Uma Abordagem Distribuída para Refatoração Automática de Código Fonte Guiada por Atributos de Qualidade

Heliomar Kann da R. Santos, Leonardo G. P. Murta, Viviane T. da Silva

Instituto de Computação – Universidade Federal Fluminense (UFF)
São Domingos Niterói - RJ – Brasil - CEP: 24210-240
{hkann, leomurta, viviane.silva}@ic.uff.br

Resumo. *É comum que equipes de desenvolvimento dediquem grande parte do tempo a manutenções de funcionalidades pré-existentes. Contudo, como diversas dessas manutenções são emergenciais, o software tende a degradar com o passar do tempo, gerando efeitos colaterais nos seus requisitos não funcionais. Desta forma, este trabalho propõe o monitoramento contínuo de alguns atributos de qualidade do software, por meio de métricas, e a execução de manutenções preventivas (i.e., refatorações) automáticas para evitar sua degradação. Este trabalho será avaliado por meio de um estudo experimental no contexto do sistema de gestão acadêmica da Universidade Federal Fluminense, que tem em torno de 30 desenvolvedores e 300.000 linhas de código.*

Abstract*. Usually, development teams devote a huge amount of time on maintaining preexisting features. However, due to the fact that many of these maintenances are not planned, the software tends to degrade over time, causing side effects in its non-functional requirements. Thus, this paper proposes the continuous monitoring of some quality attributes of the software through metrics and the execution of automatic preventive maintenance (i.e., refactorings) to avoid its degradation. This work will be evaluated through an experimental study in the context of an academic management system at Fluminense Federal University, which has about 30 developers and 300,000 lines of code.*

Palavras-chave. *Refatoração, métricas, gerência de configuração, agentes de software.*

1. Caracterização do problema

Desenvolver software é usualmente diferente do desenvolvimento de outros produtos [Brooks 1987]. A manufatura de um software é algo muito barato, bastando, por exemplo, copiar o instalador para um CD, ou disponibilizar um serviço na Web, o que o distingue da manufatura de casas, automóveis, produtos agrícolas, etc. No entanto, um dos fatos que tornam o desenvolvimento de software árduo e diferente de outros produtos é a aparente facilidade de evolução. Esse fato leva muitas equipes de desenvolvimento a dedicarem grande parte do seu tempo evoluindo e mantendo funcionalidades. Porém, muitas equipes de desenvolvimento não estão aptas, por diversas razões (e.g., inexperiência ou falta de recursos), de elaborar uma arquitetura

capaz de acolher modificações com facilidade, gerando envelhecimento precoce do software em desenvolvimento [Parnas 1994]. Este cenário, quando levado ao extremo, pode tornar o software muito difícil de ser mantido, chegando a um ponto em que a criação de uma nova funcionalidade pode ser inviável devido aos custos envolvidos.

Além disso, é importante observar que na grande maioria dos ambientes de desenvolvimento de software, as equipes trabalham usualmente por períodos de 8 horas diárias, deixando os recursos computacionais ociosos no restante do tempo. Este fato pode ser visto como uma oportunidade de utilização do parque computacional ocioso para fazer um monitoramento contínuo do software em desenvolvimento, com o intuito de melhorar sua qualidade. É importante notar que a utilização de recursos computacionais durante períodos de ociosidade não é algo novo. Por exemplo, iniciativas como *World Community Grid* [IBM 2010] visam à descoberta da cura de doenças de forma distribuída, rodando em PCs ociosos de usuários comuns.

2. Fundamentação teórica e trabalhos relacionados

A Gerência de Configuração do Software (GCS) é uma importante subárea de Engenharia de Software que trata da evolução controlada do software [Dart 1991]. Para controlar a evolução, a GCS utiliza principalmente Sistemas de Controle de Versão (SCV), sistemas de controle de modificações e sistemas de gerenciamento de construção e entrega do software [Murta 2006]. Esses sistemas podem ser apoiados por ferramentas ou executados manualmente.

Em um sistema de controle de versão centralizado, as operações mais comuns são: ***checkin***, representando a disponibilização de artefatos para o controle de versões e acarretando na chegada dos artefatos no repositório do SCV; ***checkout***, operação que obtém alguma versão do repositório para um espaço de trabalho; ***update***, representando a atualização de um espaço de trabalho com a versão mais atual do repositório. Mais detalhes sobre sistemas de controle de versão podem ser encontrados em Leon [2000].

Para um controle eficiente de versões é necessário utilizar alguma estratégia de ramificação [Walrad *and* Strom 2002], pois os projetos possuem particularidades de uso que necessitam ser mantidas. Para um software que é distribuído e comercializado em diversas versões, a criação de um ramo por versão seria interessante. Já para um sistema web, a criação de ramos para desenvolvimento e manutenção juntamente com marcações de *releases* (*tags*) poderia ser uma estratégia mais adequada, pois a distribuição é feita através da implantação do sistema em servidores e os usuários, dessa forma, utilizam apenas a versão mais atual do sistema.

O uso de repositórios para controle de versão é considerado fundamental em qualquer cenário de desenvolvimento. Isso pode ser observado inclusive nos modelos de maturidade CMMI [Chrissis *et al*. 2006] e MPS.BR [Softex 2009] que introduzem esse requisito nos seus níveis iniciais, respectivamente, nível 2 e nível F. Além disso, o uso de estratégias de ramificação adequadas pode apoiar de sobremaneira a evolução de acordo com a necessidade de cada projeto [Walrad *and* Strom 2002]. Uma estratégia de ramificação bem implantada tem grandes chances de facilitar a manutenção do projeto e consequentemente contribuir para o aumento da qualidade do produto.

Como pode ser observada, a GCS é elemento chave para permitir a evolução

segura de software. Contudo, diversas abordagens com o propósito de utilizar refatorações e métricas para gerar melhoria na qualidade do software não empregam recursos da GCS. Por exemplo, O'Keeffe e Ó Cinnéide [2004] apresentam uma abordagem de refatoração automática de software. Para avaliar a qualidade das refatorações aplicadas, os autores utilizam algumas heurísticas existentes na literatura. Essas heurísticas baseiam-se em maximizar ou minimizar métricas. A fim de evitar conflitos de interesses entre as heurísticas, os autores listam uma ordem de prioridades entre as mesmas.

Em um trabalho posterior, O'Keeffe e Ó Cinnéide [2008] se basearam nas funções de avaliações de QMOOD [Bansiya *and* Davis 2002] que proveem um conjunto de métricas e atributos de qualidade de software, como, por exemplo, flexibilidade, reusabilidade e legibilidade. Para fazer as buscas em direção do que deve ser refatorado, os autores utilizam quatro técnicas: uma que para de aplicar transformações ao encontrar a primeira melhoria em relação ao código fonte original; uma que encontra a melhor de todas as transformações; uma que é a mescla entre a primeira e a segunda; e uma heurística.

Seng *et al.* [2006] também apresentam uma abordagem para refatoração automática. A refatoração "Mover Métodos" foi utilizada e aplicada em um projeto Open Source. Para efetuar as transformações, eles utilizaram uma estrutura que combina refatorações (genótipo) e modelo de dados (fenótipo). Com essa estrutura, eles aplicam o genótipo de um fonte (estrutura genótipo + fenótipo) a outros dois e geraram extensivamente as permutações possíveis. A fim de analisar os resultados, os autores utilizam uma função de avaliação. Ao final tem-se uma lista de possíveis refatorações, o que faz o processo de escolha e refatoração não ser totalmente automático. Os autores também se preocuparam em poder voltar ao estado original do projeto dado os passos para gerar as transformações.

Além de não empregarem GCS, essas abordagens não foram desenvolvidas para utilizar máquinas ociosas. Ou seja, apesar de tratarem de assuntos de elevada complexidade, a resolução do problema não ocorre de uma maneira mais proveitosa, otimizando recursos e evitando paralelismo com outras atividades de desenvolvimento. Finalmente, os trabalhos relacionados não apresentaram testes robustos em diversas aplicações reais.

3. Resultados esperados

Dado o problema apresentado na Seção 1, é proposto o desenvolvimento de um sistema multiagentes[1] que faça o monitoramento contínuo, por meio de métricas, de alguns atributos de qualidade de software [Bansiya *and* Davis 2002] e que execute refatorações automaticamente a fim de manter qualidade suficiente para amenizar a degradação do software em construção. Para que essas refatorações não interfiram no desenvolvimento e para oferecer um grande poder de processamento ao sistema multiagentes, sua execução será em momentos de ociosidade do parque computacional,

[1] Um sistema multiagentes é composto por um conjunto de agentes de software que são entidades autônomas (capazes de executar sem a necessidade da intervenção humana ou de outros agentes), adaptativas (podem adaptar seu comportamento às mudanças no ambiente no qual estão inseridos), interativas e orientadas a objetivos [Wooldridge *and* Ciancarini 2001]

de acordo com objetivos previamente estipulados por gerentes de desenvolvimento. Essa ação também evita o surgimento de conflitos entre as alterações dos agentes e dos desenvolvedores caso fossem feitas em paralelo.

Os três primeiros passos da Figura 1 indicam que a qualquer momento no desenvolvimento de software um gerente, em uma máquina de desenvolvimento, poderá: efetuar *login* no servidor (passo 1); solicitar a instalação do agente localmente (passo 2); e então realizar um agendamento de um determinado período do dia para disponibilizar essa máquina para o uso do sistema multiagentes (passo 3). Os dados de inicialização necessários são: os atributos de qualidade que se deseja melhorar, a URL do projeto a ser refatorado e o exato intervalo de tempo que os agentes deverão estar utilizando a máquina para realizarem seu trabalho. No Servidor é feita a configuração de quais atributos de qualidade devem ser melhorados e quais projetos estarão passíveis dessas melhorias. O sistema utiliza métricas associadas a atributos de qualidade apresentadas por Bansiya e Davis [2002] para avaliar eventuais refatorações. É responsabilidade do servidor controlar as tarefas dos agentes nas máquinas disponíveis.

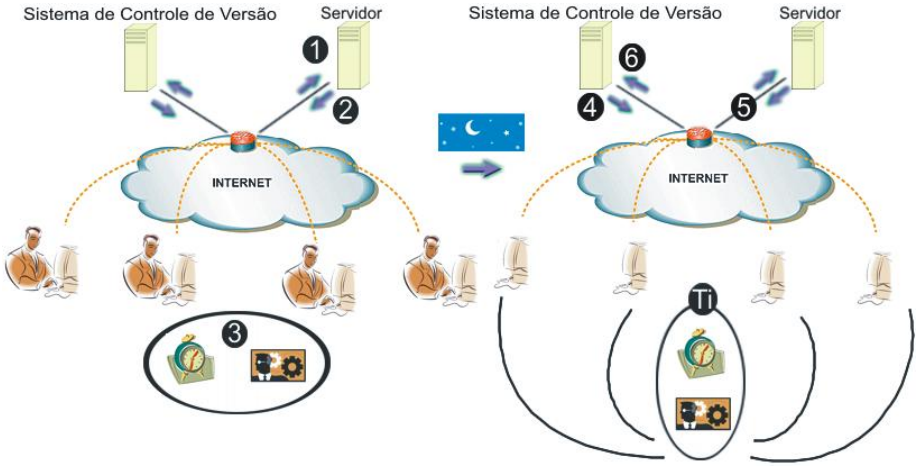

Figura 1: Criação e início do trabalho dos Agentes

Ainda na Figura 1, no momento "Ti" (tempo inicial de trabalho) o agente executa o passo 4 para obter a "Configuração Atual" através do *checkout* do projeto utilizando a URL informada pelo servidor, dando início ao diagrama apresentado na Figura 2. O passo 5 simboliza a comunicação com o servidor, onde os agentes requisitam tarefas a serem feitas (refatorações e métricas a serem utilizadas) e informam sucesso ou fracasso na execução de suas tarefas.

O diagrama da Figura 2 representa os agentes, a partir de uma configuração inicial, executando o seguinte procedimento: (1) **aplicação de refatorações automáticas**, onde os agentes são capazes de saber, por meio dos atributos de qualidade objetivados pelo gerente, quais algoritmos de refatorações e métricas devem ser utilizados. Com a execução das refatorações os agentes geram as "Configurações Derivadas"; e (2) **avaliação das configurações derivadas**, onde os agentes aplicam métricas pré-determinadas sobre as configurações derivadas e avaliam qual configuração melhorou em relação à configuração atual. A configuração eleita passa a ser a configuração atual e o procedimento se repete.

Ocorrendo o momento "Tf" (Tempo final), em função da configuração do sistema multiagentes, o agente gera relatórios ou faz o *checkin* da configuração atual em um ramo isolado do repositório (passo 6 - Figura 1) e termina sua execução.

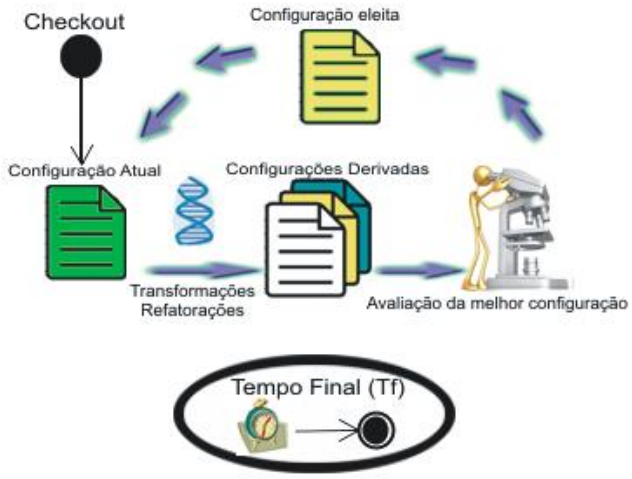

Figura 2: Trabalho dos Agentes

O sistema proposto será avaliado em um ambiente real de desenvolvimento, no contexto do sistema de gestão acadêmica da Universidade Federal Fluminense, que tem em torno de 30 desenvolvedores e 300 mil linhas de código. Os resultados obtidos serão apresentados a essa equipe de desenvolvimento para obter informações sobre a significância das melhorias obtidas.

4. Metodologia e estado atual do trabalho

Para o propósito desde trabalho, um estudo direcionado a determinar em que ponto se encontra o estado da arte vem sendo realizado para as seguintes pesquisas: (1) refatorações automáticas; (2) aplicações de métricas; (3) soluções distribuídas; e (4) problemas de busca conhecidos como *Search-Based Software Engineering*.

Para consolidar as ideias do trabalho, são realizados periodicamente seminários em que o trabalho é apresentado para o grupo de alunos de mestrado e doutorado, além dos orientadores, de forma que todos possam criticar e contribuir com sugestões para o trabalho. A própria participação no WTDQS é de grande importância para a consolidação das ideias do trabalho e para avaliar a aceitação deste pela comunidade acadêmica, constituindo uma base mais sólida para a continuidade do seu desenvolvimento.

O estado atual do trabalho encontra-se na adequação de métricas e refatorações automáticas identificadas na literatura para o sistema proposto, para isso, estão sendo estudadas refatorações automatizáveis [O'Keeffe *and* Ó Cinnéide 2008][Seng *et al.* 2006] e métricas para atributos de qualidade [Bansiya *and* Davis 2002]. A Infraestrutura de comunicação, instalação e atualização do sistema multiagentes encontra-se em fase final de implementação, já a criação dos agentes com seus respectivos papéis, responsabilidades e ações encontra-se em processo inicial de desenvolvimento.

5. Agradecimentos

Agradecemos ao CNPQ e a CAPES pelo apoio financeiro.

Referências

Bansiya, J. and Davis, C., (2002), "A Hierarchical Model for Object-Oriented Design Quality Assessment", *IEEE Transactions on Software Engineering*, v. 28, n. 1, p. 4-17.

Brooks, J., (1987), "No Silver Bullet Essence and Accidents of Software Engineering", *Computer*, v. 20, n. 4, p. 10-19.

Chrissis, M. B. and Konrad, M. and Shrum, S., (2006), *CMMI(R): Guidelines for Process Integration and Product Improvement*. 2 ed. Addison-Wesley Professional.

Dart, S., (1991), "Concepts in configuration management systems". In: *Proceedings of the 3rd international workshop on Software configuration management*, p. 1-18, Trondheim, Norway.

IBM, (2010), World Community Grid. Disponível em: <"http://www.worldcommunitygrid.org">. Acesso em: 9 Abr 2010.

Leon, A., (2000), *A Guide to Software Configuration Management*. Artech House Publishers.

Murta, L. G. and Werner, C. M., (2006), *Gerência de Configuração no Desenvolvimento Baseado em Componentes*. Tese de Doutorado, UNIVERSIDADE FEDERAL DO RIO DE JANEIRO - UFRJ

O'Keeffe, M. and Ó Cinnéide, M. O., (2004), "Towards automated design improvement through combinatorial optimisation". In: *Proceedings of the Workshop on Directions in Software Engineering Environments*

O'Keeffe, M. and Ó Cinnéide, M., (2008), "Search-based refactoring for software maintenance", *Journal of Systems and Software*, v. 81, n. 4 (Abr.), p. 502-516.

Parnas, D. L., (1994), "Software aging". In: *Proceedings of the 16th international conference on Software engineering*, p. 279-287, Sorrento, Italy.

Seng, O. and Stammel, J. and Burkhart, D., (2006), "Search-based determination of refactorings for improving the class structure of object-oriented systems". In: *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, p. 1909-1916, Seattle, Washington, USA.

Softex, (2009), Mps.Br - Melhoria de Processos do Software Brasileiro. Disponível em: <"http://www.softex.br/mpsbr/_home/default.asp">. Acesso em: 20 Maio 2010.

Walrad, C. and Strom, D., (2002), "The importance of branching models in SCM", *Computer*, p. 31–38.

Wooldridge, M. and Ciancarini, P., (2001), "Agent-oriented software engineering: the state of the art". In: *First international workshop, AOSE 2000 on Agent-oriented software engineering*, p. 1-28, Limerick, Ireland.