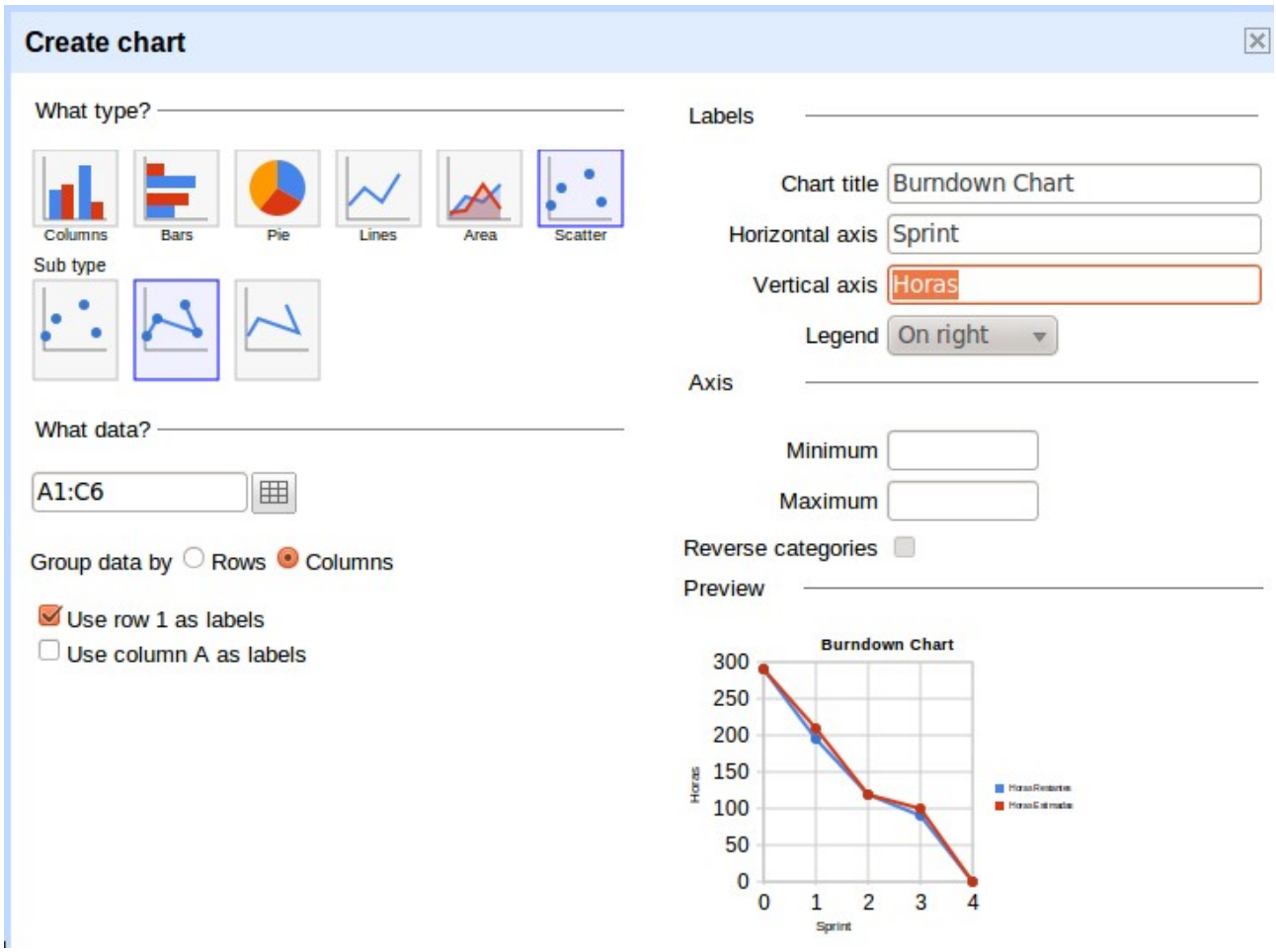
	A	B	C	D	E	F	G	H	I	J	K	L	M
1	**Colaborador**	**Tarefas**	**Semana 1 (Estimada)**	**Semana 1 (Realizada)**	**Tarefas**	**Semana 2 (Estimada)**	**Semana 2 (Realizada)**	**Tarefas**	**Semana 3 (Estimada)**	**Semana 3 (Realizada)**	**Tarefas**	**Semana 4 (Estimada)**	**Semana 4 (Realizada)**
2	Vinícius de Moraes	Desenho da interface	40	60	Aplicação do leiaute à interface	40	40		0		Adequação do leiaute de acordo com o feedback do cliente	20	20
3	Tom Jobim	Obtenção dos elementos que compõem o leiaute e adequação aos formatos padrão	20	15	Criação do paper prototype para testar com o cliente	40	20	Testes do paper prototype junto ao cliente. Relatório de adequação para o leiaute	20	30	Entrega ao cliente da versão funcional para testes e breve capacitação	40	35
4	Baden Powel	Montagem da infra (servidor, base de dados, core do CMS)	20	20	Criação da estrutura para customizações no CMS	10	15		0		Conclusão da estrutura necessária à entrega ao cliente (publicação e testes)	40	35
5	**Total**		**=SUM(C2:C4)**	**=sum(D2:D4)**		**=sum(F2:F4)**	**=sum(G2:G4)**		**=sum(I2:I4)**	**=sum(J2:J4)**		**=sum(L2:L4)**	**=sum(M2:M4)**

A planilha na aba Resumo é construída automaticamente, a partir dos dados da aba Tarefas. O gráfico, por dua vez, é construído automaticamente a partir dos dados da aba Resumo. Ou seja, a única planilha que você alterará durante a evolução de seus Sprints é a de Tarefas. Todas as demais serão automaticamente atualizadas. Abaixo a planilha Resumo e suas fórmulas:

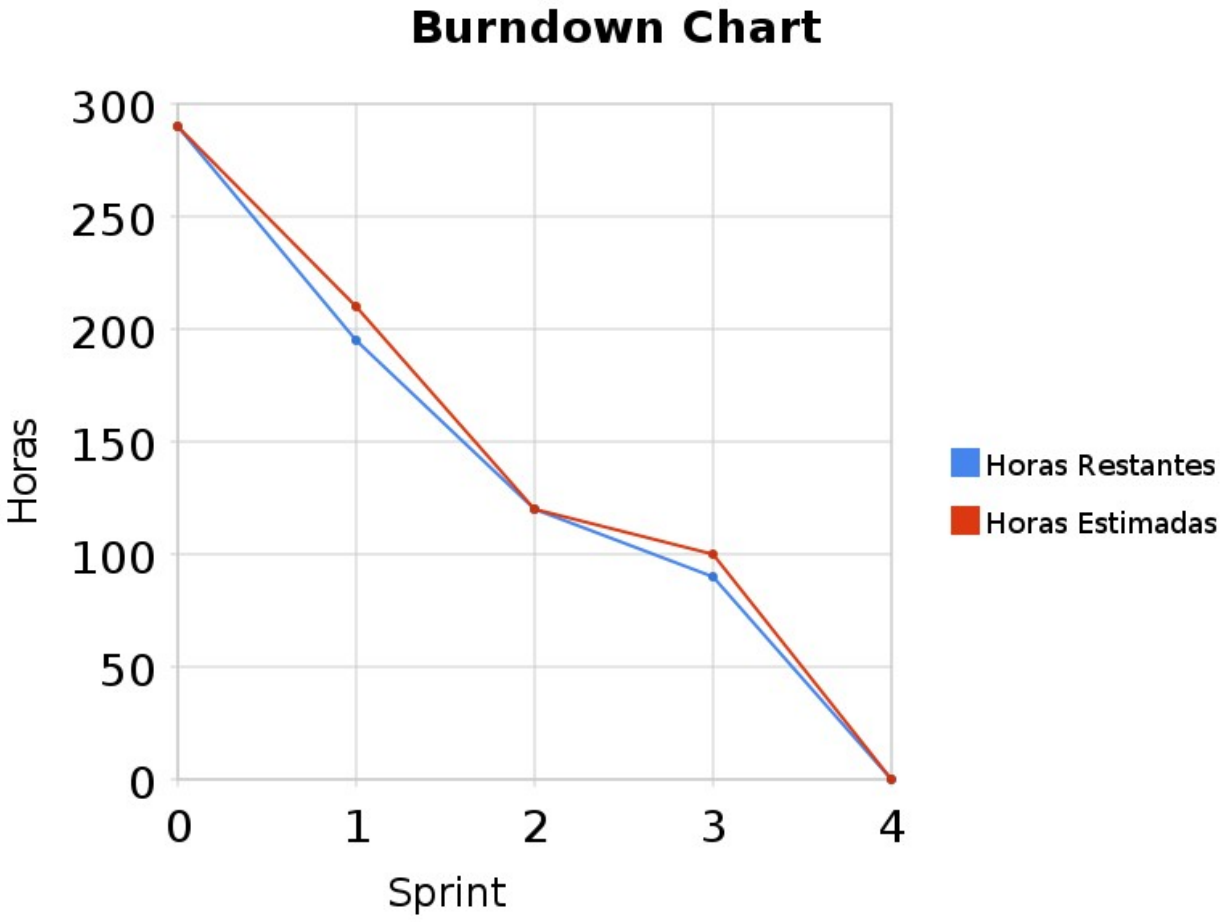
	A	B	C
1	Semana	Horas Restantes	Horas Estimadas
2	0	290	290
3	1	195	210
4	2	120	120
5	3	90	100
6	4	0	0

	A	B	C
1	Semana	Horas Restantes	Horas Estimadas
2	0	=C2	=sum(Tarefas!C5,Tarefas!F5,Tarefas!I5,Tarefas!L5)
3	1	=B2-Tarefas!D5	=sum(Tarefas!F5+Tarefas!I5+Tarefas!L5)
4	2	=B2-Tarefas!D5-Tarefas!G5	=sum(Tarefas!I5+Tarefas!L5)
5	3	=B2-Tarefas!D5-Tarefas!G5-Tarefas!J5	=Tarefas!L5
6	4	=B2-Tarefas!D5-Tarefas!G5-Tarefas!J5-Tarefas!M5	0

Para construir o gráfico, basta você selecionar todas as células da planilha Resumo, incluindo seu título, e clicar em Inserir → Gráfico (Insert → Chart, caso você esteja utilizando o Google Docs em inglês). Selecione as opções de acordo com o esquema abaixo e está pronto o seu Burndown Chart.



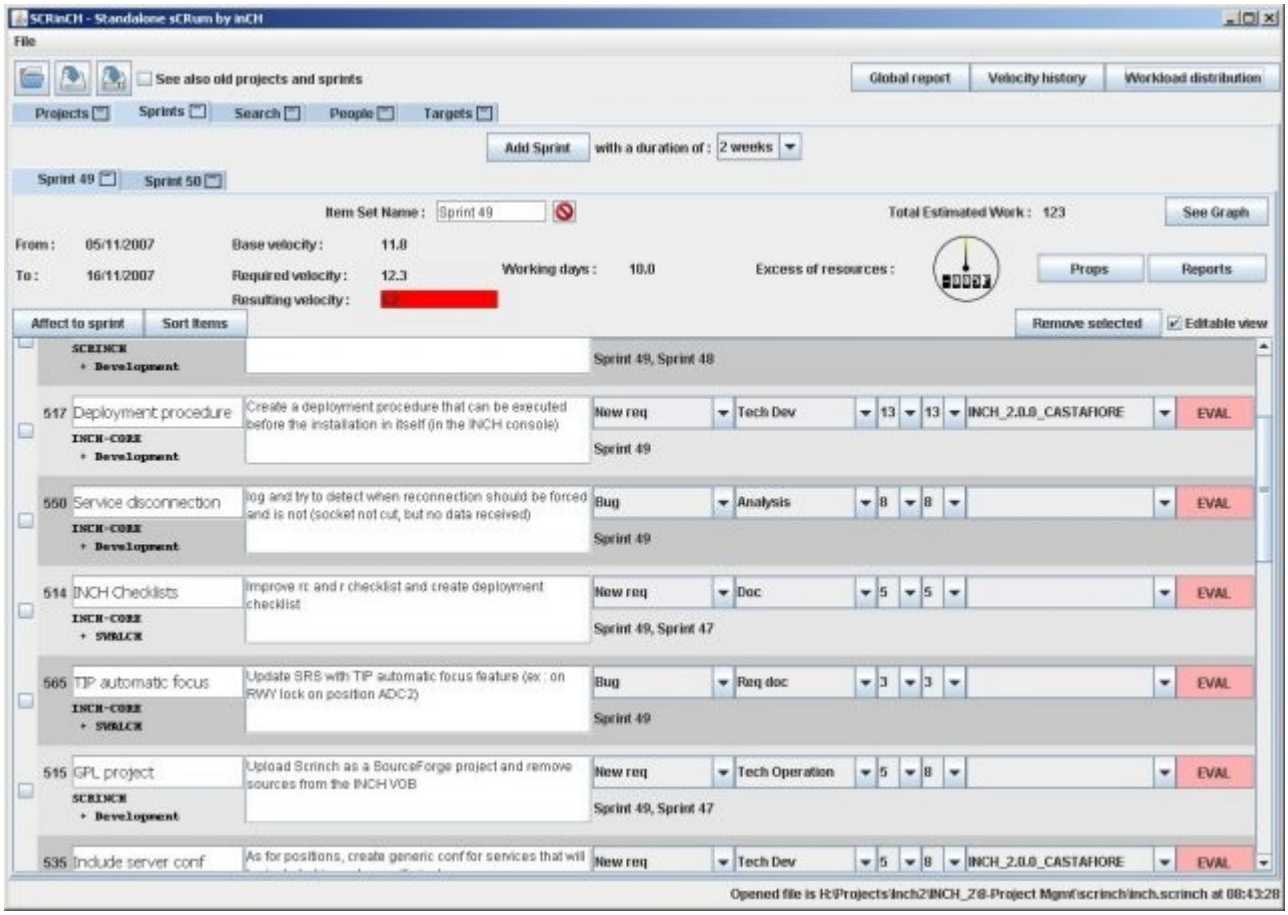
O resultado, em nosso exemplo, é o seguinte:



Scrinch

Aqui no Brasil não estamos muito familiarizados com dois personagens que fazem parte do imaginário natalino norte-americano: Ebenezer Scrooge e o Grinch. O primeiro, criado por Charles Dickens em 1843, é um velho rabugento e pão-duro. O segundo, criado pelo Dr. Seuss, é uma espécie de hermitão que odeia os Whos, habitantes de uma pequena vila ao pé da montanha onde fica a caverna onde mora. Ambos têm em comum o fato de odiarem o Natal. Daí, alguém que é um Scrinch, mistura de Scrooge com Grinch, é alguém que odeia MUITO o Natal.

Toda a vez que falo sobre Scrum, dentre as várias perguntas que me fazem, uma delas é: "Há algum software que ajude no controle e implantação desses processos?". Minha resposta típica sempre é: "um monte de post-its, um quadro branco, planilhas e documentos que possam ser compartilhados." Já dei exemplos de Product Backlog, Sprint Backlog e Burndown Chart que se valem desse tipo de documentos. Uma busca por Scrum no Freshmeat ou no SourceForge também aponta para uma série de programas que podem ser úteis, boa parte deles baseados em uma interface web, com maior ou menor grau de complexidade. Um dos que mais gostei chama-se, justamente, Scrinch.



O programa é escrito em Java e, para começar a explorá-lo basta fazer o download diretamente de sua página oficial: http://scrinch.sourceforge.net. Por ser escrito em Java, o Scrinch é multiplataforma por natureza e poderá ser utilizado no sistema operacional de sua preferência. Se você tiver o Java Web Start instalado em sua máquina, você não precisará de nenhum esforço adicional para fazer o Scrinch rodar, caso contrário, siga as instruções na página do projeto. O manual, que encontra-se junto ao pacote de download do Scrinch, é bastante prático e sucinto. Os relatórios (que incluem gráficos) gerados para o acompanhamento do projeto podem ser exportados para arquivos em PDF, facilitando o compartilhamento de informações. Um projeto exemplo também está disponível para os preguiçosos que, como eu, querem ver rapidinho tudo o que o programa é capaz de fazer antes de começar a brincar com ele.

Para aqueles que estão começando a se aventurar com o Scrum, o Scrinch e seu manual podem ser um bom ponto de partida, mantendo todos os artefatos e controles do processo em um ambiente único e consolidado. Para os que já conhecem o processo, ainda assim, vale a pena dar uma olhada na ferramenta e seus exemplos.

Bibliografia

Brooks, Frederick P. Jr. O Projeto do Projeto – da modelagem à implementação

Aquele com a história da Apple que o Rubens me emprestou

Grahamm, Paul. Hackers and Painters - conferir

BROOKS, F. P. **The Mythical Man-Month: essays on software engineering.** Boston, MA: Addison Wesley, 1995. ISBN 0201835959.

CASTELLS, M. **A galáxia da Internet: Reflexões sobre Internet, Negócios e Sociedade**. Rio de Janeiro: Jorge Zahar Editor, 2003. ISBN 9788571107403.

CONLON, M. P. **An Examination of Initiation, Organization, Participation, Leadership, and Control of Successful Open Source Software Development Projects.** Slippery Rock, PA: Information Systems Education Journal, 2007. Disponível em: <http://www.isedj.org/5/38/ISEDJ.5(38).Conlon.pdf >. Acesso em: 5 mai. 2009.

FABERLUDENS, Instituto de Design de Interação. Disponível em: <http://www.faberludens.com.br/pt-br/node/470>. Acesso em 05 mai. 2009.

FIORINI, S. T., et al. **Engenharia de Software com CMM.** Rio de Janeiro: Brasport, 1998.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. M. **Design Patterns: Elements of Reusable Object-Oriented Software**. Boston, MA: Addison Wesley, 1995. ISBN 0201633612

LEE, G. K.; COLE, R. E. **From a Firm-Based to a Community-Based Model of Knowledge Creation: The Case of The Linux Kernel Development**. Haas School of Business University of California, Berkeley , 2003. Disponível em: <http://www.stillhq.com/pdfdb/000501/data.pdf >. Acesso em: 2 mai. 2009.

MOCKUS, A.; FIELDING, R. T.; HERBSLEB, J. D. **Two Case Studies of Open Source Software Development: Apache and Mozilla.** ACM Transactions on Software Engineering and Methodology, Vol. 11, No. 3, 2002.

RAYMOND, E. S. **The Cathedral and The Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary**. Sebastopol, CA: O'Reilly & Associates, 1999. ISBN 1565927249.

REIS, C. **Caracterização de um Modelo de Processo para Projetos de Software Livre** . São Carlos, SP, 2001. Disponível em: <http://www.async.com.br/~kiko/quali/>. Acesso em: 11 mai. 2009.

TORVALDS, L.; DIAMOND, D. **Just for Fun: The Story of an Accidental Revolutionary.** New York, NY: HarperCollins Publishers, 2001.

TUOMI, I. **Internet, Innovation, and Open Source: Actors in the Network** .The Finnish National Fund for Research and Development, Berkeley, 2000. Disponível em: <http://flosspapers.org/12/1/Ilkka%2520Tuomi%2520-%2520Actors%2520in%2520the%2520Network.pdf>. Acesso em: 8 mai. 2009.

VIA DIGITAL. **Gestão de Projetos de Software Livre: Uma Abordagem de Práticas**. Disponível em: <http://www.viadigital.org.br/docs/Praticas.pdf> Acesso em: 06 mai. 2009.

WILLIAMS, S. **Free as in Freedom: Richard Stallman's Crusade for Free Software**. Sebastopol, CA: O'Reilly & Associates, 2002. ISBN 0596002874.

YOUNKER, J. **Foundations of Agile Python Development**. Berkeley, CA: Apress, 2008.

[New New Product Development Game by Hirotaka Takeuchi, Ikujiro Nonaka 10 pages.

Publication date: Jan 01, 1986. Prod. #: 86116-PDF-ENG]
O Oitavo Hábito, Stephen Covey
Mike Cohn, User Stories Applied