A Ferramenta de Análise Estática Klocwork Integrada a um Processo Formal de Revisão de Código, nível 3 do CMMI

Denise Piubeli Prado[1], Aletéia Xavier Bettin[1],

Carlos Miguel Tobar[2], Vinicius Asta Pagano[1]

[1]Instituto de Pesquisas Eldorado – IPE – Departamento de Software Aplicado –
Avenida Érico Veríssimo s/no – Campus Unicamp – Cidade Universitária
Campinas – SP – Brasil

[2]Pontifícia Universidade Católica de Campinas – PUC-Campinas
Caixa Postal 317 – 13.086-900– Campinas – SP – Brazil

{denise.prado, aleteia.bettin, vinicius.pagano}@eldorado.org.br,
tobar@puc-campinas.edu.br

Abstract. *This paper describes how the Klocwork tool for static analysis was integrated in the code review process adopted by a CMMI level 3 software development organization. Benefits and quality improvements are presented, such as: (1) improved quality of code reviews, as evidenced by the discovery of a greater number of software defects; (2) reduction of the amount of time required for code inspections. Time savings can either lead to a decrease in overall software development costs, or it can be redirected to other stages or processes within the software engineering cycle, for instance in the development of new functionalities that represent a competitive differential for the resulting software product.*

Resumo. *Este artigo apresenta como a ferramenta Klocwork de análise estática foi integrada a um processo de revisão de código adotado por uma organização desenvolvedora de software nível 3 de CMMI. São apresentados os benefícios e os ganhos de qualidade obtidos com essa integração, tais como (1) a melhoria da qualidade da revisão, envolvendo a descoberta de um maior número de defeitos de software, e (2) a redução do tempo utilizado em inspeções de código. O tempo economizado contribui para a redução de custo de desenvolvimento, ou pode ser aplicado em outras fases ou processos da engenharia de software, como no desenvolvimento de novas funcionalidades que representam um diferencial competitivo para o produto desenvolvido.*

1. Introdução

O mercado de desenvolvimento de *software* está cada vez mais globalizado e competitivo. Rapidez na criação de produtos, implementação de funcionalidades diferenciadas e baixos custos são premissas para se conquistar cada vez mais clientes. Motivadas por essas premissas, empresas que desenvolvem *software* procuram aperfeiçoar e melhorar a qualidade de seus processos de engenharia, investindo em integrações entre processos e ferramentas.

Além disso, os sistemas de *software* estão se tornando cada vez mais complexos e isso demanda atender, além dos requisitos funcionais, requisitos de segurança, de *performance* e, em geral, de distribuição em redes de dados. É fundamental que os engenheiros tenham conhecimento e tempo para analisarem e implementarem esses requisitos adequadamente. Essas duas variáveis podem ser satisfeitas a partir dos resultados da integração entre ferramentas e processos . Um outro aspecto, que também estimula essa integração, é a variedade de tecnologias e de linguagens de programação que podem ser utilizadas no desenvolvimento. Ferramentas de análise estática de código, por exemplo, suportam diferentes linguagens de programação e atuam em diversas categorias de defeitos, como gerenciamento de memória, gerenciamento de dados de programa, violações de limites de variáveis, etc. e podem realizar revisões de código que um engenheiro menos experiente em determinada linguagem realizaria.

Se realizada adequadamente, a integração de uma ferramenta de análise estática e um processo de revisão de código pode afetar positivamente variáveis de tempo para desenvolvimento e inspeções; custos de projetos; e qualidade de produtos.

Tal integração pode promover uma melhoria da qualidade da revisão em si, envolvendo a descoberta de um maior número de defeitos. Além disso, pode reduzir o tempo necessário para a realização de inspeções. Essa economia de tempo contribui para a redução de custos de um projeto e pode ser utilizada para encontrar outros defeitos associados com a *performance* ou robustez, por exemplo. Pode-se também utilizar esse tempo em outras fases da engenharia de *software*, como validação e testes, ou mesmo investi-lo na implementação de novas funcionalidades que tragam um diferencial competitivo para o produto.

Considerando: (a) as premissas para atuar em um mercado global cada vez mais competitivo, (b) as questões de complexidade e de segurança de requisitos de sistemas, (c) a variedade de tecnologias e de linguagens de programação, e (d) os potenciais benefícios de uma integração entre uma ferramenta de análise estática e o processo de revisão, o Instituto de Pesquisas Eldorado [IPE 1999] aperfeiçoou seu processo de revisão de código, promovendo sua integração com a ferramenta de análise estática *Klocwork* K7 [Klocwork 1996]. Esse aperfeiçoamento também foi estimulado pela obtenção de resultados positivos em um projeto interno de desenvolvimento de *software*. Este artigo tem como objetivo apresentar os resultados dessa integração.

A seção seguinte apresenta brevemente o processo de revisão de código em questão. A seção três mostra o conceito da análise estática de código. A seção quatro é usada para apresentar a integração propriamente dita. Por fim, apresenta-se a conclusão a respeito dos resultados da integração realizada.

2. Processo de Revisão de Código

Um processo de revisão de código consiste na definição e execução de etapas que verifiquem um produto de *software*. Tais etapas podem ser conduzidas por uma ou mais pessoas com similar conhecimento do produto analisado, com o intuito de assegurar que o mesmo está sendo construído de acordo com especificações e requisitos previamente definidos e acordados. O intuito, ao longo do ciclo de vida, é antecipar a identificação de possíveis defeitos, a tempo de serem corrigidos, evitando tardia detecção e custosos retrabalhos [Chess e West 2007].

Existem diversos métodos e técnicas para a realização de uma revisão [Barreto e Rocha 2003]. No entanto, considerando o modelo de maturidade CMMI [SEI 2009] no seu nível três, um processo de revisão foca três principais etapas: (1) preparar a revisão; (2) realizar revisões por pares (*peer reviews*); e (3) verificar os produtos selecionados. A primeira etapa resume-se a: selecionar os artefatos que serão revisados e os métodos que serão adotados; estabelecer o ambiente; e determinar os procedimentos e critérios para a revisão. A segunda etapa consiste na real preparação: da revisão por pares, da sua condução e da posterior análise dos dados. A terceira etapa é constituída pela realização efetiva da revisão por pares, com a análise dos resultados obtidos e registro de ações corretivas ou possíveis melhorias identificadas [Chrissis et al. 2007].

Por meio da experiência vivenciada, pôde-se constatar que desenvolver *software,* adotando uma prática de revisão de fato, eleva o número de defeitos detectados nas fases iniciais do ciclo de vida, o que auxilia o cumprimento de custo e prazo acordados com o cliente, assim como a aderência aos requisitos definidos e conseqüente satisfação com o produto entregue. Além disso, revisões constantes geram uma padronização do código, facilitando uma posterior manutenção. Outro fator positivo diz respeito à comunicação realizada entre os integrantes do time técnico ao longo das revisões, quando ocorre uma troca de conhecimentos e um amadurecimento da equipe como um todo.

Apesar das vantagens mencionadas, existem alguns pontos negativos identificados, que geram fortes questionamentos sobre os investimentos requeridos. Entre estes, pode-se citar o esforço gasto para a execução das etapas de revisão, que segundo Bathi e Kepler (2005), podem chegar a 80% do desenvolvimento de um produto. Também, pode-se citar a complexa questão da alocação de recursos, que nem sempre estão disponíveis. Quando isto acontece, podem-se identificar muitos casos em que um recurso, com alto conhecimento técnico, é alocado para a realização da revisão de um código de baixa complexidade ou que tenha sido desenvolvido por um programador com pouca experiência. Fato que, ao longo do tempo, além de elevar o custo da implementação daquele código, pode provocar desmotivação dos colaboradores com maior senioridade, e, ainda mais crítico do que isto, sem a garantia de que todos os defeitos serão mapeados nesta fase, devido ao desgaste na execução da mesma.

3. Análise Estática de Código

A análise estática de código é um método que visa revisar um código fonte e buscar por vulnerabilidades e potenciais defeitos, garantindo que os desenvolvedores programem de forma correta e segura. Ferramentas que empregam a análise estática, durante o ciclo de desenvolvimento, atingem maior ganho quando executadas nas fases iniciais de desenvolvimento, pois aí há maior injeção de erros e o custo para correção é baixo. Essa estratégia promove diretamente a redução dos custos de correção e, como conseqüência eleva as taxas de ROI (*Return of Investment*) das empresas que as utilizam, pois, segundo Gordon (2006), um defeito, encontrado em fases de testes finais, ou mesmo após a entrega do produto, é entre 50 a 1000 vezes mais caro de ser corrigido do que se fosse detectado em fases iniciais.

A motivação para o uso de ferramentas de análise estática, também reside nos fatos: i) das aplicações serem construídas de forma menos isoladas, ou seja, a forte integração entre elas exige que mais testes de regressão sejam executados; ii) das

aplicações possuírem requisitos mais complexos, como por exemplo a demanda da implementação com tecnologias *Web* e que atendam a demanda de 24x7, diminuindo assim a tolerância à ocorrência de falhas; e principalmente iii) da diversidade das aplicações e plataformas que se fazem necessárias para atender a todos os requisitos solicitados [Murphy 2008].

As ferramentas de análise estática possuem a vantagem de serem utilizadas sem que o código necessite ser executado, além de analisarem diversas classes de defeitos, como por exemplo: gerenciamento de memória; variáveis não inicializadas; estouro de *buffers;* validação de parâmetros; formatação de cadeia e caracteres; acesso à memória com ponteiro previamente liberado; retorno de funções em variáveis locais; e dados externos não validados, entre outras. Segundo Teixeira et al. (2007), algumas classes , como variáveis não inicializadas e conversões de tipos, caracterizam vulnerabilidades de código, ou seja, mapeiam os defeitos diretamente relacionados à segurança dos dados e do código desenvolvido.

Dentre as diversas ferramentas disponíveis no mercado [NIST 2009], duas opções são consideradas mais completas e se destacam na área de segurança de código, *design* de aplicações, gerenciamento de risco de potenciais defeitos e mapeamento da evolução do *software* em desenvolvimento. São elas a Klocwork K7 e a Coverity Prevent [Coverity 2000]. Consideradas líderes de mercado [Emanuelsson e Nilsson 2008], em avaliação realizada pela conceituada publicação InfoWorld [Binstock 2006], a ferramenta K7 foi considerada superior. Devido ao seu histórico e também ao fato do Instituto de Pesquisas Eldorado já possuir uma experiência de sucesso de uso da ferramenta em um projeto de um parceiro [Reis et al. 2008], a ferramenta K7 foi selecionada para fazer parte de um projeto interno de manutenção, denominado projeto piloto, com os objetivos de: melhorar a agilidade no processo de revisão de código, reduzir o esforço em inspeções e promover um desenvolvimento com qualidade.

4. Integração entre Klocwork e o Processo de Revisão de Código

No Instituto de Pesquisas Eldorado existe uma equipe de sete engenheiros, com alto conhecimento sobre ferramentas de análise estática, especialmente a desenvolvida pela empresa Klocwork. O grupo provê suporte e consultoria a clientes internos e externos, interessados em utilizar este tipo de tecnologia.

Quando foi tomada a decisão de aplicar a ferramenta K7 no projeto piloto, um engenheiro júnior dessa equipe de especialistas dedicou 30 horas para instalar e configurar a ferramenta no ambiente de desenvolvimento dos futuros usuários, um time de 20 indivíduos. Além disso, duas horas do especialista foram utilizadas para treinar o time envolvido. O treinamento foi detalhado, com atividades práticas, simulações e análises de defeitos comumente apontados pela ferramenta.

A divisão de atividades no projeto piloto foi realizada através de registros de solicitações de mudança, conforme padronizado na organização, sendo que o planejamento de horas para desenvolvimento de cada registro é realizado seguindo um processo único para estimativa da demanda de esforço. Para toda solicitação de mudança é aplicada ao código que será modificado a técnica da complexidade ciclomática de McCabe [McCabe e Watson 1994], para auxiliar a distribuição de atividades entre os desenvolvedores, considerando sua senioridade.

Alocado um recurso para atuar em uma solicitação de mudança, este deverá consultar a especificação de requisitos e o *design* do *software* para compreender o quê deverá ser desenvolvido e/ou alterado. O desenvolvedor deve realizar a implementação do código, seguindo o guia de codificação da linguagem utilizada – no caso do projeto piloto, *Java*. Para garantir a aderência aos requisitos e a inexistência de erros pontuais, testes unitários devem ser executados ao final da codificação. O autor do código, ao considerá-lo concluído, deve convocar um moderador e inspetores para a sessão de revisão, disponibilizando o trecho implementado e garantindo que os envolvidos tenham tempo hábil em se preparar para esta reunião. A partir deste momento, a sessão acontece, seguindo as etapas mencionadas anteriormente, e dela participa o autor do código, a fim de esclarecer possíveis dúvidas e discutir eventuais defeitos identificados.

Adotando a prática de utilizar a ferramenta K7, o processo de revisão descrito mantém-se praticamente o mesmo, com duas exceções. A primeira acontece ao longo do processo de codificação, quando o desenvolvedor aciona o *plug in* do K7, previamente instalado na máquina, quando assim achar necessário e corrige os eventuais defeitos identificados pela ferramenta. A Figura 1 ilustra resultados gerados pela K7.

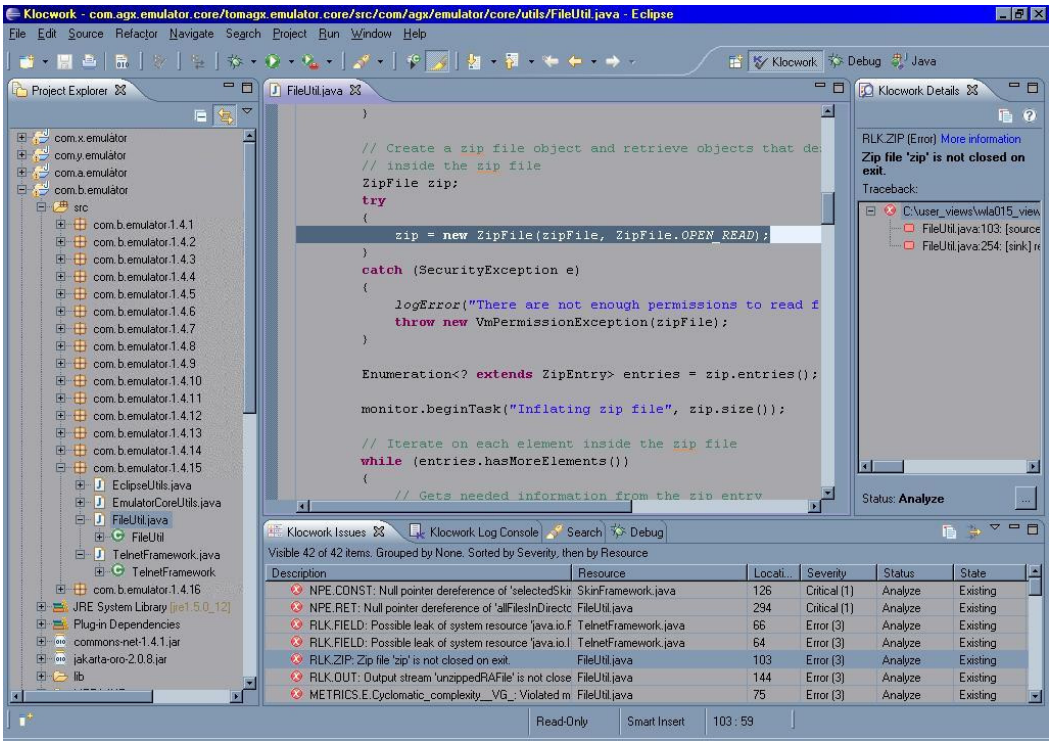

Figura 1. Exemplo de defeitos identificados pela ferramenta Klocwork

Na segunda exceção, os relatórios de defeitos e de complexidade emitidos pela ferramenta K7 passam a ser entradas do processo de revisão, sendo disponibilizados aos participantes antes da sessão de revisão, junto ao código que será inspecionado.

Os relatórios da ferramenta K7, na revisão, têm o intuito de exibir de antemão os defeitos apontados e já corrigidos pelo autor do código. Os defeitos encontrados são defeitos que normalmente acontecem durante o desenvolvimento de código, como por exemplo, uma variável que não tenha sido inicializada, o acesso a um ponteiro inválido, ou até mesmo o descumprimento de uma norma da organização. Com isso, se os

defeitos apontados foram todos corrigidos, o tempo dedicado para a reunião fica voltado principalmente para os defeitos considerados não triviais, os quais exigem maior experiência de inspetores, como por exemplo aqueles que decorrentes de arquitetura ou *design* pobres, ou ainda os que comprometem a *performance* ou a robustez do produto.

De forma a comprovar os benefícios ganhos com a integração realizada, alguns dados foram levantados para comparação de dois cenários no projeto piloto: (cenário 1) sem a utilização da ferramenta Klocwork e (cenário 2) com a sua utilização. Foram selecionados dois pares de solicitações de mudança. Cada par de solicitação envolveu: a mesma complexidade McCabe, a mesma senioridade do desenvolvedor, o mesmo número de linhas de código, semelhante escopo para a inspeção e semelhante senioridade dos indivíduos participantes das reuniões de revisão. O intuito, com isso, foi nivelar ao máximo as variáveis consideradas no processo de revisão. A Tabela 1 apresenta os dados e os resultados obtidos.

Tabela 1. Comparativo de resultados do projeto piloto, com e sem o uso da ferramenta Klocwork.

Variáveis		Solicitação de Mudança 1	Solicitação de Mudança 2	Solicitação de Mudança 3	Solicitação de Mudança 4
Utilização da ferramenta Klocwork K7		Não	Sim	Não	Sim
Senioridade do desenvolvedor		Júnior	Júnior	Pleno	Pleno
Esforço total para desenvolvimento do código (h)		8 h	8 h	16 h	16 h
Número total de inspetores envolvidos – além do autor		1	1	3	2
Senioridade dos inspetores envolvidos		Júnior	Júnior	1 Sênior e 2 Plenos	1 Sênior e 1 Pleno
Número de linhas de código a serem inspecionadas		250	250	250	250
Complexidade McCabe do código		Baixa	Baixa	Média/Alta	Média/Alta
Tempo de preparação para a inspeção (h e/ou min)		1:12 h:min	30 min	2 h	1:10 h:min
Tempo da execução da inspeção (h ou min)		1 h	30 min	2 h	1 h
Tempo de retrabalho do código, após a inspeção (min)		10 min	0 min	1 h	30 min
Número total de defeitos encontrados no código	Via Inspeção	2	0	8	3
	Via Klocwork	-	5	-	10
Número total de defeitos triviais encontrados no código		2	5	6	10
Número total de defeitos não triviais encontrados no código		0	0	2	3
Esforço Total demandado para execução de todo o processo de revisão de código (tempo total acumulado)		3:12 h:min	1:30 h:min	15 h	5:50 h:min

Comparando-se os seguintes itens entre as solicitações de mudança 1 e 2: (a) tempo de preparação para a sessão de revisão do código; (b) tempo da execução da inspeção e (c) tempo de retrabalho dos defeitos identificados, pode-se observar uma considerável melhora no tempo total gasto para essas atividades.

Além disso, a quantidade de defeitos identificados pela ferramenta foi maior que o número identificado durante a reunião, considerando erros de mesmo nível. Fato que pode ocorrer, visto que, no caso da solicitação 1, há a necessidade de se considerar o conhecimento e experiência de desenvolvimento do inspetor (júnior) envolvido.

Os resultados observados na comparação das solicitações de mudança 3 e 4 são ainda mais relevantes, visto que a redução no esforço total demandado para execução de todo o processo de revisão de código, levando em conta o número de inspetores envolvidos e a participação do autor, foi de aproximadamente 62%. Além da diferença apresentada na quantidade de defeitos mapeados pela ferramenta – no caso da solicitação de mudança 4 – possibilitando que os esforços dos inspetores ficassem voltados para os problemas mais críticos a serem analisados.

Os resultados mostrados até aqui comprovam que há uma potencial economia de tempo de preparação e inspeção de código, tanto para solicitações de mudanças de baixa quanto de alta complexidade, conseqüentemente, gerando maior rapidez na execução do processo e permitindo que os inspetores foquem mais profundamente problemas não triviais, gerando um código com mais qualidade

Mais estudos estão sendo feitos com a integração no projeto piloto, de forma a coletar dados que comprovem seus benefícios diante dos outros motivadores, também já citados, como segurança, variedade de tecnologias, plataformas e linguagens, e utilização do tempo economizado em outras fases da engenharia de *software*. Além disso, estudos que mostrem quantitativamente o valor de ROI também estão caminhando em paralelo e serão oportunamente publicados.

5. Conclusão

Este artigo apresenta resultados parciais de um projeto piloto onde se integrou a ferramenta de análise estática K7 da Klocwork em um processo de revisão, aderente ao nível 3 do CMMI.

Os benefícios e a qualidade do *software* desenvolvido por uma organização que adota uma política voltada para a utilização de processos, seguindo práticas e estando aderente ao nível três de maturidade no modelo CMMI, são indiscutíveis. No entanto, considerar as necessidades impostas por um mercado competitivo e global quanto à agilidade na produção, mantendo alto índice de qualidade, a um custo razoável, são fatores que exigem grande experiência e habilidade para que uma organização mantenha-se competitiva.

A integração entre a ferramenta K7 da Klocwork e o processo de revisão no estudo de caso apresentado neste artigo aponta para vários benefícios, como análises mais precisas e corretas dos defeitos encontrados no código sob revisão, redução do esforço gasto nas reuniões de revisão, aumento no número de defeitos encontrados durante as fases iniciais de desenvolvimento, e não após entrega do produto, e conseqüentemente, a redução do custo final do *software* produzido. Estudos estão em andamento para complementar os resultados até aqui obtidos.

Integrar a um processo, bem definido e maduro, técnicas e ferramentas que permitam elevar a competitividade está se tornando fundamental para manter o destaque do Instituto de Pesquisas Eldorado no mercado global.

Referências

Barreto, A. O. S.; Rocha, A. R. C. (2003) "Apoio ao Processo de Verificação em Ambientes de Desenvolvimento de Software Orientados a Organização", Rio de Janeiro – Brasil: COPPE/Universidade Federal do Rio de Janeiro.

Bhati, S.N., Kepler, J. (2005) "*Why Quality? ISO 9126 Software Quality Metrics (Functionality) Support by UML*", *ACM Software Engineering Notes*, v. 30, n. 2.

Binstock, A. (2006) "*Coverity and Klocwork code analyzers drill deeper*", http://www.infoworld.com/article/06/01/26/73919_05FEcode_3.html, Janeiro.

Chess, B.; West, J. (2007) "*Secure Programming with Static Analysis.*" Boston: Addison-Wesley.

Chrissis, M. B.; Konrad, M.; Shrum, S. (2007) "*CMMI: Guidelines for Process Integration and Product Improvement – Second Edition*", N.York: Addison–Wesley.

Coverity, Inc. (2000) http://www.coverity.com/

Emanuelsson, P.; Nilsson, U. (2008) "*A Comparative Study of Industrial Static Analysis Tools (Extended Version)*"; *Technical reports in Computer and Information Science*, *LinkÄoping Electronic Press*: http://www.ep.liu.se/ea/trcis/. Institute of Technology – Linköping University, Linköping, Sweden.

IPE, Instituto de Pesquisas Eldorado (1999) http://www.eldorado.org.br

Gordon, I. (2006), "*Automated Source Code Analysis: Reduce Customer and QA Defects to Save Time and Money!*", http://www.nohau.se/images/pdf/Test-roadshow-nohau-klocwork.pdf, Setembro.

Klocwork, Inc. (1996) http://www.klocwork.com.

McCabe, T. J. and Watson, A. H. (1994) "Software Complexity." *Crosstalk, Journal of Defense Software Engineering*. n. 7, p. 5-9.

Murphy, T. E. (2008) "*Key Issues for Software Quality and Testing, 2008"*, http://www.gartner.com, Agosto.

NIST, National Institute of Standards and Technology (2009) "*Source Code Security Analyzers*", https://samate.nist.gov/index.php/Source_Code_Analyzers, Janeiro.

Reis C.; Prado, D.; Fernandes, M.G. (2008), "Uso da ferramenta de Análise Estática Klocwork na Motorola", http://www.incremental.com.br/sbqs2008/ ?show=Index&itemmenu=11

Teixeira, E.; Antunes, J.; Neves, N. (2007) "Avaliação de Ferramentas de Análise Estática de Código para Detecção de Vulnerabilidades", disponível em: http://www.navigators.di.fc.ul.pt/archive/papers/teixeira07.pdf.

SEI, *Software Engineering Institute* (2009) "*CMMI for Development, Version 1.2*", http://www.sei.cmu.edu/cmmi/models/, Janeiro.