Faculdade de Tecnologia Senac RS

Curso de Tecnologia em Análise e Desenvolvimento de Sistemas

ROTEIRO DO PLANO DE TRABALHO

IDENTIFICAÇÃO	
Aluno: Camila San Martin Ayres	
Endereço Residencial: Rua Riachuelo 925/403	
Bairro: Centro	**CEP:** 90010-270
Cidade: Porto Alegre	**UF:** RS
Telefone(s): 51 91434791 / 3226 6987	
E-mail(s): smayres@gmail.com	

• Título do Projeto

Refatoração e Testes: garantindo uma transição segura.

• Orientador

Rafael Jeffman

• Histórico de Versões:

Versão	Autor	Descrição	Data
1	Camila Ayres	Apenas Introdução/Definição do Problema	17/03/11
2	Camila Ayres	Descrição da Solução/Cronograma	23/03/11
3	Camila Ayres	Revisão da versão final	30/03/11

• Apresentação Geral do Projeto

Software Libre, Software Livre e *Open Source* (FLOSS – *Free Libre Open Source Software*) descrevem o conceito de software com licença *copyleft*. Comparado aos *Copyright*, *copyleft* dá a quem recebe o software direitos adicionais[1]. Este trabalho fará uso do termo "Software Livre" para abranger todos os aspectos de FLOSS que enfatizam a filosofia e os aspectos do desenvolvimento colaborativo, bem como o aspecto legal e prático[2].

A licença *copyleft* bem como a GNU *Public License* garantem principalmente que: “A liberdade de redistribuir cópias deve incluir formas binárias ou executáveis do programa, bem como código-fonte, para ambas as versões modificadas.“ (GNU, 2010) O código-fonte, uma vez publicado sob a licença GNU *Public License*, estará sempre disponível sob essa licença, incluindo qualquer alteração realizada nesse. Embora as razões por trás da criação de licenças como *copyleft* e a GPL têm sido principalmente ideológica (STALLMAN, 2010), o termo mais comercial[3] *Open Source* tem impulsionado a GPL e licenças similares a tornarem-se dominante no mundo da TI (Tecnologia da Informação).

1 http://www.gnu.org/philosophy/free-sw.html
2 http://www.gnu.org/philosophy/open-source-misses-the-point.html
3 http://www.gnu.org/philosophy/open-source-misses-the-point.html e http://www.opensource.org/osd.html

A filosofia do Software Livre, tem tido um impacto muito além do mundo da TI em si. Software Livre pode ser visto como um exemplo de *Crowd Sourcing*[4], mas é muito mais antigo que esse conceito. Sites como Wikipedia, mídias sociais e colaborativas como *Facebook*, *Youtube* e *Flickr* têm forte ligação com a cultura e o modo de pensar do movimento do Software Livre.

O desenvolvimento de Software Livre foi explicado por Linus Torvalds em uma entrevista para a revista *Forbes* em Março de 2006: "Se você é um cientista louco, você pode usar software sob licença GPLv2 com os seus planos malignos de dominar o mundo ("Tubarões com lasers na cabeça !!"), e a GPLv2 apenas diz que você tem que devolver o código fonte. E por mim está tudo bem. Eu gosto de tubarões com lasers. Eu só quero que os cientistas loucos do mundo me paguem de volta do mesmo jeito que eu colaborei. Fiz o código fonte disponível para eles, então eles têm que disponibilizar as alterações para mim. Depois disso, podem fritar-me com os lasers de tubarão como todos eles querem." (LYONS, 2005). Linus Torvald desenvolveu o kernel do Linux seguindo essa lógica: pode-se usar e alterar livremente o código com a responsabilidade de disponibilizar essas contribuições a quem tiver interesse. Baseadas nessa filosofia, surgiram as comunidades de software livre.

O KDE[5] é uma comunidade de Software Livre que tem desenvolvido e entregue Software Livre e *Open Source* há mais de 15 anos. O KDE é, de acordo com a maioria das métricas, a segunda maior comunidade no mundo, apenas atrás da comunidade do Kernel do Linux. Entre os seus produtos estão uma variedade de aplicações, para a computação móvel, *desktops* e uma plataforma de desenvolvimento onde os aplicativos são projetados e desenvolvidos. KDE desenvolve software principalmente para Linux, mas seus produtos também estão disponíveis para Windows, Mac OS X e um número crescente de dispositivos móveis. A comunidade constituída por cerca de 800 a 1000 desenvolvedores ativos[6] e um número semelhante de contribuidores para arte gráfica, tradução, desenvolvimento de sites e outras tarefas dentro do projeto.

Os aplicativos desenvolvidos pelo KDE são escritos em C++, linguagem de programação projetada e implementada por Bjarne Stroustrup[7], utilizam Qt[8], um framework multiplatforma sob a licença LGPL (*Lesser General Public License*); e KDE-Libs, bibliotecas de software em C++/Qt que compõem o *KDE Developer Platform* ou KDE SDK, exigida por todos os aplicativos do KDE SC (KDE *Software Compilation).*

O Umbrello é a ferramenta de modelagem UML (*Unified Modeling Language*) distribuída com as ferramentas de desenvolvimento do KDE. O Umbrello foi escrito inicialmente em C++ para sistemas Unix pelo estudante universitário Paul Hensgen[9]. Em 2002, Jonathan Riddell[10] assumiu o projeto refatorando o código e incluindo novas funcionalidades. No mesmo ano, o Umbrello foi integrado as aplicações do KDE utilizando o framework Qt. Em 2008, Gopala Krishna[11], na época um estudante universitário, começou o port do limitado Framework QCanvas do Qt 3 para o QGraphicsView do Qt 4. Hoje o Umbrello tem uma versão para o KDE 4.6, mas a versão com o port iniciado pelo estudante anteriormente citado, está instável, como consequência, não foi integrada a versão atual do Umbrello. O Qt 4 alcançou um nível maior

4 http://crowdsourcing.typepad.com
5 http://kde.org
6 http://www.kde.org/announcements/4.3
7 http://www2.research.att.com/~bs
8 http://qt.nokia.com
9 http://phensgen.users.sourceforge.net/
10 http://jriddell.org
11 http://krishnaggk.blogspot.com

de estabilidade com melhorias como suporte a arquitetura *ModelView* e um novo *build system* modular. O Framework Graphics View surgiu na versão 4.2 e fornece recursos para manusear e interagir com elementos gráficos 2D, e um widget para a visualização dos itens[12].

• Definição do Problema

O Umbrello é a única ferramenta de modelagem UML presente no KDE SDK e amplamente utilizado por alguns projetos[13] do próprio KDE.

Considerando o significado da UML para a Engenharia de Software[14] e o fato que a importância do Software Livre tem aumentado na educação[15] e no governo (GROOT, KÜGLER, ADAMS e GOUSIOS 2006), pode-se afirmar um projeto como o Umbrello tem grande relevância. Em "*The State of Open Source*" da empresa de pesquisas Garner predisse: "Em 2012, mais de 90 por cento das empresas irá usar código aberto de forma direta ou incorporado," (JUDGE, 2008). A empresa de pesquisa IDC, escreveu em "*Linux in the New Economy*" (2009): "IDC projeta que o crescimento do software relacionados ao Linux vai liderar o setor durante o período 2008-2013 de recuperação pós-recessão, com uma taxa de crescimento anual composta (CAGR) de 23,6%. (...) Em comparação, o mercado global está projetado para aumentar em 5% 2008-2013 CAGR ". (GILLEN, 2009)

Hoje o Umbrello utilliza a classe Q3Canvas – a mesma classe QCanvas, renomeada apenas durante o processo do port - que faz parte do Qt3Support, biblioteca que foi desenvolvida para facilitar o port do Qt 3 para Qt 4, e que não devem estar no código em produção, mas apenas no processo intermediário, devendo ser imediatamente substituítas por uma classe equivalente do Qt 4. Nesse caso, o QCanvas deve ser portado para a classe QGraphicsView[16].

• Objetivos

Objetivo geral:

Conforme a definição do problema, o objetivo do projeto é realizar um port confiável do software utilizando ferramentas de análise e testes automatizados que garantem a aplicação correta dos conceitos de refatoração.

Objetivos específicos:

. Eliminar o uso do QCanvas no Umbrello.

. Criar uma cobertura de testes para as classes refatoradas especificadas no cronograma.

. Apresentação dos resultados da análise de código e testes após refatoração.

12 http://doc.qt.nokia.com
13 http://uml.sourceforge.net/users.php
14 http://www.uml.org/uml_success_stories/index.htm
15 http://one.laptop.org/about/mission e http://ndos.codeplex.com
16 http://doc.qt.nokia.com/4.7/q3canvas.html

• Análise de Tecnologias/Ferramentas

As ferramentas foram selecionadas baseadas nas especificações do projeto já existente. Estas ferramentas são amplamente usadas e testadas, o que oferece mais estabilidade.

. **Linguagens de programação:**

. C++: linguagem de programação em que o Umbrello foi desenvolvido.

. **Ferramentas para o desenvolvimento:**

. KDevelop 4: IDE para C/C++ e outras linguagens de programação. É baseado no KDE *Developer Platform*, no KDE e nas bibliotecas do Qt.

. Ferramenta de versionamento:

. SVN (*Subversion*): sistema de controle de versão de software utilizado pelos projetos do KDE.

. Frameworks:

. Qt: framework multiplataforma para C++, base do KDE e também utilizado pelo Umbrello.

. Documentação:

. Doxygate: é uma interface para a ferramenta Doxygen, utilizada para gerar documentação, escrita em Qt e utilizada em projetos do KDE.

. Análise e Testes:

. Krazy[17]: ferramenta para análise estática do código.

. Valgrind[18]: ferramenta de análise dinâmica.

. QtTestLibs[19]: framework para desenvolvimento de testes unitários em Qt.

. cppunit[20]: framework para desenvolvimento de testes unitários para C++.

. T-Plan Robot[21]: ferramenta para testes automatizados de interface.

• Descrição da Solução

O Umbrello utiliza recursos gráficos que permitem ao usuário desenhar diagramas 2D, essa é a sua principal função quanto software para diagramação UML. Logo, o projeto sendo explicado ao longo desse documento propõe o port dos recursos do QCanvas - cerca de 30 classes são dependentes desses recursos - para o QGraphicsView utilizando técnicas de refatoração de código juntamente com ferramentas de análise estática e dinâmica, concluindo o projeto com ferramentas de testes para a garantir a continuidade das funcionalidade originais do software.

O Framework Graphics View tem uma abordagem baseada na programação *model-view*. Varias visualizações (QGraphicsView*)* podem conter única cena (QGraphicsScene*)*, e a cena pode conter itens (QGraphicsItem) de diferentes formas geométricas.

17 http://techbase.kde.org/Development/Tutorials/Code_Checking
18 http://valgrind.org/
19 http://doc.qt.nokia.com/latest/qtestlib-manual.html
20 http://sourceforge.net/projects/cppunit/
21 http://www.t-plan.com/robot/ e http://sourceforge.net/projects/tplanrobot/

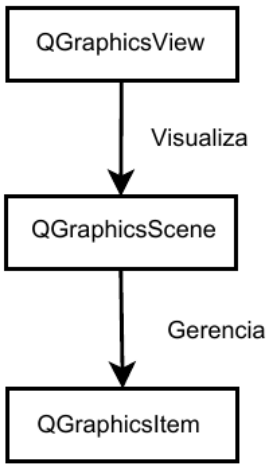

Figura 1. Framework Graphics View

Essa arquitetura é bastante similar a do Qcanvas, mas o QGraphicsItem, equivalente ao Q3CanvasItem é mais poderoso e mais fácil de usar. Por exemplo, o QGraphicsView implementa o arrastar e soltar (*drag* e *drop*) de um item em uma cena, funcionalidade inexistente no Qcanvas: “Uma das mais notáveis melhorias na nova API (QGraphicsView) é o sistema de coordenadas. Agora podemos usar coordenadas em números reais para adicionais um nível elevado de precisão a capacidade de visualização do QGraphicsView, especialmente quando utilizamos os zoom em uma cena. Isso simplifica ao ter que lidar com dads de fontes externas, como topologia, que geralmente usam coordenadas reais.” (HANSSEN, 2006)

No QCanvas, a geometria dos items eram relativa ao canvas, a cena onde os items são colocados, no QGraphicsView é possível criar items compostos, items dentro de items e redimensioná-los de forma independente ou de acordo com o seu item pai.

As melhorias propostas com o port para o Umbrello são na sua maioria relacionadas a perfomance e otimização de código (HANSSEN, 2006), pois o Framework Graphics View utiliza os melhores recursos do Qt4 que a partir da versão 4.2 utiliza BSP *tree*[22] (*Binary Space Partitioning Tree*) com um método de indexação que acelera a busca dos itens na cena. (SLETTA, 2010)

Processo de Refatoração e Testes proposto como solução para o Projeto:

. Análise:

Processo de análise do problema e desenvolvimento de diagramas UML conforme a necessidade para a ajudar na resolução. É nesse momento que decide-se pela melhor técnica de refatoração para determinada classe e quais os testes que devem ser aplicados.

. Desenvolvimento (Refatoração):

Refatorar é uma técnica amplamente utilizada na engenharia de software, que consiste em alterar um código existente com o objetivo de obter um código legível, de fácil manutenção. Para garantir que o software mantenha o seu comportamento após as devidas melhorias na sua estrutura, aplica-se então testes automatizados. (FOWLER, 2002)

22 http://web.cs.wpi.edu/~matt/courses/cs563/talks/bsp/bsp.html

. Testes:

. Análise do Código:

.**Estática**: análise das instruções e declarações para destacar possíveis erros de codificação, por exemplo, não deve-se usar QString::null ou Qstring(), ao invés disso usar o método clear().

.**Dinâmica**: detecção de erros em tempo de execução,

.Testes de Funcionalidade: são os testes unitários – testes específicos para cada cada funcionalidade do software e classe do código - e testes automatizados – testes executados por ferramentas desenvolvidas para tal objetivo, que garantem o correto funcionamento do software após a refatoração.

. Bug Fixing:

Correção dos erros encontrados durante a análise estática e dinâmica.

• Abordagem de Desenvolvimento

A metodologia adotada será uma variação do *Scrum*[23], que contará com a organização de *sprints* e pequenas entregas. De acordo com a solução proposta, o ciclo do *Scrum* será da seguinte maneira:

Ciclo														
Dias	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Tarefas	Análise				Desenvolvimento						Testes			*Bug Fixing*	

As alterações feitas no projeto estarão sendo versionadas com o controlador de versões SVN, logo, todo o histórico das alterações estará seguro e atualizado em svn+ssh://ayres@svn.kde.org/home/kde/trunk/KDE/kdesdk/umbrello/.

O KDE funciona, como muitos projetos de Software Livre, de forma descentralizada, que é freqüentemente chamada de *Bazaar* (RAYMOND, 2001), modelo de desenvolvimento ao contrário do mais tradicional *Cathedral*. Em suma, no modelo *Cathedral*, uma ou mais arquitetos projetam o produto, enquanto os desenvolvedores, sem muita autonomia, executam seus planos. Em contraste, no estilo Bazaar, pequenas equipes de trabalho semi-independentes dos componentes de todo o projeto e é bastante frequente o desenvolvimento de soluções concorrentes. Qualidade e capacidade de resposta das equipes de desenvolvedores determina a composição final do produto. A tomada de decisão é feito em público através de uma combinação de discussão meritocrática e do desenvolvimento de soluções concorrentes. Os desenvolvedores com um longo histórico na comunidade, muitas vezes exercem uma influência considerável e a maioria das aplicações tem um mantenedor que toma decisões. No entanto, devido à natureza voluntária, na maior parte do trabalho (mesmo os pagos por empresas costumam agir muito independente), não é possível dizer às pessoas o que fazer e a função da maioria dos tomadores de decisão se limita a manter portão. Logo, o Umbrello tem uma equipe responsável, atualmente pequena em comparação com outros projetos do KDE, com um mantenedor, Jonathan Riddell.

23 http://www.scrumalliance.org/

- **Arquitetura do Sistema**
  - **Modelagem Funcional**

    Diagramas de Classe e Diagramas de Sequência.

- **Validação**
  - **Estratégia**

Os testes e validação fazem parte do processo de port do software, conforme explicado no item Descrição da Solução e Abordagem de Desenvolvimento do projeto. Serão realizados a cada 15 dias após a refatoração de cada classe.

- **Cronograma**

O cronograma deste TCC foi todo baseado em *Sprints* de 15 dias, seguindo o ciclo explicado na abordagem de desenvolvimento do projeto, com datas programadas e concluídas. Pode-se também visualizar neste cronograma, separada por *sprints*, o produto de cada tarefa. As datas de entrega de relatórios estão destacadas separadamente dos *Sprints*.

TCC I				
Descrição da Atividade	**Produto**	**Início**	**Entrega**	**Detalhamento descritivo**
Plano de Trabalho	**Plano de Trabalho**	**18/03**	**01/04**	**Análise do Problema, definição da solução e criação do cronograma.**
Sprint 1: Classe UMLViewCanvas	Código	02/04	17/04	Análise, Desenvolvimento, Testes e Bug Fix.
Sprint 2: Classe LinePath::Circle e LinePath::SubsetSymbol	Código	18/04	02/05	Análise, Desenvolvimento, Testes e Bug Fix.
Sprint 3: Classe SeqLineWidget	Código	03/05	17/05	Análise, Desenvolvimento, Testes e Bug Fix.
Sprint 4: Classe UMLView	Código	18/05	01/06	Análise, Desenvolvimento, Testes e Bug Fix.
Relatório de Projeto	**Relatório**		**02/06**	**Entrega da versão Parcial do Relatório de Projeto Parcial para o Orientador**
Sprint 5: Classe UMLWidget	Código	02/06	16/06	Análise, Desenvolvimento, Testes e Bug Fix.
Relatório de Projeto	**Relatório**		**10/06**	**Entrega do Relatório de Projeto Parcial (versão final)**
Apresentação de Status do Projeto	**Apresentação**	**13/06**	**17/06**	**Apresentação de Status do Projeto.**

TCC II				
Descrição da Atividade	**Produto**	**Início**	**Entrega**	**Detalhamento descritivo**
Sprint 6: Merge das alterações realizadas no trunk com a brunch atual.	Código	17/06	01/07	Merge, Testes e Bug Fix.
Sprint 7: Classe UMLWidget, WidgetBase e AssociationWidget	Código	02/07	17/07	Análise, Desenvolvimento, Testes e Bug Fix.
Sprint 8: Classe ActivityWidgetActorWidget, ArtifactWidget, BoxWidget e ForkJoinWidget	Código	18/07	01/08	Análise, Desenvolvimento, Testes e Bug Fix.
Sprint 9: Classe CategoryWidget, ClassifierWidget e CombinedFragmentWidget	Código	02/08	17/08	Análise, Desenvolvimento, Testes e Bug Fix.
Sprint 10: Classe ComponentWidget, DatatypeWidget e EntityWidget	Código	18/08	01/09	Análise, Desenvolvimento, Testes e Bug Fix.
Sprint 11: Classe EnumWidget, FloatingDashLineWidget e FloatingTextWidget	Código	02/09	16/09	Análise, Desenvolvimento, Testes e Bug Fix.
Relatório de Projeto Atualizado	**Relatório**		**12/09**	**Entrega do Relatório de Projeto Atualizado.**
Sprint 12: Classe NodeWidget, NoteWidget e ObjectWidget	Código	17/09	01/10	Análise, Desenvolvimento, Testes e Bug Fix.
Seminário de Andamento	**Apresentação**	**19/09**	**23/09**	**Apresentação do Andamento do Projeto.**
Sprint 13: Classe PackageWidget, PinWidget e PreconditionWidget	Código	02/10	16/10	Análise, Desenvolvimento, Testes e Bug Fix.
Sprint 14: Classe RegionWidget, SignalWidget e StateWidget	Código	17/10	31/10	Análise, Desenvolvimento, Testes e Bug Fix.
Sprint 15: Classe UseCaseWidget e MessageWidget	Código	01/11	15/11	Análise, Desenvolvimento, Testes e Bug Fix.

Sprint 16: Merge das alterações realizadas no trunk com a brunch atual.	Código	16/11	30/11	Merge, Testes e Bug Fix.
Relatório de Projeto	**Relatório**		**21/11**	**Entrega do Relatório Final de Projeto.**
Apresentação Final	**Apresentação**	**28/11**	**02/12**	**Apresentação da Banca Final.**
Relatório Final de Projeto Revisado	**Relatório**		**12/12**	**Entrega do Relatório Final de Projeto Revisado.**

- ## Referências Bibliográficas

RAYMOND, Eric S. **The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary**. O'Reilly & Associates, Inc. Sebastopol, CA, USA. 2001. Disponível em: <http://catb.org/~esr/writings/homesteading/cathedral-bazaar/> Acesso em: 19 mar. 2011.

GROOT, Adriaan de; KÜGLER Sebastian; ADAMS, Paul J.; GOUSIOS, Giorgo. **Call for Quality: Open Source Software Quality Observation**. European research projects, COSPA e CALIBRE. 2006. Disponível em: <http://istlab.dmst.aueb.gr/~george/pubs/2006-OSS-GKAG/paper.pdf> Acesso em: 19 mar. 2011.

JUDGE, Peter. **Gartner: Open source will quietly take over**. Abr. 2008. Disponível em: <http://www.zdnet.co.uk/news/it-strategy/2008/04/04/gartner-open-source-will-quietly-take-over-39379900/> Acesso em: 19 mar. 2011.

JUDGE, Peter. **Gartner: Open source will quietly take over**. Abr. 2008. Disponível em: <http://www.zdnet.co.uk/news/it-strategy/2008/04/04/gartner-open-source-will-quietly-take-over-39379900/> Acesso em: 19 mar. 2011.

GILLEN, Al. **The Oportunity for Linux in the New Economy.** 2009. Disponível em: <http://www.redhat.com/f/pdf/idc-whitepaper-linux-in-new-economy.pdf> Acesso em: 19 mar. 2011.

STALLMAN, Richard. **Why Software Should Not Have Owners.** 2010. Disponível em: <http://www.gnu.org/philosophy/why-free.html> Acesso em: 19 mar. 2011.

GILLEN, Al. **The Oportunity for Linux in the New Economy.** 2009. Disponível em: <http://www.redhat.com/f/pdf/idc-whitepaper-linux-in-new-economy.pdf> Acesso em: 19 mar. 2011.

FOWLER, Martin. **Refactoring: Improving the Design of Existing Code**. 2ª ed. New York: Addison-Wesley, Fev. 2001.

BLANCHETTE, Jasmin; SUMMERFIELD, Mark. **C++ GUI Programming with Qt4**. 2ª ed. Boston: Prentice-Hall Open Source Software Development Series, Fev. 2008.

SUMMERFIELD, Mark. **Advanced Qt Programming: Creating Great Software with C++ and Qt 4**. 1ª ed. Boston: Prentice-Hall Open Source Software Development Series, Jul. 2010.

ANSSEN, Andreas Aardal. **A Better Canvas.** Qt Quarterly, 2006. Disponível em: <http://doc.trolltech.com/qq/qq19-graphicsview.html> Acesso em: 25 mar. 2011.

HANSSEN, Andreas Aardal. **A GraphicsView sneak-peek.** Maio 2006. Disponível em: <http://labs.qt.nokia.com/2006/05/01/a-graphicsview-sneak-peek/> Acesso em: 25 mar. 2011.

SLETTA, Gunna. **Qt Graphics and Performance – The Cost of Convenience.** Jan. 2010. Disponível em: <http://labs.qt.nokia.com/2010/01/11/qt-graphics-and-performance-the-cost-of-convenience/> Acesso em: 26 mar. 2011.

GNU Operating System; Free Software Fundation. **The Free Software Definition.** Nov. 2010. Disponível em: <http://www.gnu.org/philosophy/free-ww.html> Acesso em: 20 mar. 2011.

LYONS, Daniel. **Linux Rules Supercomputers**. Forbes. Mar. 2005. Disponível em: <http://www.forbes.com/2005/03/15/cz_dl_0315linux.html> Acesso em: 22 mar. 2011.

- **Componentes Re-utilizados**
  - Classes do Umbrello.
  - KDE-Libs.

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

PLANO DE TRABALHO

CAMILA SAN MARTIN AYRES

PORTO ALEGRE

2011

_____	_____
Camila Ayres	Rafael Jeffman
Orientando	Orientador