
CoMDD: uma abordagem colaborativa para
auxiliar o desenvolvimento orientado a
modelos

David Fernandes Neto

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito: 11 de fevereiro de 2012

Assinatura: _____

CoMDD: uma abordagem colaborativa para auxiliar o desenvolvimento orientado a modelos

David Fernandes Neto

Orientadora: *Profa. Dra. Renata Pontin de Mattos Fortes*

Monografia apresentada ao Instituto de Ciências Matemáticas e de Computação — ICMC/USP, para o exame de Qualificação, como parte dos requisitos para obtenção do título de Mestre em Ciências de Computação e Matemática Computacional.

USP - São Carlos
Fevereiro/2012

Resumo

O

Sumário

Resumo	i
Lista de Abreviaturas e Siglas	ix
1 Introdução	1
1.1 Descrição geral da área, o Problema e a Justificativa	1
1.2 Hipótese e objetivo	4
1.3 Escopo	4
1.4 Resultados Esperados	4
1.5 Estrutura deste trabalho	5
2 Revisão Bibliográfica	7
2.1 Model Driven Development	7
2.2 Domain Specific Languages	9
2.3 Colaboração	10
2.4 Wiki	10
3 Desenvolvimento	13
3.1 O CoMDD - <i>Collaborative Model Driven Development</i>	13
3.2 Arquitetura do CoMDD	16
3.3 Método	17
3.3.1 Estudo de Caso I	17
3.3.2 Estudo de Caso II	22
3.3.3 Estudo de caso 3: análise de vídeos e planilha de ticar	31

3.4	Trabalhos Relacionados	31
3.5	Cloud 9 IDE	32
3.6	Google Docs e ferramentas de edição concorrente em tempo real	33
3.7	Artigos	33
4	Conclusão	35
4.1	Conclusões - verificar com carinho	35
4.2	Contribuições	36
4.3	Trabalhos Futuros	37
4.4	Limitações do Trabalho	37
4.5	Lições aprendidas	38
	Referências	41
A	Implementação do CoMDD	45
B	Apêndice C	47
A	Textos apresentados no experimento	49
A.1	Texto I: O que é o CoMDD?	49
A.2	Texto II: Como programar para o CoMDD	50
A.3	Instalação do Eclipse+SVN e importação da DSL	51
A	Textos apresentados no experimento	53
A.1	Conversa da Equipe B do Estudo de Caso 2 realizada no final do experimento . . .	53
A	Perguntas Respondidas pelas Equipes A e B do Estudo de Caso 2	55
A.1	Perguntas e respostas da Equipe A	55
A.2	Perguntas e respostas da Equipe B	56
A	Detalhes de implementação	59
A	Resultados do Aluno	61

Lista de Figuras

2.1	Processo convencional de desenvolvimento de software (Lucrédio, 2009)	8
2.2	Processo de desenvolvimento de software orientado a modelos (Lucrédio, 2009) . .	8
3.1	Dua pessoas editando o mesmo modelo, simultaneamente ou não. Na atual imple- mentação o CoMDD não aceita edição simultânea.	14
3.2	Três pessoas editando diferentes modelos que dependem entre si, simultaneamente ou não. Na atual implementação o CoMDD não aceita edição simultânea.	14
3.3	Comentários feitos na página de edição	15
3.4	Comentários feitos para cada versão da página	15
3.5	Processo de edição de modelos e geração de código-fonte realizado na wiki.	16
3.6	Arquitetura do CoMDD.	17
3.7	Log da conversa entre o desenvolvedor 1 e o desenvolvedor 2	27
3.8	Log da conversa entre o desenvolvedor 1 e o desenvolvedor 2	29

Lista de Tabelas

--

Lista de Abreviaturas e Siglas

API *Application Programming Interface*
CDE *Collaborative Development Environment*

Introdução

Este capítulo descreve o desenvolvimento colaborativo orientado a modelos, na qual o trabalho está inserido, apresenta o problema de pesquisa e sua justificativa; define a hipótese e o objetivo do trabalho. Os resultados esperados e o escopo do trabalho também são apresentadas aqui.

1.1 Descrição geral da área, o Problema e a Justificativa

O desenvolvimento orientado a modelos (*Model Driven Development* - MDD) é um método de desenvolvimento de software que foca nos modelos, como os principais artefatos de desenvolvimento, e nas transformações desses modelos para gerar código-fonte. O objetivo do MDD é reduzir a distância semântica existente entre o domínio do problema e o domínio da implementação/solução, utilizando modelos mais abstratos que protegem os desenvolvedores de software das complexidades inerentes às plataforma de implementação (Teppola et al., 2009; France e Rumpe, 2007).

A principal vantagem do MDD é poder expressar modelos usando conceitos menos vinculados à detalhes de implementação, além do fato de modelos serem mais próximos do domínio do problema. Isto torna os modelos mais fáceis de se especificar, entender e manter. E, em alguns casos, ainda é possível os especialistas do domínio produzirem os sistemas ao invés dos especialistas da tecnologia de implementação (Selic, 2003; Sriplakich et al., 2006)

E ainda o MDD possui traz benefícios como: produtividade, portabilidade, manutenção etc (Kleppe et al., 2003; Lucrédio, 2009; Bendix e Emanuelsson, 2009). Há diversos trabalhos (Aho et al., 2009; France e Rumpe, 2007; Kleppe et al., 2003; Lucrédio, 2009; Mellor et al., 2003;

Sánchez et al., 2009; Stahl et al., 2006; Liggesmeyer e Trapp, 2009) que propõem e que usam o MDD como abordagem para desenvolvimento de software. Ainda, entre as áreas da indústria que usam MDD, a área de sistemas embarcados já se beneficia desta abordagem antes mesmo do surgimento da UML ou da *Model Driven Architecture* - MDA¹ (Liggesmeyer e Trapp, 2009).

Entretanto, em relação à modelos textuais², como equipes de desenvolvimento de software trabalham?

Uma possibilidade para trabalhar com MDD em uma equipe de desenvolvimento é usar uma IDE, para o desenvolvimento de modelos e geração de código-fonte, e um sistema de versionamento, para compartilhar modelos entre os membros da equipe. Esta abordagem, a de uma IDE associada à um sistema de versionamento^{3,4} é comum na indústria.

Entretanto, os sistemas versionamento podem causar conflitos quando dois ou mais desenvolvedores modificam o mesmo documento e submetem ao servidor⁵.

Neste momento, desenvolvedores têm que dedicar certo tempo para resolver os conflitos, e às vezes esses conflitos só podem ser resolvidos manualmente. Além disso, para os desenvolvedores saberem como resolver o conflito corretamente eles precisam estar em contato entre si e para decidirem como realizar o *merge* precisam estar disponíveis no momento (ao Gustavo Prudêncio et al., 2012).

Ainda há um outro ponto ao se usar as IDEs e os sistemas de versionamento: a necessidade de instalação desses softwares, ou seja, para as equipes usarem uma IDE e um sistema de versionamento é preciso que cada desenvolvedor tenha instalado esses softwares em seus computadores. O que significa dizer que se um *stakeholder* estiver interessado em acompanhar o desenvolvimento de um software ele precisa instalar, em seu computador, essas ferramentas⁶, o que pode-se dizer ser uma tarefa difícil e trabalhosa para não desenvolvedores.

Além da instalação e configuração, o *stakeholder* deve saber como funciona essas ferramentas. Se ele quiser participar da edição dos modelos⁷ deverá ainda conhecer como funciona a geração de códigos da IDE, como fazer *check-in*⁸, *check-out*⁹, entre outros termos e suas variações de nome¹⁰. De forma que, embora para um usuário não desenvolvedor, trabalhar com modelos seja

¹A MDA (Model Driven Architecture) é um padrão da OMG como abordagem do MDA, com uso da UML para a modelagem

²Lembrando que modelos podem ser textuais ou gráficos.

³Este trabalho usará os termos sistema de controle de versão ou SCV ou sistema de versionamento como sinônimos.

⁴Este trabalho está focando somente no uso de versionadores, pois o estudo de sistemas de gerenciamento de configuração tornaria a pesquisa muito extensa.

⁵Isso no caso de uma política de versionamento *otimistic*, a qual permite o desenvolvimento concorrente (Sarma et al., 2003).

⁶No experimento deste trabalho, descrito no Seção 3.3.2, dois desenvolvedores experientes levaram cerca de quarenta minutos para baixar, instalar e configurar o Eclipse e o plugin do SVN

⁷Este trabalho parte do uso de MDD como abordagem de desenvolvimento e por isso está-se referindo ao MDD.

⁸Também chamado de *commit*, segundo Sussman et al (Ben Collins-Sussman e Pilato, ????) é a operação de enviar mudanças da cópia local para a cópia no repositório.

⁹Segundo Sussman et al (Ben Collins-Sussman e Pilato, ????) é a operação de fazer uma cópia local da cópia do repositório.

¹⁰Por exemplo, no plugin do SVN para o Eclipse, o *check-out* é quando faz-se uma cópia completa de todos os arquivos do servidor e *update* quando apenas atualiza as modificações.

mais fácil que código-fonte (Selic, 2003), essa vantagem é minimizada quando há a necessidade do uso dessas ferramentas por parte deste usuário não desenvolvedor.

Além do que, todo esse processo de instalação e de *check-in/check-out* e etc, é relativamente demorado, de forma que é mais uma desvantagem somada a necessidade de conhecimento desses conceitos. Jones (Jones e Scaffidi, 2011) apoia esta ideia quando ele afirma que cientistas raramente usam controle de versão devido ao alto custo inicial para configuração desses sistemas.

Por fim, mesmo que o usuário ainda receba um treinamento, as abordagens são diferentes entre si. Por exemplo, a abordagem do SVN¹¹ é diferente da abordagem do Git¹², que é diferente da abordagem *Team Foundation Service* da Microsoft¹³, de modo que além da necessidade do conhecimento da instalação, configuração, conceitos e uso, é também necessário o conhecimento da lógica de funcionamento dessas abordagens e de suas ferramentas, mostrando o versionamento que é empregado desta forma um processo com relativa alta curva de aprendizado inicial.

Por outro lado, as wikis são ferramentas conhecidas por serem simples de usar e ágeis para colaborar (Abeti et al., 2009; Tetard et al., 2009). De forma que, dependendo do problema, o uso de uma wiki ao invés de uma IDE associada a um SVN pode contornar todos os problemas descritos até aqui, pois as wikis funcionam em um navegador, possuem um controle de versões com histórico simplificado, possibilitam inserção de comentários e, no caso da Xwiki¹⁴ não há problemas de conflito, pois ela permite travar a edição concorrente. Assim, falta apenas implementar na wiki um suporte a modelagem e geração de código-fonte, para que as wikis possam servir de plataforma de MDD colaborativo.

Ainda, analisando o histórico de aplicações como músicas, vídeos, jogos, o armazenamento de arquivos, entre outros, observa-se uma tendência das aplicações migrarem para Web, como exemplo disso tem-se o Grooveshark®¹⁵ (reprodutor de músicas on-line), o Youtube®¹⁶ (reprodutor de vídeos on-line), o Dropbox®¹⁷ (armazenador de arquivos on-line) e até mesmo IDEs estão convergindo para a Web, como é o caso do Cloud9¹⁸, por exemplo. De modo que, ter uma ferramenta possível de editar, compartilhar e versionar modelos, gerar código-fonte e ainda permitir a entrada de comentários pode beneficiar desenvolvedores e *stakeholders*.

Contudo, seria possível uma wiki ser usada como uma ferramenta de desenvolvimento e compartilhamento de modelos, de forma que possa atender às necessidades dos usuários das IDEs e dos sistemas de versionamento? A partir deste questionamento este trabalho definiu uma abordagem colaborativa orientada a modelos (*Collaborative Model Driven Development* - CoMDD), a qual usa uma wiki para apoiar o MDD em uma equipe de desenvolvimento nas tarefas de modelagem, geração de código, versionamento e comunicação.

¹¹<http://subversion.tigris.org/>

¹²<http://git-scm.com/>

¹³<http://tfspreview.com/>

¹⁴<http://www.xwiki.org>

¹⁵grooveshark.com

¹⁶youtube.com

¹⁷dropbox.com

¹⁸<http://c9.io/>

1.2 Hipótese e objetivo

A partir do cenário levantado na seção anterior, definiu-se a seguinte hipótese que norteia este trabalho:

É possível usar uma wiki, ao invés do processo tradicional de desenvolvimento¹⁹, para desenvolver um sistema colaborativamente usando MDD²⁰.

O objetivo deste trabalho é evidenciar que a hipótese de pesquisa é verdadeira para um caso específico (a ser tratado no estudo de caso) e com isso incentivar o uso do CoMDD por equipes de desenvolvimento.

1.3 Escopo

Não faz parte do escopo deste trabalho comparar o desenvolvimento tradicional (que usa linguagens de amplo propósito, como Java, por exemplo) com o MDD, uma vez que a literatura já discute sobre isso e sobre os benefícios em termos de produtividade que o MDD agrega para o desenvolvimento de software (Aho et al., 2009; France e Rumpe, 2007; Kleppe et al., 2003; Lucrédio, 2009; Mellor et al., 2003; Sánchez et al., 2009; Stahl et al., 2006; Liggesmeyer e Trapp, 2009).

Este trabalho cita sobre a comunicação entre os integrantes, desenvolvedores ou não, e entende que é importante promover e incentivar a comunicação; contudo, não é escopo deste trabalho desenvolver ferramentas ou um processo de comunicação entre os desenvolvedores e nem um estudo a fundo sobre, mas apenas incentivar a partir da forma que o CoMDD é concebido e incentivar o uso de mensageiros instantâneos ou ferramentas de vídeo-conferência para promover a comunicação.

O controle de versão é uma das áreas de gerenciamento de configuração e apenas esta área será discutida neste trabalho, do contrário, este trabalho se tornaria muito extenso e pouco profundo.

1.4 Resultados Esperados

Espera-se com este trabalho, que o uso do MDD seja cada vez mais empregado no desenvolvimento de sistemas e de que as wikis evoluam a ponto de terem mais recursos de uma IDE, até que um dia, a computação tenha o suporte em termos de ferramentas e processo para desenvolver softwares de grande porte e complexos usando wikis e MDD.

¹⁹Definimos processo tradicional de desenvolvimento como sendo o uso de uma IDE e de um sistema de controle de versão *otimistic*

²⁰No caso estamos nos referindo a dsls como abordagem de mdd. No capítulo de revisão bibliográfica isso ficará mais claro.

1.5 Estrutura deste trabalho

No Capítulo 2 faz-se uma revisão bibliográfica dos conceitos necessários para entender a abordagem proposta. No capítulo 3 tem-se a apresentação do CoMDD, de três estudos de caso realizados para tentar evidenciar a veracidade da hipótese, as conclusões e resultados dos estudos de caso e os trabalhos correlatos. Por fim, o Capítulo 4 apresenta as considerações finais.

Revisão Bibliográfica

2.1 Model Driven Development

O Desenvolvimento Orientado a Modelos (*Model-Driven Development* - MDD) é uma abordagem da Engenharia de Software que consiste na aplicação de modelos para elevar o nível de abstração, na qual os desenvolvedores criam e evoluem o software. Sua intenção é tanto simplificar (tornar mais fácil) quanto formalizar (padronizando, de forma que a automação seja possível) as várias atividades e tarefas que formam o ciclo de vida do software (Hailpern e Tarr, 2006); ou numa definição mais singela: o MDD é a simples noção de construir um modelo de um sistema e depois transformá-lo em algo real (Mellor et al., 2003).

Uma outra definição semelhante é a de que o MDD é uma abordagem de desenvolvimento, integração e interoperabilidade de sistemas de tecnologia da informação. Ela se refere ao uso sistemático de modelos e de transformações de modelos como artefatos primários, durante todo o ciclo de vida do software. O MDD eleva o nível de abstração em que os desenvolvedores produzem softwares, simplificando e formalizando as diversas atividades e tarefas que compõem o processo de desenvolvimento de software (Sánchez et al., 2009), expressando o que um computador deve fazer por meio de modelos, que escondem detalhes de implementação (Meijler, 2005).

O que caracteriza o MDD são os modelos como foco primário do desenvolvimento de software, ao invés das linguagens de programação. Os modelos são usados para descrever vários aspectos do software e para automatizar a geração de código. A principal vantagem disto é poder expressar modelos usando conceitos menos vinculados a detalhes de implementação, além do fato de modelos serem mais próximos do domínio do problema. Isto torna os modelos mais fáceis de se especificar, entender e manter, do que abordagens que não usam modelos. E em alguns casos,

ainda é possível os especialistas do domínio produzirem os sistemas ao invés dos especialistas da tecnologia de implementação (Selic, 2003; Sriplakich et al., 2006).

A Figura 2.1 ilustra um engenheiro de software elicitando os requisitos, modelando com base nesses requisitos e por fim implementando com base nos modelos. A Figura 2.2 ilustra o um engenheiro de software elicitando os requisitos, modelando com base nos requisitos e os modelos gerando o código.

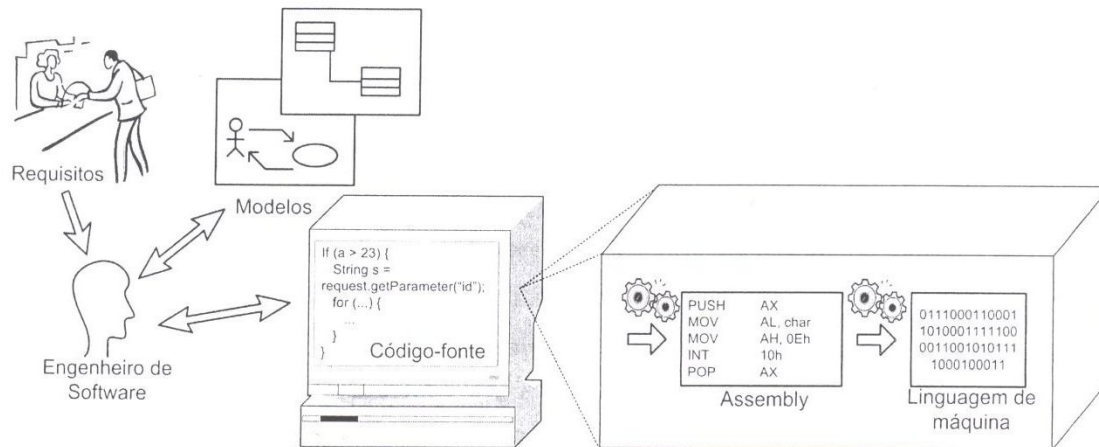


Figura 2.1: Processo convencional de desenvolvimento de software (Lucrédio, 2009)

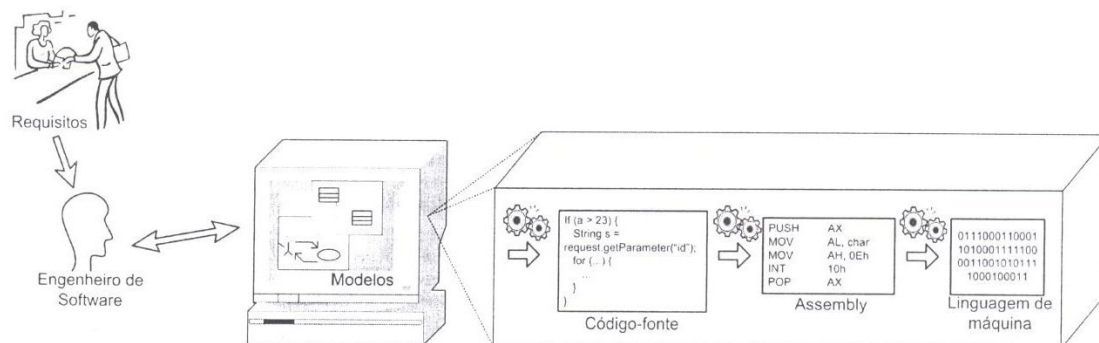


Figura 2.2: Processo de desenvolvimento de software orientado a modelos (Lucrédio, 2009)

As Vantagens do MDD

Segundo (Kleppe et al., 2003; Lucrédio, 2009) o MDD apresenta grandes benefícios como:

- **Produtividade:** fatores como redução de atividades repetitivas e manuais e o aumento da possibilidade de reuso, podem contribuir para o aumento da produtividade no processo de desenvolvimento;
- **Portabilidade e Interoperabilidade:** como o modelo é independente de plataforma, um mesmo modelo pode ser transformado em código para diferentes plataformas;

- Corretude: o MDD evita que os desenvolvedores exerçam atividades repetitivas e manuais para gerar código; dessa maneira, evita-se também alguns erros como: geradores de código não introduzem erros acidentais, como o de digitação, por exemplo;
- Manutenção: alterações no código relativas à manutenção podem requerer o mesmo esforço produzido durante o desenvolvimento;
- Documentação: modelos são o artefato principal do processo de desenvolvimento e por isso não se desatualizam, pois o código é gerado a partir deles e com isso a documentação se mantém também sempre atualizada;
- Comunicação: modelos são mais fáceis de entender do que código-fonte, assim isso facilita a comunicação entre os desenvolvedores, os *stakeholders* e demais envolvidos.

Analisando tais definições de MDD e suas vantagens apresentadas, pode-se concluir que MDD é um conceito bem definido, pois diferentes autores apresentam as mesmas idéias de que MDD é uma elevação no nível de abstração do desenvolvimento de software, no qual o modelo passa de um artefato auxiliar para uma peça fundamental no processo de desenvolvimento. Em virtude dessa elevação na abstração, muitos problemas relativos a portabilidade, interoperabilidade, corretude, manutenção e documentação, são reduzidos nesta abordagem. Contudo, é fato que mesmo o MDD com tantos benefícios, inclusive o aumento de produtividade, ainda está longe de ser a solução para todos os problemas relativos ao desenvolvimento de software.

Metamodelagem

A metamodelagem é um dos principais aspectos do MDD. É necessário criar um metamodelo do conhecimento para o MDD lidar com alguns desafios, como: a construir uma DSL; validar modelos; realizar transformações de modelo; gerar código e integrar ferramentas de modelagem a um domínio (Stahl et al., 2006).

Um metamodelo descreve a estrutura de um modelo. De maneira abstrata, o metamodelo define os construtores de uma linguagem de modelagem e seus relacionamentos, bem como as constantes e regras de modelagem; contudo ele não descreve a sintaxe concreta da linguagem. Metamodelos e modelos têm um relacionamento de classe e instância, ou seja, cada modelo é uma instância de um metamodelo (Stahl et al., 2006).

2.2 Domain Specific Languages

Uma DSL (*Domain-Specific Language*) é uma linguagem de programação ou uma linguagem de especificação executável, que oferece, através de notações e abstrações, poder expressivo focado em um problema de um domínio particular, e geralmente restrito a este domínio. Embora seja uma

linguagem específica e não forneça uma solução geral para muitas áreas, ela provê uma solução muito melhor para um domínio particular (van Deursen et al., 2000) quando comparada com uma linguagem de propósito geral, como a UML, por exemplo.

A DSL é uma das abordagens do MDD, havendo outras como a da OMG (*Object Management Group*), a MDA, por exemplo. O foco deste trabalho está no desenvolvimento de uma DSL. Ainda é importante ressaltar que as DSLs podem ser textuais ou gráficas, como por exemplo a SQL é uma DSL textual para banco de dados e o Labview é uma DSL gráfica, usada na engenharia elétrica, para medições e automações.

2.3 Colaboração

Colaboração significa duas ou mais pessoas trabalhando juntas para compartilhar e trocar dados, informações e conhecimento (McQuay, 2004). A colaboração é uma atividade que tem sido evidenciada cada vez mais com o aumento da complexidade do desenvolvimento de sistemas, pois para se desenvolver um sistema robusto e complexo uma só pessoa dificilmente é o suficiente. Assim, o desenvolvimento e a manutenção de softwares complexos requerem a colaboração de diversos desenvolvedores (Sriplakich et al., 2006).

2.4 Wiki

As wikis são consideradas sistemas de computador que tornam fácil a edição de páginas web por qualquer pessoa. Elas também são vistas como uma filosofia em relação à forma como os usuários devem editar páginas web (Louridas, 2006). Em sua essência, a wiki é uma página web, ou um conjunto de páginas web, editável. Ela permite que qualquer pessoa possa, facilmente, adicionar ou revisar o conteúdo por meio de, praticamente, qualquer navegador web (Jang e Green, 2006).

As wikis ganham muita aceitação pelos usuários (Tetard et al., 2009) e são utilizadas no meio acadêmico. As wikis se mostram uma opção no mundo das ferramentas colaborativas (Jang e Green, 2006) e ganham cada vez mais espaço no setor privado, se tornando uma tecnologia reveladora para apoiar a colaboração dentro e entre empresas (Majchrzak et al., 2006).

A colaboração que a Wiki oferece tem permitido muitos trabalhos científicos em um leque amplo de áreas, como pode-se notar em trabalhos que usam a wiki para gerenciar conhecimento (Jing e Fan, 2008) e requisitos (Abeti et al., 2009), auxiliar o ensino e aprendizado (Elrufaie e Turner, 2005; Tetard et al., 2009), compartilhar informações ¹, entre outros.

As Wikis são colaborativas por natureza e a qualidade do conteúdo melhora à medida que as pessoas contribuem (Mehta, 2009). Todos usuários habilitados podem editar uma Wiki e revisar

¹http://en.Wikipedia.org/Wiki/Main_Page

seu conteúdo (Abeti et al., 2009), de maneira irrestrita e democrática (Tetard et al., 2009). A Wiki enfatiza velocidade e flexibilidade ao invés de controle restrito (Abeti et al., 2009) e o fato da Wiki ser fácil de implementar e de entender é, provavelmente, a chave para sua popularidade (Tetard et al., 2009).

Além da colaboração, simplicidade, facilidade e do suporte à edição colaborativa do conteúdo como sendo uma das principais funcionalidades da Wiki (Tetard et al., 2009), as Wikis são as plataformas eleitas pela engenharia de requisitos, uma vez que permitem que diversos *stakeholders* distribuídos, compartilhem a definição do sistema; além do que as Wikis têm controle de versão de páginas, gerenciamento de alterações de conteúdo, comunicação entre usuários por meio de fóruns e *chats* (Abeti et al., 2009).

O que as torna especiais não é apenas a facilidade de contribuição, mas também seus recursos (Mehta, 2009), como:

- Simples sintaxe de marcação;
- Criação de links simples ou automáticos, mesmo quando a página de destino ainda é inexistente;
- Controle de acesso total de usuários e páginas;
- Histórico completo das revisões, com possibilidade de reverter a qualquer momento;
- Muitas Wikis não precisam de banco de dados, o que as deixa portáteis;
- Categorizar páginas para melhor organização;
- Customizáveis.

Atualmente, existem centenas de Wikis desenvolvidas, como: MediaWiki², DokuWiki³, Xwiki⁴, entre outras. A wiki usada neste trabalho é a Xwiki, por ser implementada em Java.

Porquê wikis e não CMSs

Atualmente as Wikis são os sistemas que melhor se encaixam na simplicidade e facilidade para o CoMDD, ou seja, são sistemas aceitos tanto pela academia quanto pela indústria no que diz respeito à troca de conhecimento de maneira simples e rápida. Este é o motivo do uso de uma wiki no CoMDD. Por exemplo, existem os *Content Management System* - CMS. Os CMSs são ferramentas para desenvolver sites e que possibilitam criar uma hierarquia de permissões, na qual se pode ter um administrador geral, editores de seções, editores de páginas e etc (Mehta, 2009).

²<http://www.mediawiki.org/wiki/MediaWiki>

³<http://www.dokuwiki.org/dokuwiki>

⁴<http://www.xwiki.org/xwiki/bin/view/Main/WebHome>

A princípio, os CMSs poderiam suportar uma DSL, entretanto pode-se dizer que criar e editar conteúdos em CMSs é, geralmente, mais restrito do que em Wikis. As Wikis são mais abertas e têm menos rigor no controle de permissão, diferentemente dos CMSs, em que cada usuário tem permissões específicas de acesso para determinadas páginas e permissões de ações. Enquanto o controle de permissão em CMSs é mais completo, as Wikis são projetadas para serem fáceis e rápidas de editar. As Wikis também são mais fáceis de *linkar* páginas por usuários finais, além de permitirem que usuários não técnicos gerenciem e acessem o conteúdo de maneira mais fácil do que nos CMSs (Zimmerly, 2009).

Abeti et al. (Abeti et al., 2009) também comparam as Wikis com os CMSs. Eles afirmam que as Wikis são mais fáceis de aprender pelos usuários, pois basta saber editar uma página; enquanto que nos CMSs deve-se conhecer a base do CMS. Nas Wikis, as páginas não têm estrutura e são o conceito central para gerenciamento de conteúdo; enquanto que nos CMSs as páginas são estruturadas e possuem uma hierarquia de páginas, seções, itens etc, além da estrutura do site de um CMS ser gerenciada por um administrador.

As Wikis focam na velocidade, flexibilidade, colaboração e compartilhamento de conteúdo; ao invés do controle restrito de acesso e edição (Abeti et al., 2009).

Assegurar a integridade e consistência de artefatos versionados em um ambiente que suporte acesso concorrente é um problema difícil (Zhang e Ray, 2007), portanto para evitar os conflitos de edição e merge adotou-se uma abordagem *pessimistic*, ou seja, uma política em que o artefato é bloqueado quando já está sendo editado por outra pessoa. Aplicando essa política, conflitos de edição são evitados e com isso desenvolvedores não gastarão tempo ou esforços resolvendo conflitos. Ainda, a política *pessimistic* pode ajudar na comunicação, pois os desenvolvedores saberiam quando outra pessoa estaria modificando o mesmo artefato, incentivando a comunicação entre os desenvolvedores com os mesmos interesses (ao Gustavo Prudêncio et al., 2012; Sarma et al., 2003). Uma outra possível solução seria a edição simultânea estilo google docs citeMeu trabalho!...

Desenvolvimento

Este capítulo apresenta a abordagem desenvolvida neste trabalho, cita suas funcionalidades e relata três estudos de caso desenvolvidos para evidenciar que a hipótese de pesquisa é verdadeira. Também discute os resultados e as lições aprendidas dos estudos de caso bem como as limitações da abordagem e da implementação.

3.1 O CoMDD - *Collaborative Model Driven Development*

O CoMDD é uma abordagem que consiste no desenvolvimento colaborativo orientado a modelos, onde a colaboração é referente a edição colaborativa de modelos. Seja mais de uma pessoa editando o mesmo modelo (concorrentemente ou não), seja várias pessoas editando vários modelos, de forma que cada modelo tenha apenas uma pessoa e que esses modelos sejam dependentes.

A figura 3.1 ilustra duas pessoas editando um mesmo modelo e a figura 3.2 ilustra três pessoas editando diferentes modelos, mas que são dependentes.

A implementação da abordagem CoMDD realizada neste trabalho, para evidenciar a hipótese, consiste no uso de uma DSL como abordagem de MDD e de uma Wiki para promover a colaboração e suporte da DSL. Mais especificamente, a DSL é para o domínio de robôs móveis autônomos.

A DSL permite a edição de modelos e geração de código-fonte, usando templates, em uma Wiki. Desta forma, a Wiki serve tanto como ferramenta de suporte da DSL quanto ferramenta colaborativa pois possibilita o trabalho em equipe e o versionamento de modelos.

A implementação do CoMDD permite:

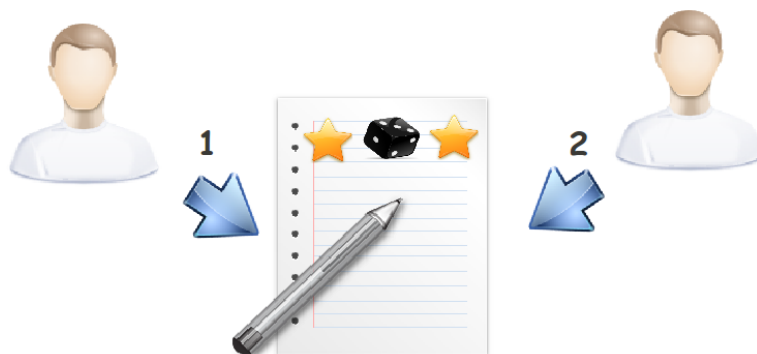


Figura 3.1: Duas pessoas editando o mesmo modelo, simultaneamente ou não. Na atual implementação o CoMDD não aceita edição simultânea.

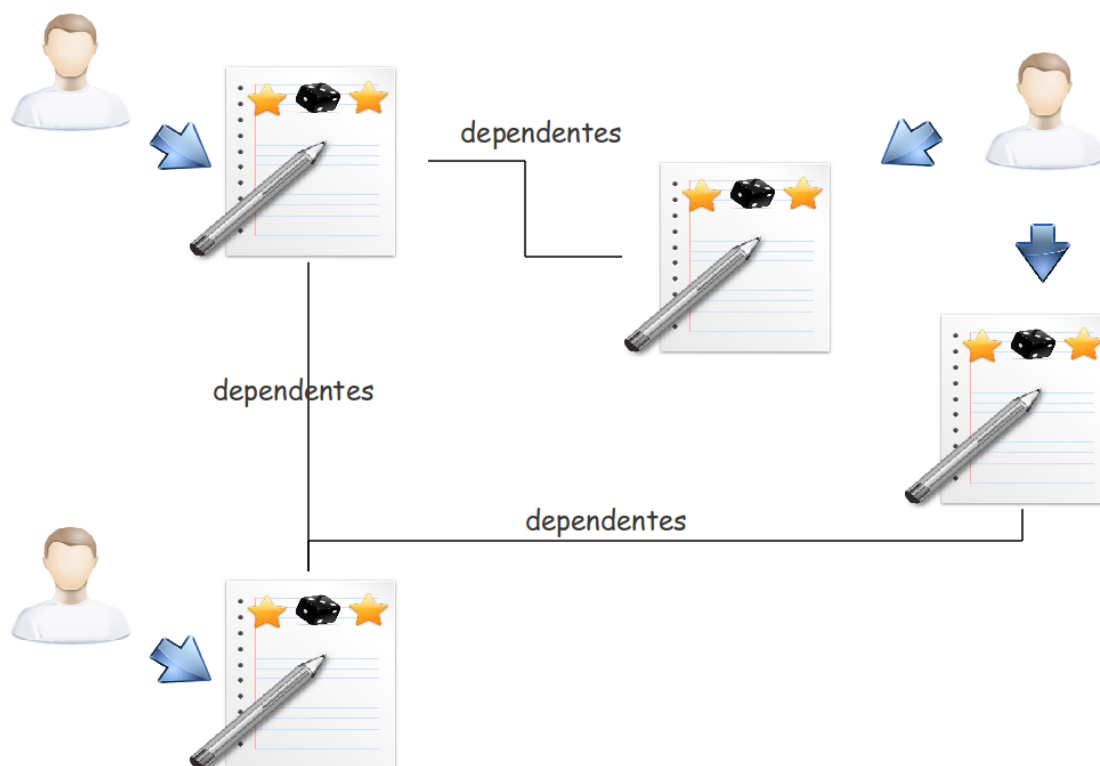


Figura 3.2: Três pessoas editando diferentes modelos que dependem entre si, simultaneamente ou não. Na atual implementação o CoMDD não aceita edição simultânea.

1. **Edição de modelos:** criar ou alterar modelos de acordo com o metamodelo definido¹. Os modelos são editados na wiki e serão a entrada do transformador;
2. **Transformação de modelos em código-fonte:** o transformador, que funciona na wiki, gera o código-fonte de acordo com o modelo de entrada;
3. **Colaboração:** a wiki permite que mais desenvolvedores possam editar o mesmo modelo usando uma política *pessimistic*;

¹O metamodelo pode ser alterado na wiki, assim como as transformações, mas a princípio está sendo definido fora da wiki e esta funcionalidade ainda não foi implementada.

4. **Versionamento de modelos:** a wiki armazena um histórico dos modelos com a possibilidade de comparar as versões e retornar às edições anteriores;
5. **Controle de acesso por grupos:** a wiki permite criar um grupo com permissão de edição de modelos e outro com permissão de apenas visualização, por exemplo;
6. **Comentários:** há dois tipos de comentários. Um é inserido em cada versão alterada e o outro é inserido na página editada. A figura 3.3 ilustra comentários feitos na página de edição do modelo, ou seja, na mesma página em que o usuário está editando o modelo ele pode deixar um comentário no final da página. A figura 3.4 ilustra comentários específicos para cada versão da página.

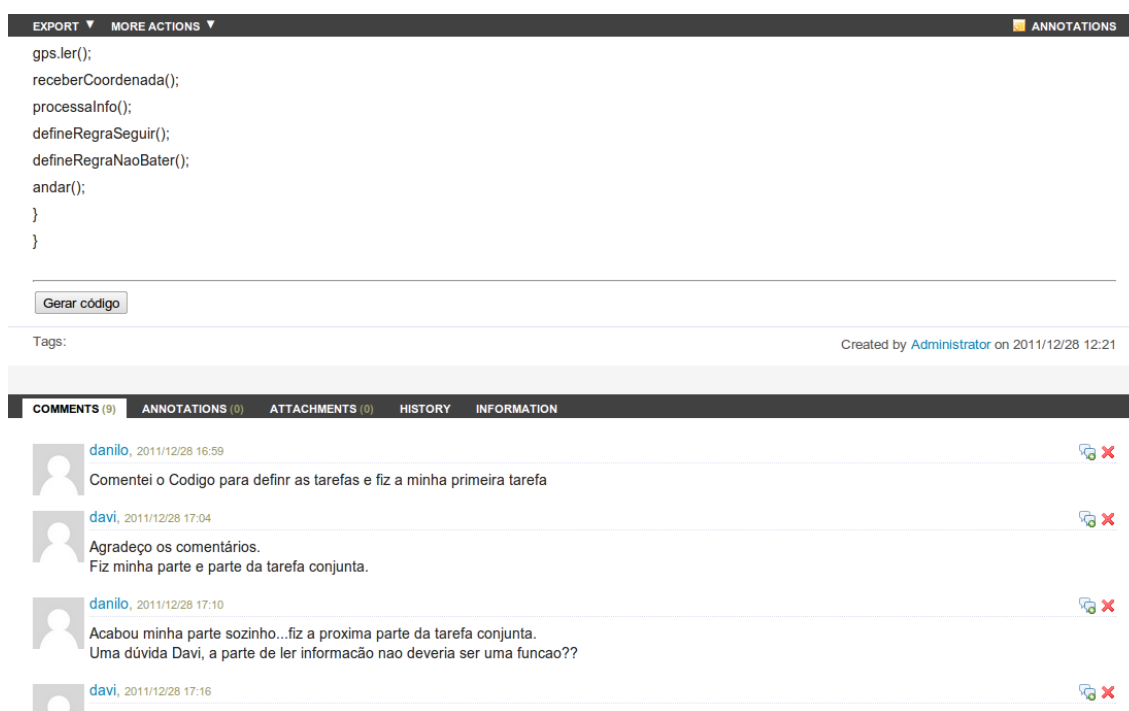


Figura 3.3: Comentários feitos na página de edição

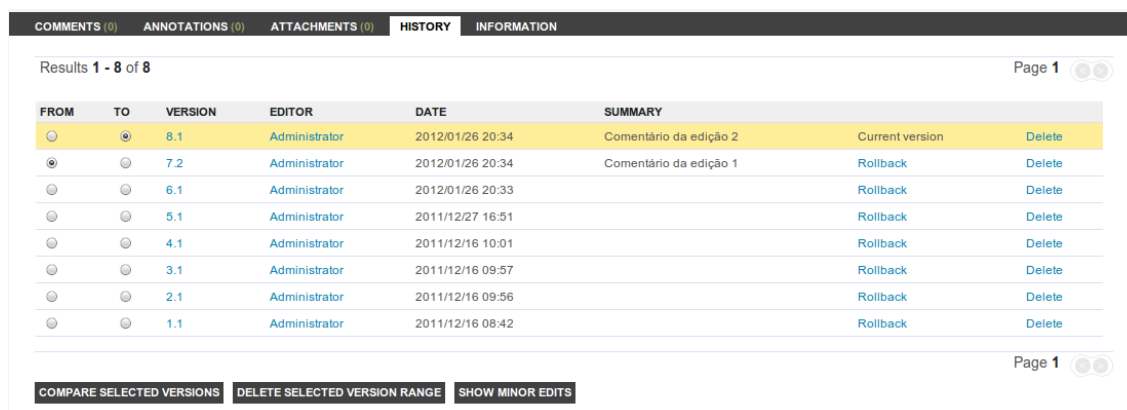


Figura 3.4: Comentários feitos para cada versão da página

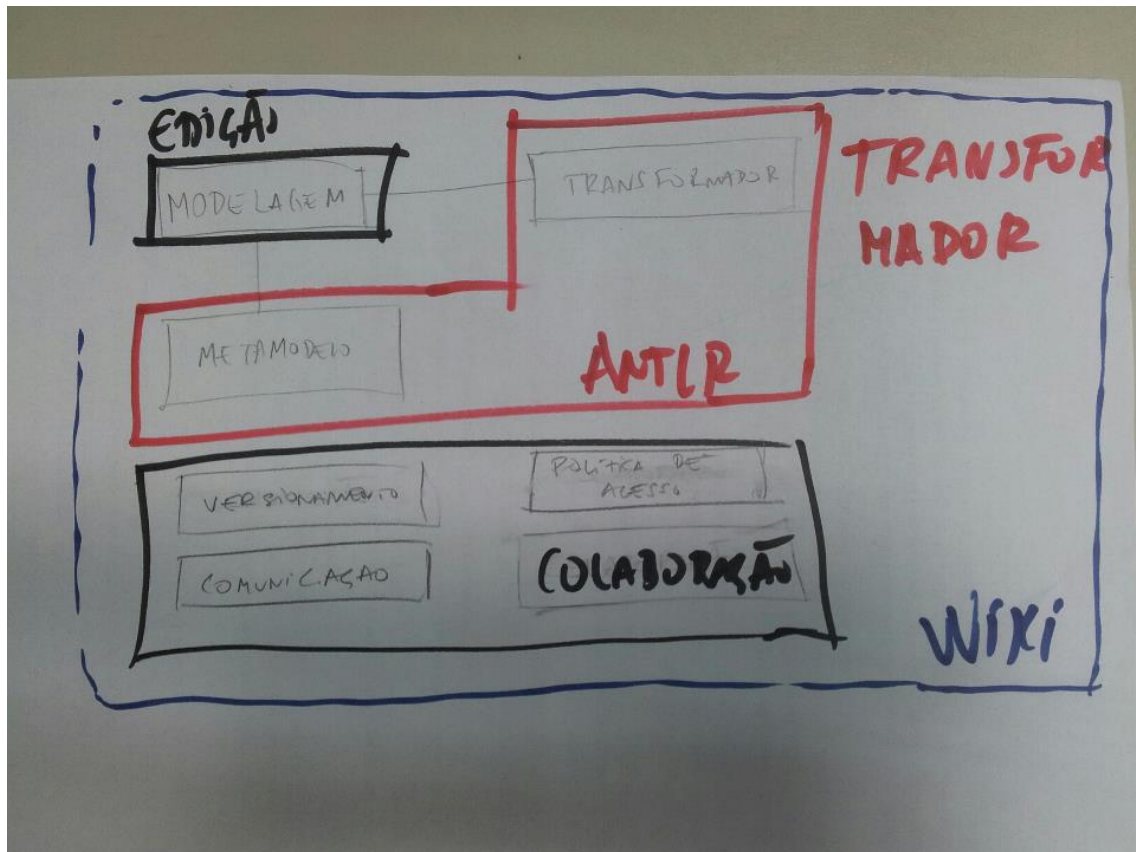


Figura 3.6: Arquitetura do CoMDD.

3.3 Método

Para tentar evidenciar que a hipótese é possível/verdadeira foram executados três estudos de caso.

3.3.1 Estudo de Caso I

O objetivo do primeiro estudo de caso foi avaliar o CoMDD por pessoas não envolvidas. A avaliação consistia em verificar como participantes jovens, com relativa baixa formação, sem conhecimentos de programação e sem conhecimentos do domínio, conseguiriam entender os conceitos e a linguagem do CoMDD.

3.3.1.1 Descrição dos participantes

Alunos de pré-iniciação científica do Laboratório de Robótica Móvel da Universidade de São Paulo. Três deles tinham 16 anos e um 17 anos, sendo dois garotos e duas garotas. Todos os quatro haviam concluído o segundo colegial. Já tinham programado para o robô SRV⁴ com o intuito

⁴vem o site do srv que por algum motivo não tá compilando no latex!

educacional, usando uma API própria e simples. Não possuíam conceitos de algoritmos ou de programação orientada a objetos.

3.3.1.2 Instruções e Problema Apresentado

Foram formadas duas equipes com dois integrantes cada. Cada equipe ficava em um único computador, ou seja, eram dois computadores com duas pessoas. A distribuição foi definida dessa forma para estimular a comunicação e facilitar a resolução do problema. Estima-se que se fosse uma pessoa por computador o experimento iria durar o dobro do tempo, se tornando inviável de conseguir pessoas para participar.

Antes de passar o comando foi lido junto com os participantes dois textos, o primeiro explicando o que é o CoMDD (ver *Texto I: O que é o CoMDD?* no apêndice A) e o segundo texto explicando como usar a DSL criada para o estudo de caso (ver *Texto II: Como programar para o CoMDD* no apêndice A.2).

Após a leitura, o seguinte comando foi passado:

O código é de um robô que ao receber uma lista de coordenadas deve ser capaz de passar por elas. Seu algoritmo de funcionamento é o seguinte:

Os sensores devem ler a informação

O robô recebe a próxima coordenada

O robô processa a informação

O robô recebe uma regra a aplicar

O robô anda

Esse algoritmo deve ficar sendo executado eternamente, até o robô ser desligado.

Agora você deve a partir do algoritmo e das explicações no site ser capaz de escrever um código usando a linguagem do CoMDD.

Requisitos:

O robô deve ser da plataforma pionner

O robô deve usar os pacotes de localização e o do player

O robô deve usar um gps e uma bússola

O robô seguir um conjunto de coordenadas

O tempo entre o momento que o pesquisador começou a falar com os participantes até o final do experimento foi de aproximadamente 2h45. Este tempo inclui a instrução do pesquisador aos participantes, a resolução do problema e a avaliação feita pelos participantes.

3.3.1.3 Resultados

A resposta correta é a seguinte:


```
plataforma pionner
robo david
adicionar includes
adicionar defines
importar pacote player;
importar pacote localizacao;
criarSensor gps
criarSensor bussola
    int main() {
        gps.ligar();
        bussola.ligar();
        carregarListaCoordenadas();
        inicializarPlayer();
        while(true) {
            bussola.ler();
            gps.ler();
            receberCoordenada();
            processaInfo();
            defineRegraSeguirMultiplasCoordenadas();
            andar();
        }
    }
```

3.3.1.3.1 Resultados da equipe A A equipe A apresentou a seguinte resposta, observando que a indentação apresentada é proporcional à original:

```
plataforma pioneer
robo MLJ123
adicionar defines
adicionar includes
adicionar while

importar pacote player;
importar pacote localizacao;

criarSensor gps
criarSensor bussola

int main() {
    «
    gps.ligar();
    bussola.ligar();
    carregarListaCoordenadas();
    inicializarPlayer();
    while(true) {
        gps.ler();
        bussola.ler();
        carregarListaCoordenadas();
        receberCoordenada();
        processaInfo();
        defineRegraSeguirMultiplasCoordenadas();
        andar();
    }
    »
}
```

3.3.1.3.2 Resultado da equipe B A equipe B apresentou a seguinte resposta, observando que a identificação apresentada é proporcional à original:

```
plataforma pioneer
importar pacote player;
importar pacote localizacao;
criarSensor gps
criarSensor bússola
carregarListaCoordenadas();
int main()
«Inicializações
Loop
Funções
»
```

3.3.1.3 Observações (trocar esse nome)

1. Ambos grupos receberam as instruções pelo pesquisador por vídeo conferência;
2. Durante todo o experimento, o grupo B teve suporte via vídeo conferência, enquanto que o grupo A teve suporte via *instant messenger* e o auxílio de um aluno de doutorado do LRM que estava no local;
3. O grupo A leu uma segunda vez o texto II;
4. O grupo A teve melhores resultados que o grupo B. Os resultados do grupo A são considerados como corretos, enquanto que os do grupo B estão errados.

3.3.1.4 Opiniões dos participantes

Prós: A linguagem é fácil de entender e o fato de ser em português facilita.

Contras: Foi mais complicado entender as instruções do experimento; ficou confusa a função *carregarListaCoordenadas()*.

3.3.1.5 Conclusões e observações do Estudo de Caso I

A seguir as conclusões e observações do primeiro estudo de caso realizado:

1. O experimento foi transmitido a distância, via vídeo conferência e com a ajuda de um aluno de doutorado do LRM que acompanhou os participantes e auxiliou o grupo A, enquanto o pesquisador auxiliava o grupo B. Não pode-se precisar quanto foi o auxílio que o grupo A recebeu;
2. Ao observar as equipes estima-se a necessidade de pelo menos duas horas adicionais para que eles aprendessem a usar e instalar o Eclipse e SVN;

3. Os participantes tinham dificuldade ou preguiça de lerem o texto que ensinava como programar para o CoMDD (A.2)
4. Os grupos, principalmente o grupo B, tinham o hábito de perguntarem constantemente se o que faziam estava certo;
5. Os participantes copiavam do texto A.2 e colavam na página de edição;
6. Era esperado que mesmo os participantes sem conhecimento de programação teriam mais facilidade em aprender a linguagem.

3.3.1.6 Lições aprendidas

Como aprendizado do estudo de caso I é necessário planejar melhor em dois aspectos:

1. Organizar melhor a infra-estrutura: verificar quantos computadores serão necessários; se a comunicação for on-line, preparar os softwares necessários e orientar bem uma pessoa que possa ajudar e que esteja presente com os participantes. No caso deste experimento houve um mal planejamento da infra-estrutura e da orientação ao ajudante;
2. Para um experimento deste tipo o ideal é evitar ajudar os participantes após passar o problema. Dessa forma, tem-se uma visão mais real de como não desenvolvedores se comportaria com o CoMDD. Os participantes estiveram muito dependentes do pesquisador e estima-se quatro hipóteses não excludentes que expliquem essa dependência: (i) relativa baixa instrução dos participantes, (ii) ruídos provocados pela distância, (iii) mal planejamento ou instrução por parte do pesquisador e/ou (iv) a disposição do pesquisador para tirar dúvidas.

3.3.2 Estudo de Caso II

O objetivo do segundo estudo de caso foi avaliar comparativamente, entre desenvolvedores, o CoMDD com a abordagem tradicional. A avaliação consiste na coleta de opinião por parte destes usuários. Neste caso a abordagem tradicional é definida pelo uso do Eclipse com o plugin SVN instalado e será referenciada por "Eclipse+SVN" como sinônimos.

Para realizar este estudo de caso foi implementado para o Eclipse, a mesma linguagem usada no CoMDD. O Eclipse possui dois projetos: o Xtext⁵, usado para definir a linguagem e o Xpand⁶ usado para definir as transformações;

⁵<http://www.eclipse.org/Xtext/>

⁶<http://wiki.eclipse.org/Xpand>

3.3.2.1 Descrição dos participantes

Todos os quatro participantes são formados em ciências da computação, atuaram na indústria desenvolvendo sistemas e estão cursando mestrado ou doutorado na Universidade de São Paulo em ciências da computação.

3.3.2.2 Instruções e Problema Apresentado

Os quatro integrantes foram divididos em duas equipes: equipe A (usando o CoMDD) e equipe B (usando Eclipse+SVN). Ambos integrantes de cada equipe estavam em locais (mais especificamente cidades) diferentes na hora do experimento. A equipe A realizou o experimento em um dia diferente da equipe B. O experimento foi passado por um software de conferência por voz.

Antes de passar o comando todos os grupos receberam instruções comuns e instruções específicas. As instruções comuns foram:

1. Ler o texto base. O texto base da equipe A é o *Texto I: O que é o CoMDD?* (ver A.1) e da equipe B é o *Texto IB: Instalação do Eclipse+SVN e importação da DSL* (ver A.3);
2. Ler o *Texto II: Como programar para o CoMDD* (ver A.2);
3. Pode-se comentar na edição do modelo usando "//";
4. A função `carregarListaCoordenadas()` já está configurada para a lista de coordenadas, desta forma vocês não devem se preocupar em passar um lista de coordenadas, apenas chamar a função;
5. O desenvolvimento é feito em turnos e em cada turno apenas um desenvolvedor pode trabalhar no código (editá-lo), para evitar conflitos e *merges*. Cada turno tem 3 minutos para desenvolver e mais 2 minuto para comentar no CoMDD (equipe A) ou fazer o *commit* (equipe B). Por ex:
 - 0'-3': O desenvolvedor 1 começa programando enquanto o desenvolvedor 2 fica esperando (enquanto isso os desenvolvedores podem se comunicar via email ou gtalk);
 - 3'01s - 5': o desenvolvedor 1 tem de salvar suas alterações e comentar o código caso ache necessário;
 - 5'01s - 6': ninguém faz nada para que o servidor atualize os dados;
 - 6'01s-9': o desenvolvedor 2 assume o comando de edição do código e o desenvolvedor 1 fica aguardando ou conversando via e-mail ou gtalk com o desenvolvedor 2 caso este precise;
 - 9'01s-11': o desenvolvedor 2 tem de salvar suas alterações e comentar o código caso ache necessário;
 - 11'01s-14': o ciclo se repete

O seguinte comando foi passado:

O código é de um robô que ao receber uma lista de coordenadas deve ser capaz de passar por elas. Seu algoritmo de funcionamento é o seguinte:

Os sensores devem ler a informação

O robô recebe a próxima coordenada

O robô processa a informação

O robô recebe uma regra a aplicar, de acordo com os requisitos

O robô atua no ambiente

Esse algoritmo deve ficar sendo executado eternamente, até o robô ser desligado.

Agora vocês devem a partir do algoritmo e das explicações no site serem capazes de escrever um código usando a linguagem do CoMDD.

Requisitos:

O robô deve ser da plataforma pionner _____ Dev1

O nome do robô deve ser uma concatenação dos nomes dos desenvolvedores _____ Dev2

O robô deve usar os pacotes de localização _____ Dev1

O robô deve usar os pacotes do player _____ Dev2

O robô deve usar um gps _____ Dev1

O robô deve usar uma bússola _____ Dev2

O robô seguir um conjunto de coordenadas _____ Dev1 + Dev2

3.3.2.3 Equipe A: CoMDD

Além das instruções comuns apresentadas na seção 3.3.2.2, para equipe A foram apresentadas as seguintes instruções:

1. Pode-se usar todas as funcionalidades da wiki como comentários, histórico e comparação de versões⁷;
2. Dar preferência para comunicação que a própria wiki fornece: os comentários, caso contrário usar um instant messenger ou o e-mail e depois fornecer o log para o pesquisador

3.3.2.4 Equipe B: Eclipse+SVN

Além das instruções comuns apresentadas na seção 3.3.2.2, para equipe B foram apresentadas as seguintes instruções:

⁷Um exemplo de edição normal (sem ser usando a DSL) foi apresentado para que os participantes pudessem saber como e onde editar, comentar e comparar versões.

1. Pode-se usar todas as funcionalidades do plugin SVN como check-in check-out, comentários, histórico e comparação de versões⁸;
2. Dar preferência para comunicação que o próprio SVN fornece: os comentários, caso contrário usar um instant messenger ou o e-mail e depois fornecer o log para o pesquisador
3. Não usar as funções comuns de uma IDE como highlight, auto-complete e etc⁹¹⁰

3.3.2.5 Resultados

A resposta correta é a mesma apresentada na seção 3.3.1.3.

3.3.2.5.1 Resultados da Equipe A O pesquisador acompanhou a equipe e foi responsável por avisar quando um desenvolvedor podia editar ou devia esperar, para evitar conflitos de edição, contudo o tempo apresentado nas instruções não foi rígido. A equipe A apresentou a seguinte resposta:

⁸Os participantes já conheciam o plugin.

⁹Assim as equipes ficam mais iguais, uma vez que no CoMDD essas funcionalidades não foram implementadas.

¹⁰O Eclipse tem um ambiente que não usa as funções citadas.

```
//O robô deve ser da plataforma pionner - Danilo
plataforma pionner

//O nome do robô deve ser uma concatenação dos nomes dos desenvolvedores - Davi
robo davnilo

//O robô deve usar os pacotes de localização - Danilo
importar pacote localizacao

//O robô deve usar os pacotes do player - Davi
importar pacote player;

//O robô deve usar um gps - Danilo
criarSensor gps

//O robô deve usar uma bússola- Davi
criarSensor bussola

//função principal do código
int main() {
bussola.ligar();
gps.ligar();

//O robô seguir um conjunto de coordenadas - Danilo + Davi
while(true) {
bussola.ler();
gps.ler();
receberCoordenada();
processaInfo();
defineRegraSeguir();
defineRegraNaoBater();
andar();
}
}
```

A figura 3.7 registra a comunicação realizada pelo grupo. O grupo usou os comentários da página de edição e não usou os comentários de versões. Observando a figura 3.7 pode-se dizer que o grupo usou esse comentário como um *instant messenger*.

3.3.2.5.2 Resultados da Equipe B O pesquisador acompanhou a equipe e foi responsável por avisar quando um desenvolvedor podia editar ou devia esperar, para evitar conflitos de edição,

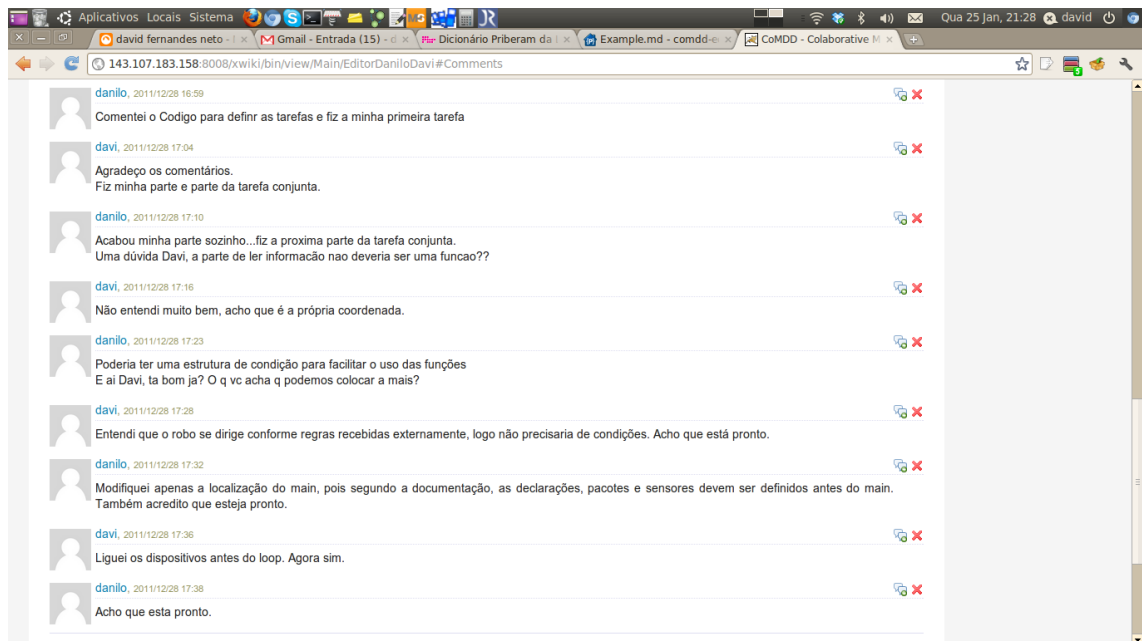


Figura 3.7: Log da conversa entre o desenvolvedor 1 e o desenvolvedor 2

contudo o tempo apresentado nas instruções não foi rígido. A equipe B apresentou a seguinte resposta:

```
// Definindo a plataforma do robo como pioneer
plataforma pioneer

// Define o nome do robo como uma concatenacao dos nomes dos desenvolvedores
robo filipeAlexandre

adicionar includes
adicionar defines

importar pacote localizacao;
importar pacote player;

criarSensor gps
criarSensor bussola

int main() {

    // Ligar o GPS
    gps.ligar();

    // Ligar Bussola
    bussola.ligar();

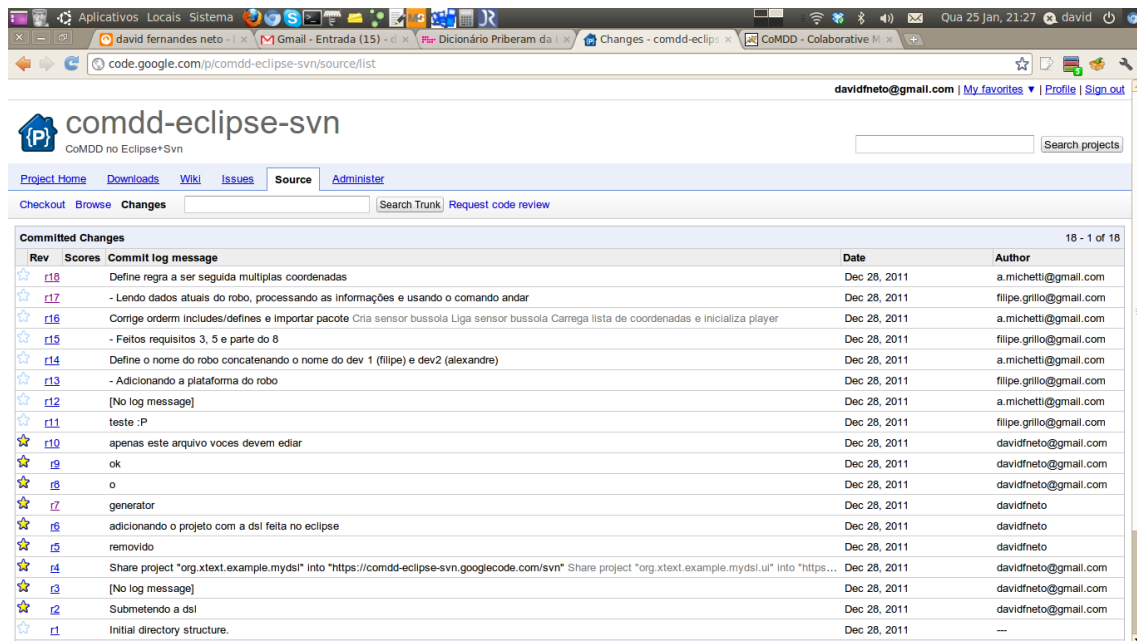
    // Carrega lista de coordenadas e inicializa Player
    carregarListaCoordenadas();
    inicializarPlayer();

    // Define que robo ira seguir um conjunto de coordenadas
    defineRegraSeguirMultiplasCoordenadas();

    // Loop
    while(true) {
        // Le os dados atuais
        gps.ler();
        bussola.ler();

        // Processa informacoes
        processaInfo();
        receberCoordenada();

        andar();
    }
}
```



Rev	Scores	Commit log message	Date	Author
r18		Define regra a ser seguida multiplas coordenadas	Dec 28, 2011	a.michetti@gmail.com
r17		- Lendo dados atuais do robo, processando as informações e usando o comando andar	Dec 28, 2011	filipe.grilio@gmail.com
r16		Corrige ordem includes/defines e importar pacote Cria sensor bussola Liga sensor bussola Carrega lista de coordenadas e inicializa player	Dec 28, 2011	a.michetti@gmail.com
r15		- Feitos requisitos 3, 5 e parte do 8	Dec 28, 2011	filipe.grilio@gmail.com
r14		Define o nome do robo concatenando o nome do dev 1 (filipe) e dev2 (alexandre)	Dec 28, 2011	a.michetti@gmail.com
r13		- Adicionando a plataforma do robo	Dec 28, 2011	filipe.grilio@gmail.com
r12		[No log message]	Dec 28, 2011	a.michetti@gmail.com
r11		teste :P	Dec 28, 2011	filipe.grilio@gmail.com
r10		apenas este arquivo voces devem editar	Dec 28, 2011	davidfneto@gmail.com
r9		ok	Dec 28, 2011	davidfneto@gmail.com
r8		o	Dec 28, 2011	davidfneto@gmail.com
r7		generator	Dec 28, 2011	davidfneto
r6		adicionando o projeto com a dsl feita no eclipse	Dec 28, 2011	davidfneto
r5		removido	Dec 28, 2011	davidfneto
r4		Share project "org.xtext.example.mydsl" into "https://comdd-eclipse-svn.googlecode.com/svn" Share project "org.xtext.example.mydsl.ui" into "https://comdd-eclipse-svn.googlecode.com/svn"	Dec 28, 2011	davidfneto@gmail.com
r3		[No log message]	Dec 28, 2011	davidfneto@gmail.com
r2		Submetendo a dsl	Dec 28, 2011	davidfneto@gmail.com
r1		Initial directory structure.	Dec 28, 2011	---

Figura 3.8: Log da conversa entre o desenvolvedor 1 e o desenvolvedor 2

A figura 3.8 registra a comunicação realizada pelo grupo. O grupo usou os comentários do check-in do SVN, e não usaram como um *instant messenger* e quando estavam acabando usaram um *instant messenger*. O log da conversa é apresentado no apêndice A.

Depois do experimento a equipe B foi apresentada ao CoMDD e então foi questionada pelo pesquisador.

3.3.2.6 Opiniões dos Participantes e Observações do Pesquisador

Serão apresentadas as opiniões dos participantes juntamente com as observações do pesquisador.

3.3.2.6.1 Equipe A Quando os participantes da equipe A foram questionados sobre a *linguagem* usada, eles responderam que precisava melhorar aspectos pontuais como deixar toda em inglês ou em português, mas que era fácil de aprender. Em relação aos *comentários* que a Wiki propicia eles são úteis mas sentiram falta de um *instant messenger*.

De uma maneira geral, a equipe gostou da abordagem do CoMDD principalmente pela facilidade de edição, flexibilidade e independência de softwares instalados. Eles acham a necessidade de instalar programas como IDEs e SVNs algo trabalhoso. Entretanto sentiram falta do processo de compilação, sendo esta a maior barreira apontada pela equipe A para usar o CoMDD.

A equipe A usou os comentários da página de edição, mas não os comentários de versionamento e comentaram no modelo. A comparação de modelos não foi necessária.

No apêndice A são apresentadas as perguntas feitas para a Equipe A com as respostas coletadas pelo pesquisador.

3.3.2.6.2 Equipe B Quando os participantes da equipe B foram questionados sobre a *linguagem* usada, eles responderam que era fácil e simples, com poucos comandos e com uma curva de aprendizagem rápida, mas que a parte dos *includes* estava confusa. Em relação aos *comentários* que o SVN propicia eles não foram úteis pois o problema apresentado era simples e por isso apenas os comentários no código e o diálogo no *instant messenger* era o suficiente.

De uma maneira geral, a equipe aprovaria o uso do CoMDD para um projeto de pequeno porte e apontaram como o costume a maior barreira para usar o CoMDD, pois os desenvolvedores estão acostumados com o uso de IDEs e SVNs.

A equipe B usou o check-in, check-out e comentaram no modelo. A comparação de modelos não foi necessária.

No apêndice A são apresentadas as perguntas feitas para a Equipe B com as respostas coletadas pelo pesquisador.

3.3.2.7 Conclusões e Observações do Estudo de Caso II

A seguir as conclusões e observações do segundo estudo de caso realizado:

1. Pode-se dizer que para ambas equipes os comentários ao longo do modelo foram importantes para permitir que duas pessoas editassem a mesma classe, podendo separar as atividades de cada um;
2. Ambas equipes conseguiram chegar na resposta correta, entretanto a equipe A com menos esforços por não precisarem instalar ferramentas;
3. A equipe A usou os comentários da página de edição para se comunicarem como se fosse um *instant messenger*. Através desses comentários cada integrante da equipe estava ciente do que o outro estava fazendo e puderam compartilhar conhecimento para resolver a tarefa, compartilhando dúvidas e conclusões. Essa comunicação fica registrada na página do modelo, o que pode ser útil para documentação;
4. A equipe B usou um *instant messenger* quando estavam terminando de resolver o problema para concluírem se já haviam terminado de resolver o problema ou não;
5. Um ponto positivo do SVN é que ele "obriga" que o usuário comente ao fazer o check-in, entretanto a leitura dos comentários da wiki é mais simples;
6. Para o problema dado, desconsiderando os recursos de uma IDE, os quatro participantes das duas equipes usariam o CoMDD no lugar do Eclipse+SVN;
7. Os quatro participantes das duas equipes concordaram que o CoMDD é melhor para usuários não desenvolvedores do que o Eclipse+SVN.

3.3.2.8 Lições aprendidas

O estudo de caso II foi necessário para se fazer uma análise qualitativa das abordagens, contudo foi necessário um esforço grande para reunir pessoas que estivessem dispostas a realizar um experimento que durasse em torno de 2 ou 3h. O experimento exigiu e um esforço ainda maior para realizar os experimentos, pois para isso o pesquisador teve que além de preparar o experimento, acompanhar cada uma das equipes durante a execução.

A partir dessas observações surgiu a ideia de fazer dois vídeos: um apresentando o CoMDD e o outro apresentando o Eclipse+SVN. Esses vídeos seriam transmitidos para desenvolvedores da indústria, preferencialmente, e seriam acompanhados de um questionário. Assim teria-se, além de um estudo qualitativo, um estudo quantitativo. Esse foi o princípio de formação do estudo de caso 3.

3.3.3 Estuo de caso 3: análise de vídeos e planilha de ticar

BLAHBLHABLHA...

highlight, auto complete sao coisas simples de se acrescentar, mas o mais importante é observar que esses foram os principais empecilhos de uso do CoMDD, ou seja, se as pessoas tivessem essas funcionalidades implementadas numa wiki possivelmente usariam o comdd. -> Talvez venha concluir isso

A DSL do CoMDD poderia aceitar algumas partes do modelo linguagens de programação como C, C++ ou Java e não realizar transformações nessas partes. Assim, a DSL teria mais aceitação entre os usuários.

3.4 Trabalhos Relacionados

Adessowiki

Os trabalhos de Lotufo et al (Lotufo et al., 2009) e de Machado et al. (Machado et al., 2011) apresentam o Adessowiki¹¹, um ambiente colaborativo para desenvolvimento, documentação, ensino e repositório de algoritmos científicos, com foco na área de processamento de imagens. Este ambiente está sendo usado desde 2008 como um ambiente educacional e como uma ferramenta colaborativa para escrita de artigos e outros textos científicos.

Segundo os autores, o que torna o Adessowiki especial em relação às outras wikis e aplicações Web é sua capacidade de incluir pedaços de código em *Python* ou *C++* que são executados no servidor quando as páginas wiki são renderizadas. Desta forma, os resultados deste código podem fazer parte de textos científicos, em forma de figuras, textos e tabelas.

¹¹<http://www.adessowiki.org>

O principal objetivo deste ambiente é desenvolver colaborativamente algoritmos numéricos. As páginas do Adessowiki permitem a inserção de código, o qual é executado quando a página é renderizada, incorporando figuras, textos e tabelas, gerando um documento científico.

Em relação ao ensino, o Adessowiki pode, como wiki, servir para registrar informações do curso como notas, exercícios e etc; como um ambiente de programação, serve de plataforma para atividades de programação solicitadas pelo professor, uma vez que o código pode se inserido e os resultados visualizados nas páginas wiki.

O Adessowiki se assemelha ao CoMDD por ser uma wiki para edição colaborativa no desenvolvimento de código (produto final do CoMDD), entretanto a principal diferença entre as abordagens está no foco dado por cada uma. Enquanto o Adessowiki tem mais o caráter de ensino, de pesquisa científica, de produção de textos científicos, e se limita à área de processamento de imagem e computação gráfica; o CoMDD foca no desenvolvimento de sistemas de software usando modelos, incentivando a colaboração de desenvolvedores ou não e com o intuito de ser uma espécie de substituto à IDEs e versionadores instalados localmente.

Ainda, o método usado pelo CoMDD está na comparação entre a abordagem CoMDD e a tradicional, enquanto que os autores do Adessowiki apresentam os casos de uso que o ambiente atende

Galaxy Wiki

O Galaxy Wiki permite, em uma Wiki, que desenvolvedores não só possam navegar, criar, modificar e deletar código fonte, como também compilar, executar e debugar programas Java, e ainda colaborativamente.

Os autores desenvolveram um plugin para o Eclipse de modo que o Eclipse acesse as páginas de projeto, de classes e de bibliotecas armazenadas no Galaxy Wiki e convertem em projetos e arquivos Java e JARs, respectivamente, integrando assim o Galaxy Wiki e o Eclipse.

Contudo, o CoMDD se difere do Galaxy Wiki por trazer a ideia de MDD ao invés do uso de Java. Assim, com o uso de modelos, tem-se mais incentivo à colaboração. Além do que Xiao et al., não destacam as vantagens do Galaxy Wiki em relação à uma IDE associada à um sistema de versionamento, como este trabalho faz.

3.5 Cloud 9 IDE

O Cloud 9 IDE¹² é uma IDE on line. Ele permite as mesmas coisas que uma IDE normalmente tem, como edição de código-fonte com *syntaxe highlight*, compilação, debug, versionamento de código usando Github¹³ e outros. Ainda é um projeto open-source que o CoMDD pode usar para melhorar.

¹²<http://c9.io>

¹³<https://github.com/>

O Cloud 9 IDE parece ser uma ferramenta de desenvolvimento muito robusta e completa, além de ser totalmente on-line, entretanto, ela ainda trabalha com código-fonte ao invés de modelos e usa uma ferramenta de versionamento estilo tradicional.

3.6 Google Docs e ferramentas de edição concorrente em tempo real

Parte do CoMDD, a qual não foi implementada, levanta a questão de edição concorrente em tempo real, o que significa que duas ou mais pessoas podem digitar, ao mesmo tempo, o mesmo documento, e visualizar as alterações em tempo real do documento alterado. Um exemplo de ferramenta assim é o Google Docs¹⁴, mas ainda há outras, como: o Thoughtslinger¹⁵, o Zoho¹⁶, o Moondedit¹⁷ e o Notapipte¹⁸. Entretanto, todos com exceção do último, são para edição de documentos de texto e não para modelos ou linguagens de programação com o foco de desenvolvimento de software. O Notapipte permite editar um texto com syntax highlight, mas ainda é um texto e não pode ser compilado ou executado na Web.

Então, pode-se dizer que não foi encontrado uma ferramenta que permitisse o desenvolvimento de software concorrente e em tempo real.

3.7 Artigos

1-Sobre o que fala o trabalho deles?

2-Quais as semelhanças?

3-Qual seu diferencial?

Trabalhos da Quali!!!!????

¹⁴<https://docs.google.com>

¹⁵<http://www.thoughtslinger.com/>

¹⁶<https://writer.zoho.com/>

¹⁷<http://moonedit.com/>

¹⁸<http://notapipe.net/>

Conclusão

4.1 Conclusões - verificar com carinho

A abordagem teve uma aceitação de X% por parte dos 42 usuários entrevistados...

"O problema descrito na sessão x foi resolvido como demonstrado nas sessões w e z, em que foi desenvolvido uma abordagem para tratar as situações mencionadas."concluir a hipótese da seguinte forma, mais ou menos:

O CoMDD é uma abordagem que prega o MDD colaborativo. No caso foi usado uma wiki, mas poderia ser qualquer plataforma web de simples uso e que promovesse velocidade e colaboração. Com as atuais tecnologias a wiki é a que melhor se encaixa nesse perfil.

Para o problema apresentado o CoMDD mostrou ser capaz de atingir os mesmo resultados que o Eclipse+SVN.

As wikis precisam evoluir mais para que possam, de fato, serem usadas como ferramentas de desenvolvimento e não se limitarem à documentação e afins.

Este trabalho está longe de criar um ambiente de desenvolvimento tão robusto e avançado quanto hoje são as IDEs, os SVNs, os *Bug Trackers* e etc. Até porque essas ferramentas já atingiram um grau de maturidade que as torna usáveis para produção. Este trabalho objetivou sim: mostrar que o alinhamento de duas tecnologias aparentemente não interligáveis é possível e benéfico, além de motivar novos estudos na área.

Os estudos de caso serviram para evidenciar que é possível ter uma abordagem que integre o MDD à uma Wiki e serviram também para mostrar que o CoMDD é aceitado como um possível substituto de abordagens tradicionais desde que traga algumas funcionalidades das IDEs.

O uso do CoMDD é mais fácil que o uso do Eclipse+SVN por não desenvolvedores. Portanto, pode ser interessante adotar o CoMDD quando não desenvolvedores tiverem que participar do desenvolvimento.

O CoMDD exige menos conhecimentos para colaborar do que são exigidos em um sistema de versionamento e ainda o tempo necessário para comentar algo é menor.

O MDD agrega um ganho de produtividade no desenvolvimento de software e ... ; modelos possibilitam mais comunicação principalmente por não especialistas; as wikis incentivam a colaboração devido sua simplicidade de edição e tornam o processo de colaborar/editar mais ágil. Portanto, conclui-se que com o CoMDD pode-se ter ganhos de produtividade e maior colaboração entre desenvolvedores e não desenvolvedores. Entretanto, esta conclusão não foi evidenciada pelos estudos de caso e sim pelo método dedutivo. As premissas são os benefícios do MDD e os benefícios da Wiki. A conclusão é de que o CoMDD herda os benefícios do MDD e da Wiki.

4.2 Contribuições

A principal contribuição deste trabalho é evidenciar que o uso de MDD associado a uma wiki é possível para resolver problemas da mesma forma que é possível usando uma IDE e um sistema de versionamento. E que usando essa abordagem a equipe pode ter ganhos de produtividade e intensificar a colaboração da equipe, principalmente por não desenvolvedores.

O CoMDD permite a edição de modelos, geração de código-fonte e a colaboração entre desenvolvedores e interessados sem a necessidade de ferramentas/ambientes individuais instalados nas estações de trabalho.

Por fim, este trabalho incentiva o uso de aplicações Web ao invés de aplicações locais, permitindo que uma pessoa possa trabalhar de qualquer computador.

Quando não é necessário ferramentas locais para se criar ou editar modelos, ganha-se em termos de velocidade e colaboração, pois menor tempo é necessário para uma pessoa começar a contribuir. Quando se facilita o processo de desenvolvimento/colaboração com o uso de uma wiki no lugar de uma IDE/sistema de versionamento, mais pessoas podem colaborar. Com o uso de modelos no lugar de linguagens de programação de alto nível, mais fácil torna-se o entedimento de não desenvolvedores e consequentemente mais eles podem participar.

»O CoMDD ainda poder muito útil para ensinar programação uma vez que não é necessário programas locais e o próprio CoMDD pode compilar/simular, mas para isso a ferramenta precisa ser melhor integrada. »Uma biblioteca de componentes em alto nível com a capacidade de geração de código.

»sincronismo entre modelos e documentação

MDD e tudo na web: mais produtividade // Modelos e wiki: aproximação maior de todos os envolvidos no processo, e com isso softwares mais próximos dos que o cliente precisa.

4.3 Trabalhos Futuros

Algumas perguntas não foram levantadas no decorrer deste trabalho e espera respondê-las em trabalhos futuros. São elas:

1. Pode-se dizer que a Wiki comportou-se bem com o MDD, contudo como ela se comportaria com uma abordagem tradicional orientada a código? Ou seja, como seria programar em Java, por exemplo, em uma wiki? E será que um código-fonte seria tão colaborativo quanto um modelo?
2. As equipes do segundo estudo de caso foram de dois integrantes apenas, como o CoMDD se comportaria com equipes maiores e com mais pessoas compartilhando o mesmo artefato?
3. O código-gerado pelo modelo foi validado por um especialista do domínio que ajudou e acompanhou a criação da DSL usada no CoMDD, contudo, como validar os modelos ao invés do código-fonte, de forma que garanta que o código-fonte gerado está correto e de forma que permita o retorno de erros aos desenvolvedores? – Acho que essa é uma questão mais de implementação.
4. Nesta abordagem as transformações e o metamodelo foram definidos fora da wiki, entretanto poderiam ser definidos pela própria wiki. Quais benefícios a edição colaborativa das transformações e dos metamodelos traria no desenvolvimento de software?
5. A abordagem do CoMDD implementada e avaliada contou com o desenvolvimento colaborativo seguindo uma abordagem *pessimistic*, contudo como a abordagem compartilharia-se caso tivesse outras formas de colaboração como: i. Edição concorrente e merge de modelos (adotando a abordagem *otimistic*; ii. Edição em tempo real estilo google docs; iii. Edição de outros artefatos que fossem dependentes¹.
6. Como seria o nível de aceitação caso o CoMDD tivesse funcionalidades mais próximas de uma IDE, como: *highlight*, auto-complet e link entre palavras, de forma que ao clicar em um método é apresentado a definição dele, por exemplo?
7. O estudo de caso II mostrou-se bem sucedido, entretanto para que outros casos o CoMDD pode ser útil?

4.4 Limitações do Trabalho

O CoMDD pode ser útil no desenvolvimento de algoritmos, contudo isso não foi testado.

¹Pode-se criar facilmente links de uma página para outra na wiki

O ideal seria agregar funcionalidades de uma IDE às wikis, permitir o desenvolvimento simultâneo e a edição em tempo real que hoje tem o google docs. Dessa forma o CoMDD teria mais chances de vigorar na comunidade de desenvolvedores.

Em relação ao domínio, espera-se que para qualquer domínio o CoMDD possa ser interessante, mas este trabalho não evidenciou isso.

O CoMDD não apresentava erros de sintaxe, o que deixava os desenvolvedores do estudo de caso II relativamente desorientados.

A abordagem pessimistica possui suas desvantagens como "In contrast, the pessimistic policy has some disadvantages. First, locks can cause an unnecessary serialization over the development process. Sometimes different developers can independently modify different parts of the same CI, but using a lock-based approach that is not possible. In addition, locks can cause a false sense of security (Collins-Sussman et al., 2004). Developers may think that their work will not be affected by the work of other developers due to the locks. However, sometimes CIs rely on other CIs that can be modified by other developers. This situation leads to indirect conflicts (Sarma et al., 2003). (ao Gustavo Prudêncio et al., 2012) Assim, um trabalho futuro seria no estudo de estratégias otimísticas ou até mesmo na adoção de ambas como propõe o trabalho de Prudencio et al. (ao Gustavo Prudêncio et al., 2012).

A DSL foi projetada para que fosse simples de forma que pessoas pudessem compreendê-la facilmente. Do contrário seria mais difícil realizar um estudo de caso. Entretanto, entende-se que uma DSL mais robusta e que atendesse mais casos estaria mais próxima da realidade.

Em relação ao domínio escolhido: o código gerado não podia ser testado uma vez que precisava ser inserido no robô, assim se o desenvolvedor pudesse simular o código gerado seria uma informação de retorno importante. Ainda, o cerne do domínio de robótica móvel autônoma são os algoritmos que tornam os robôs autônomos, entretanto a colaboração foi mais na definição dos sensores e atuadores que estes robôs usam, portanto a abordagem seria mais interessante para desenvolverem um algoritmo, só que para isso seria melhor uma edição concorrente estilo google docs. Assim, entende-se que o domínio escolhido foi beneficiado pelo uso de uma DSL, mas não pelo uso da colaboração.

4.5 Lições aprendidas

Uma sugestão é pensar bem no experimento antes de implementar. Acredita-se que teríamos implementado menos e teríamos sido mais diretos até mesmo nos estudos de caso se já soubéssemos exatamente o que estávamos avaliando e que resultados gostaríamos de alcançar.

Pesquisa orientada à teste. Assim como existe desenvolvimento de software orientado a teste onde voce primeiro pensa que testes vai fazer, é mto importante voce pensar em como vai provar sua proposta/hipótese. Uma vez uma mulher no WTD do SBSC me falou isso: como vc vai provar? Na época nao dei mta importancia, mas hoje vejo que é a primeira questao logo depois da

hipótese. Aí fica mais fácil até em saber o que vai implementar. Pode ir mais direto! É sempre crucial ter a visão do todo antes de fazer as partes. Ia fazendo as partes achando que no futuro tudo ia se encaixar. Bom, se encaixou, mas poderia ter sido mais rápido se fosse mais direto e soubesse de tudo antes... Assim foi no estágio da Alemanha. Ficava pensando: mas como isso que eu vou fazer se encaixa no projeto maior que o Daniel tá?! Pensei que depois ia se integrar e que de alguma forma aquilo que tava fazendo ia ter alguma utilidade, mas vi que nem ele sabia como se integrava as coisas. Nem ele tinha a visão geral. Essa é a diferença do cara que vai ser sempre um desenvolvedor e de um gerente de projeto/orientador. Este último tem a visão do todo. E o bom orientador/gerente, além da visão do todo, sabe muito bem do específico, das partes.

Mostre seu trabalho pra todo mundo! Durante o mestrado, na concepção das ideias! Aí as pessoas vão fazer perguntas que te farão pensar. Foi assim que conversando com o Artur pensei em algo que nem o Lucrécio tinha pensado antes! E saiba explicar por mais complexo que seja o que vc faz para qualquer pessoa! É sua tarefa saber simplificar e não do ouvinte saber entender conceitos complexos. O Artur e outros já diziam: se algo está complexo de mais é pq tá fazendo errado. Quebre em partes menores então! Essa é a filosofia Linux: quebre em pedaços tão pequenos a ponto de serem simples de fazer, não importa o quanto difícil e complexo o problema seja no começo.

o próprio Rafael e o Danilo me deram boas dicas mas já era no final do trabalho :P

e sempre faça questionários com perguntas que vc quer responder no seu trabalho. o lance do exp 3 que to fazendo parece ser muito bom!

cuidado com o que vc lê. Há muito lixo e artigos publicados que não tem referências ou dão ideias sem fundamento.... estão em inglês e "bons congressos

Referências Bibliográficas

- ABETI, L.; CIANCARINI, P.; MORETTI, R. Wiki-based requirements management for business process reengineering. In: *Wikis for Software Engineering, 2009. WIKIS4SE '09. ICSE Workshop on*, 2009, p. 14–24.
- AHO, P.; MERILINNA, J.; OVASKA, E. Model-driven open source software development - the open models approach. In: *4th International Conference on Software Engineering Advances, ICSEA 2009*, Porto, Portugal, 2009, p. 185–190.
Disponível em: <http://www.scopus.com/inward/record.url?eid=2-s2.0-70749149216&partnerID=40&md5=01413c76e574b96a0cc8d16043574d64>
- BEN COLLINS-SUSSMAN, B. W. F.; PILATO, C. M. Version control with subversion. Última visita em 11 de fevereiro de 2012., ???
Disponível em: <http://svnbook.red-bean.com/en/1.6/>
- BENDIX, L.; EMANUELSSON, P. Collaborative work with software models - industrial experience and requirements. In: *Model-Based Systems Engineering, 2009. MBSE '09. International Conference on*, 2009, p. 60–68.
- DEURSEN, A.; KLINT, P.; VISSER, J. Domain-specific languages: an annotated bibliography. *SIGPLAN Not.*, v. 35, n. 6, p. 26–36, 2000.
- ELRUFAIE, E.; TURNER, D. A. A wiki paradigm for use in it courses. *Information Technology: Coding and Computing, International Conference on*, v. 2, p. 770–771, 2005.
- FRANCE, R.; RUMPE, B. Model-driven development of complex software: A research roadmap. In: *29th International Conference on Software Engineering 2007 - Future of Software Engineering*, Minneapolis, MN, USA: IEEE Computer Society, 2007, p. 37–54.
- GUSTAVO PRUDÊNCIO, J.; MURTA, L.; WERNER, C.; CEPÊDA, R. To lock, or not to lock: That is the question. *Journal of Systems and Software*, v. 85, n. 2, p. 277 – 289, special issue

- with selected papers from the 23rd Brazilian Symposium on Software Engineering, 2012.
Disponível em: <http://www.sciencedirect.com/science/article/pii/S0164121211001063>
- HAILPERN, B.; TARR, P. Model-driven development: the good, the bad, and the ugly. *IBM Syst. J.*, v. 45, n. 3, p. 451–461, 2006.
- JANG, S.; GREEN, T. M. Best practices on delivering a wiki collaborative solution for enterprise applications. In: *Collaborative Computing: Networking, Applications and Worksharing, 2006. CollaborateCom 2006. International Conference on*, 2006, p. 1–9.
- JING, H.; FAN, Y. Usability of wiki for knowledge management of knowledge-based enterprises. *Knowledge Acquisition and Modeling, International Symposium on*, v. 0, p. 201–205, 2008.
- JONES, M.; SCAFFIDI, C. Obstacles and opportunities with using visual and domain-specific languages in scientific programming. In: *Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on*, 2011, p. 9–16.
- KLEPPE, A. G.; WARMER, J.; BAST, W. *Mda explained: The model driven architecture: Practice and promise*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.
Disponível em: <http://proquest.safaribooksonline.com/0-321-19442-X/ch01lev1sec1#X21udGVybmlFsX0ZsYXNoUmVhZGVyP3htbGlkPTAtMzIxLTE5NDQyLVgvdmlp>
- LIGGESMEYER, P.; TRAPP, M. Trends in embedded software engineering. *IEEE Software*, v. 26, p. 19–25, 2009.
- LOTUFO, R. A.; MACHADO, R. C.; KÖRBES, A.; RAMOS, R. G. Adessowiki on-line collaborative scientific programming platform. In: *Proceedings of the 5th International Symposium on Wikis and Open Collaboration, WikiSym '09*, New York, NY, USA: ACM, 2009, p. 10:1–10:6 (*WikiSym '09*,).
Disponível em: <http://doi.acm.org/10.1145/1641309.1641325>
- LOURIDAS, P. Using wikis in software development. *Software, IEEE*, v. 23, n. 2, p. 88–91, 2006.
- LUCRÉDIO, D. *Uma abordagem orientada a modelos para reutilização de software*. Tese de Doutorado, Universidade de São Paulo, Instituto de Ciências Matemáticas e de Computação, 2009.
- MACHADO, R. C.; RITTNER, L.; LOTUFO, R. A. Adessowiki - collaborative platform for writing executable papers. *Procedia Computer Science*, v. 4, n. 0, p. 759 – 767, <ce:title>Proceedings of the International Conference on Computational Science, ICCS 2011</ce:title>, 2011.

- Disponível em: <http://www.sciencedirect.com/science/article/pii/S1877050911001384>
- MAJCHRZAK, A.; WAGNER, C.; YATES, D. Corporate wiki users: results of a survey. In: *WikiSym '06: Proceedings of the 2006 international symposium on Wikis*, New York, NY, USA: ACM, 2006, p. 99–104.
- MCQUAY, W. Distributed collaborative environments for systems engineering. In: *Digital Avionics Systems Conference, 2004. DASC 04. The 23rd*, 2004, p. 9.D.3–91–10 Vol.2.
- MEHTA, N. *Choosing an open source cms: Beginner's guide*. Packt Publishing, 2009.
Disponível em: <http://amazon.com/o/ASIN/1847196225/>
- MEIJLER, T. Requirements for an integrated domain specific modeling, modeling language development, and execution environment. In: *in Proceedings of NWUML'2005: The 3rd Nordic Workshop on UML and Software Modeling*, 2005.
- MELLOR, S. J.; CLARK, A. N.; FUTAGAMI, T. Guest editors' introduction: Model-driven development. *IEEE Software*, v. 20, p. 14–18, 2003.
- SÁNCHEZ, P.; MOREIRA, A.; FUENTES, L.; ARAÚJO, J.; MAGNO, J. Model-driven development for early aspects. *Information and Software Technology*, v. In Press, Corrected Proof, p. –, 2009.
Disponível em: <http://www.sciencedirect.com/science/article/B6V0B-4XMD5H4-1/2/6ea38b8dfd8b3a555a3eacba8e25f226>
- SARMA, A.; NOROOZI, Z.; HOEK, A. Palantir: raising awareness among configuration management workspaces. In: *Software Engineering, 2003. Proceedings. 25th International Conference on*, 2003, p. 444 – 454.
- SELIC, B. The pragmatics of model-driven development. *IEEE Softw.*, v. 20, n. 5, p. 19–25, 2003.
- SRIPLAKICH, P.; BLANC, X.; GERVAIS, M.-P. Supporting collaborative development in an open mda environment. In: *Software Maintenance, 2006. ICSM '06. 22nd IEEE International Conference on*, 2006, p. 244–253.
- STAHL, T.; VOELTER, M.; CZARNECKI, K. *Model-driven software development: Technology, engineering, management*. John Wiley & Sons, 2006.
- TEPPOLA, S.; PARVIAINEN, P.; TAKALO, J. Challenges in deployment of model driven development. In: *Software Engineering Advances, 2009. ICSEA '09. Fourth International Conference on*, 2009, p. 15 –20.

TETARD, F.; PATOKORPI, E.; PACKALEN, K. Using wikis to support constructivist learning: A case study in university education settings. In: *HICSS '09: Proceedings of the 42nd Hawaii International Conference on System Sciences*, Washington, DC, USA: IEEE Computer Society, 2009, p. 1–10.

ZHANG, J.; RAY, I. Towards secure multi-sited transactional revision control systems. 2007, p. 365 – 375.

Disponível em: <http://www.sciencedirect.com/science/article/pii/S0920548906000742>

ZIMMERLY, B. Entendendo o software wiki. 2009.

Disponível em: <http://www.ibm.com/developerworks/br/library/os-social-mediawiki/index.html>

Implementação do CoMDD

CoMDD: como implantar uma dsl em uma wiki?

Apêndice C

Eclipse: xtext/xpand

Textos apresentados no experimento

Aqui são dispostos os textos apresentados no experimento e referenciados na seção 3.3

A.1 Texto I: O que é o CoMDD?

O que é o CoMDD? O CoMDD vem da sigla Colaborative Model Driven Development. É basicamente uma linguagem de programação que funciona em uma wiki

Certo, mas para que serve? Com o CoMDD você pode programar robôs! Isso mesmo! Robôs!

Então eu posso programar qualquer robô com ela? Não. O CoMDD é uma dsl, ou seja, uma Linguagem Específica de Domínio (do inglês Domain Specific Language). O que significa dizer que com ela você não programa qualquer coisa como o C, C++, Java e etc. Com o CoMDD você programa apenas para um domínio muito específico, o de robôs móveis autônomos. E pelo fato do CoMDD ser um protótipo em desenvolvimento, mesmo para o seu domínio ele é pouco tão versátil.

Tudo bem então, agora que entendi o que é o CoMDD, como posso programar para ele? Muito bem! Estamos felizes que o CoMDD esteja lhe interessando. Para você desenvolver nele conheça melhor ele aqui.

Fonte: <http://143.107.183.158:8008/xwiki/bin/view/Main/Tutorial>

A.2 Texto II: Como programar para o CoMDD

A-Estrutura A estrutura do código deve ser seguida dessa forma: 1-Declarações 2-Includes e Defines 3-Pacotes 4-Adicionando sensores 5-Main 5.1-Inicializações 5.2-Loop 5.3-Funções Nota Atente para as palavras. A linguagem é case sensitive, o que significa que as palavras devem ser escritas exatamente como são mostradas. Caso falte um "s", como por exemplo: adicionar define ao invés de adicionar defines o código gerado estará errado. 1-Declarações 1.1 - Plataforma Você pode desenvolver código para 3 plataformas. São elas pioneer, srv e carrinho de golfe. Veja como escrever: plataforma [tipo da plataforma]

Ex: plataforma srv Os tipos são pioneer, srv, golfe 1.2 - Nome do robô robo [nome qualquer] Ex: robo

XyZ33 2-Includes e Defines O CoMDD trás um conjunto de includes e defines. Para usá-los basta escrever: adicionar [includes ou defines] Ex: adicionar

defines 3-Pacotes Há dois pacotes disponíveis. São eles o player e o localizacao. Para usá-los basta escrever: importar pacote [tipo do pacote]; Ex: importar

pacote player; 4-Adicionando Sensores Para usar os 3 tipos de sensores (gps, bussola e camera) você deve adicioná-los antes no código, da seguinte forma: criarSensor

[tipo de sensor] Ex: criarSensor gps

5-Main A Main é a função principal do código, toda a lógica do código vai dentro desta função. Para isso você precisa escrever: int main() «Ini-

cializações Loop Funções

» O que está dentro de «» deve ser substituído de acordo com as

explicações dos itens 5.1, 5.2, 5.3. 5.1 - Inicializações Para você usar os sensores criados ou outras funções você deve iniciá-las antes. No caso do sensor: [tipo do

sensor].ligar(); Ex: gps.ligar(); Ainda é possível carregar uma lista de

coordenadas e inicializar o player da seguinte forma: carregarListaCo-

ordenadas(); inicializarPlayer(); 5.2 - Loop Loop é algo que vai ser

executado continuamente. Para usá-lo você deve escrever: while(true)

«Funções» Não se esqueça de fechar a chave depois de escrever

as funções. 5.3 - Funções São possíveis as seguintes funções, a explicação delas está após a seta

(->): gps.ler(); -> O sensor gps lê suas coordenadas bussola.ler(); -> O sensor bússola obtém a

sua orientação camera.ler(); -> O sensor câmera obtém a sua imagem carregarListaCoordenadas();

-> O robô carrega uma lista de coordenadas receberCoordenada(); -> O robô recebe uma coorde-

nada que deve seguir processaImagem(); -> Processa a imagem recebida pela câmera. Para usá-la

você deve antes obter a imagem da câmera. processaInfo(); -> Processa todas as informações dos

sensores. defineRegraSeguir(); -> Define a regra seguir um outro robô. defineRegraNaoBater();

-> Define a regra andar pelo ambiente sem bater. defineRegraSeguirMultiplasCoordenadas(); ->

Define a regra percorrer um conjunto de coordenadas dadas. andar(); -> O robô começa a se lo-

comover no ambiente de acordo com a regra definida. Para usar uma das funções basta escrever igual ao que está antes da seta

Fonte: <http://143.107.183.158:8008/xwiki/bin/view/Main/aqui>

A.3 Instalação do Eclipse+SVN e importação da DSL

Texto 1B: Instalação do Eclipse+svn e importação da dsl 1. Instalar o Eclipse Modeling Tools versão helios <http://www.eclipse.org/downloads/packages/eclipse-modeling-tools-includes-incubating-components/heliossr2> 2. A versão helios precisa do java 1.5 (pode-se colar o jre 1.5 direto na pasta do eclipse, assim não precisa alterar o path) 3. Instalar os plugins: xpanse e xtext 4. Instalar o svn 5. Configurar o svn no eclipse com o projeto <http://code.google.com/p/comdd-eclipse-svn/source/checkout>

Textos apresentados no experimento

A.1 Conversa da Equipe B do Estudo de Caso 2 realizada no final do experimento

Desenvolvedor 1: Eu acho que faz sentido o que temos lá
minha única dúvida é onde deveria ficar aquela função `carregarListCoordenadas()`
mas acho que ela faz sentido onde ela está

Desenvolvedor 2: eh eu tambem acho q talvez ela fique dentro do loop

Desenvolvedor 1: primeiro eu carrego tudo, defino a regra
e aí passo a receber coordenada por coordenada

Desenvolvedor 2: eu tambem acho
só faria sentido ficar dentro do loop, no caso de poder mudar o tipo de regra
durante a execução

mas isso não está nos requisitos

Desenvolvedor 1: exato

eu vou dar mais uma olhada na wiki
mas acho que é isso

Desenvolvedor 2: acho que sim
talvez o adicionar includes, adicionar defines, possa ser um comando so
algo como "adicionar dependencias"

ou algo assim

pq todo programa sempre teriam essas duas mesmas linhas

ou talvez ser adicionado automaticamente

mas assim também funciona

Desenvolvedor 1: sim

pra vc estar ciente do fato

Desenvolvedor 2: isso eh

outra coisa seria melhorar a documentação para incluir qual o simbolo que indica comentário, e quando se deve usar ou não ponto e virgula

Desenvolvedor 1: boa

Perguntas Respondidas pelas Equipes A e B do Estudo de Caso 2

A.1 Perguntas e respostas da Equipe A

A seguir são apresentadas as perguntas feitas aos participantes da Equipe A com suas respectivas respostas.

O que vocês acharam da linguagem? Desenvolvedor 1: está legal mas as funções estão muito genéricas, "includes" e "defines" estão confusos (redundante a parte de "adicionar includes" e "importar pacotes").

Desenvolvedor 2: os pacotes já deveriam estar todos inclusos sem a necessidade de importá-los.

Vocês mudariam algo na sintaxe? Desenvolvedor 1: a estrutura da linguagem é comum a C ou Java. É melhor deixar tudo em português ou tudo em inglês.

Desenvolvedor 2: a sintaxe está boa para quem sabe programar em C ou java, aí consegue entender, mas para quem nunca programou "()", "" e ";" não significam nada.

Os comentários que vocês escreveram foram úteis de alguma forma? Desenvolvedor 1: foram interessante. Mas no mundo real um instant messenger é mais útil caso os dois estejam programando na mesma classe. Mas para documentação ou quando os desenvolvedores não estão ao mesmo tempo e na mesma classe, os comentários da página são úteis.

Desenvolvedor 2: o sistema de comentário do SVN funciona melhor, porque ao dar o *commit* você é obrigado a comentar e ao dar o *check out* você é obrigado a ler. Na Wiki fica dissociado do código.

E o histórico das versões de edição, é útil? Desenvolvedor 1: É mais util quando alguém faz alguma besteira.

De uma maneira geral, o que vocês acharam da abordagem? Desenvolvedor 1: não é muito interessante uma vez que tenho que esperar pela pessoa terminar programar para depois programar. No tradicional os dois podem programar juntos e depois fazer o merge, mas se fosse possível fazer edição simultanea estilo o google seria interessante. **E se fosse em classes separadas ao mesmo tempo?** Aí é interessante. Programar na web também é bom e o histórico também. É bom a flexibilidade de estar em qualquer computador.

Desenvolvedor 2: programar na web é bom porque evita problemas de infraestrutura (a infraestrutura é instantanea, independente do computador).

Então seria interessante uma wiki ou seria melhor um outro site? Desenvolvedor 1: a wiki é um site mais dinâmico, mais fácil de editar, as pessoas já conhecem.

Desenvolvedor 2: poderia ser um site mais especializado.

O que vocês acharam da abordagem CoMDD e da abordagem Eclipse+SVN? Que comparação vocês fazem entre essas duas abordagens? Qual vocês preferem? Desenvolvedor 1: estar na web é mais flexível, configurar uma IDE, SVN e tudo o mais é trabalhoso. Desenvolvedor 2: em geral é melhor programar na wiki do que usar uma IDE, mas fica dependente da conexão. Aí seria interessante ter uma forma de usar a wiki mesmo sem internet.

Vocês usariam o CoMDD no lugar do Eclipse+SVN? Por que não? E se fosse para fazer programas como este, vocês usariam o CoMDD? Desenvolvedor 1: usaria desde que o processo de compilação funcionasse, agora sem compilar e sem executar não. Desenvolvedor 2: concordo.

A.2 Perguntas e respostas da Equipe B

A empresa de vocês seguia essa abordagem? Desenvolvedor 1 e Desenvolvedor 2 : usam o eclipse e o subversion.

O que vocês acharam da linguagem? Desenvolvedor 1: fácil e simples, poucos comandos. Desenvolvedor 2: curva de aprendizagem rápida, mas faltou prática, não sei se o código funciona. A sintaxe é simples. Outra coisa seria melhorar a documentação para incluir qual o simbolo que indica comentário, e quando se deve usar ou não ";".

Vocês mudariam algo na sintaxe? Desenvolvedor 2 e Desenvolvedor 1: a parte dos includes poderia ser uma linha só já que sempre vai precisar dela.

Como vocês colaboraram? O que foi mais importante, os comentários? Desenvolvedor 1 e Desenvolvedor 2: usamos apenas os comentários no código para nos comunicarmos. Os comentários do *check in* não usamos porque o código é pequeno. Desenvolvedor 1: o comentário do SVN é mais quando tem algum problema.

Os comentários que vocês escreveram no SVN foram úteis de alguma forma? Não usaram.

E o histórico, foi útil? Não usaram.

E quais funcoes do SVN vocês acham importantes além das que vocês usaram? Desenvolvedor 1: histórico caso tenha algum problema, merge caso editássemos ao mesmo tempo. Desenvolvedor 2: quando o código é muito extenso é importante comparar as versões de código para ver o que está sendo alterado

Vocês acham que a wiki atenderia à esses propósitos que o Eclipse+SVN atende? Desenvolvedor 1: eu nunca pensei numa wiki para editar código, usaria mais para colaboração na documentação. Desenvolvedor 2: senti falta de algumas coisas como auxílio de sintaxe, *auto-complete*, *import* automático, clicar em um arquivo e ir para outro. A parte de comentário, versionar, editar o texto é praticamente a mesma coisa. Mas não sei como me comportaria caso pudesse editasse ao mesmo tempo duas pessoas.

O que vocês acharam da abordagem CoMDD e da abordagem Eclipse+SVN? Que comparação vocês fazem entre essas duas abordagens? Qual vocês preferem? Desenvolvedor 1: to acostumado com o Eclipse+SVN, mas uso tambem o controle de versão integrado com um *bug-track*. Desenvolvedor 2: muita coisa já tem pronta pro Eclipse e acho que o costume é a principal barreira, pensando como programador ainda usaria o Eclipse. Mas pensando que usaríamos uma dsl e não Java, talvez o CoMDD fosse melhor.

Mas não seriam muitos programas necessários para, às vezes, resolver um problema simples? Desenvolvedor 1 e Desenvolvedor 2: Para um projeto pontual e pequeno o CoMDD supriria mas para algo maior e complexo acho que não.

Detalhes de implementação

aqui vem todos os programas usados, os scripts, o procedimento....

Resultados do Aluno

Este apêndice cita atividades realizadas durante o período do mestrado e que se relacionam a este, mas não necessariamente diretamente com o trabalho de pesquisa.

ver lattes

Carta do mr Schatz, DSLs, screencasts, artigos, participação no INCT, curso de LPS, pae entre outras atividades.