Aplicando Técnicas de Visualização de Software para Apoiar a Escolha de Políticas de Controle de Concorrência

João Gustavo Gomes Prudêncio

Orientadores:
Cláudia Maria Lima Werner
Leonardo Gresta Paulino Murta

Programa de Engenharia de Sistemas e Computação (PESC) – COPPE –
Universidade Federal do Rio de Janeiro (UFRJ)
Caixa Postal 68.511 – 21.945-970 – Rio de Janeiro – RJ – Brasil

{gustavo,werner,murta}@cos.ufrj.br

Nível: Mestrado
Ano de Ingresso: 2006
Previsão de Conclusão: Março de 2008
Aprovação da Proposta: Abril de 2007

Resumo. *O desenvolvimento de software tem se tornado uma atividade cada vez mais complexa, devido a requisitos cada vez mais difíceis de serem alcançados e ao aumento da equipe de desenvolvimento. A Gerência de Configuração de Software (GCS) representa uma disciplina importante para o controle da evolução de software e possibilita uma maior cooperação entre as pessoas envolvidas, possibilitando o trabalho concorrente. Este trabalho tem como objetivo desenvolver uma abordagem para visualização de dados gerados pela GCS utilizando técnicas de visualização de software, para fornecer aos usuários envolvidos no processo de desenvolvimento, informações importantes para tomadas de decisão relacionadas ao controle de concorrência.*

Palavras-chave: *Gerência de Configuração de Software, Visualização de Software, Controle de Concorrência.*

1. Introdução

Na engenharia de software, a redução do ciclo de vida do produto, aliado com o aumento da complexidade das soluções e da equipe de desenvolvimento, resultou em uma maior pressão por trabalho concorrente [ESTUBLIER 2000]. Dessa forma, há a necessidade de definir estratégias de cooperação para o desenvolvimento de sistemas, possibilitando que mais de um desenvolvedor possa modificar e gerenciar o mesmo conjunto de artefatos.

A Gerência de Configuração de Software (GCS) é uma disciplina que possibilita a evolução de sistemas de software de forma controlada e, portanto, contribui no atendimento a restrições de qualidade e de tempo [ESTUBLIER 2000]. Os sistemas de GCS são fundamentais para prover controle sobre os artefatos produzidos e modificados por diferentes desenvolvedores, dando suporte ao trabalho concorrente. Além disso, esses sistemas possibilitam um acompanhamento minucioso do andamento das tarefas de desenvolvimento [MURTA 2006].

As soluções existentes de GCS provêem uma infra-estrutura para a definição de quais artefatos serão passíveis de GCS, os chamados itens de configuração (ICs), e permitem que estes artefatos sejam obtidos, através de um processo conhecido como *check-out*. Com isso, o engenheiro de software faz uma cópia desses artefatos para o seu espaço de trabalho, local destinado à criação e alteração dos ICs. Depois de efetuar mudanças, o engenheiro retorna os artefatos para o repositório, através de um processo conhecido como *check-in*. Quando um engenheiro de software tenta fazer um *check-in* e o artefato em questão sofreu modificações de outros desenvolvedores desde o seu último *check-out*, pode ocorrer uma situação de conflito.

Leon [2000] apresenta a evolução das técnicas de GCS, onde, inicialmente, os artefatos de software eram compartilhados por diversos desenvolvedores, porém modificações efetuadas por um desenvolvedor geravam efeitos colaterais nos artefatos de outros desenvolvedores. Uma solução inicial foi a criação de várias cópias do mesmo artefato compartilhado, entretanto faltava controle sobre as cópias existentes e era necessário retrabalho nas diferentes cópias para implementar os mesmos requisitos e corrigir os mesmos defeitos. Adotou-se, então, a utilização de repositórios centralizados que armazenavam versões comuns dos artefatos, no entanto, era necessária a criação de mecanismos de controle sobre o repositório centralizado. Esses mecanismos são implementados por meio da utilização de políticas de controle de concorrência pelos sistemas de controle de versões. Dentre essas políticas, podemos citar as políticas pessimista e otimista.

A política pessimista enfatiza o uso de *check-out* reservado, fazendo um bloqueio (*lock*) e inibindo o paralelismo do desenvolvimento sobre o mesmo artefato. Utilizando essa política, as situações de conflito são evitadas. Por outro lado, a política otimista assume que a quantidade de conflitos é naturalmente baixa e que é mais fácil tratar cada conflito individualmente, caso eles venham a ocorrer [MURTA 2006]. O mecanismo de tratamento de conflitos utilizado pela política otimista é a junção (*merge*) das duas versões do IC.

Existem situações onde uma determinada política é mais indicada do que a outra. Nos casos onde a junção dos trabalhos tende a ser complexa, quando, por

exemplo, os ICs não são textuais e a ferramenta que conhece a representação binária dos ICs não dá apoio automatizado para junções, é mais indicado trabalhar usando políticas pessimistas. Por outro lado, nos casos onde os ICs são textuais e a junção não é complexa, a utilização de políticas otimista é mais indicada.

Por outro lado, o estabelecimento de uma infra-estrutura de GCS faz com que uma grande quantidade de dados seja coletada e armazenada. Porém, os engenheiros de software, normalmente, não têm condições de explorar todo esse conteúdo, fazendo com que informações importantes não sejam identificadas. A análise dessas informações pode possibilitar aos engenheiros de software uma visão geral do sistema e fornecer estatísticas importantes dos componentes do sistema, apoiando tomadas de decisão. Essas informações podem ser melhor apresentadas utilizando técnicas de visualização de software [CEMIN 2001].

A visualização de software tem como objetivo facilitar o entendimento das informações relacionadas ao desenvolvimento de software [CEMIN 2001] e ainda atuar como uma forma de comunicação entre as pessoas que estão explorando ou manipulando a mesma informação [MACKINLAY and SHNEIDERMAN 2000]. Ela inclui técnicas de desenho, coloração, agrupamento de informações e animação.

Com a visualização, é possível explorar o potencial da cognição humana e habilidades de percepção que não são possíveis apenas com representações textuais [LINTERN et al. 2003]. Representações visuais tendem a proporcionar uma forma mais simples e intuitiva de entender melhor e mais rapidamente o significado dos dados representados. Essa necessidade se torna ainda mais importante à medida que aumenta o volume de informação.

Este trabalho tem como objetivo geral o criação de uma abordagem para apoiar tomadas de decisão relacionadas ao controle de concorrência, tendo em vista uma maior produtividade no processo de desenvolvimento de software. Esse objetivo é dividido em duas etapas: (i) identificar os ICs do sistema que merecem uma maior atenção em relação ao controle de concorrência e (ii) sugerir a política de controle de concorrência mais indicada para cada IC. As informações necessárias para atingir essas etapas serão geradas a partir da análise dos dados gerados pelos sistemas de GCS. Além disso, essas informações serão visualizadas pelos diferentes usuários envolvidos no processo de desenvolvimento por meio de técnicas de visualização de software.

O restante desse trabalho está organizado em outras três seções. A seção 2 apresenta a abordagem proposta, na seção 3 são apresentados alguns trabalhos relacionados e na seção 4 são discutidos os resultados esperados, o estado atual do trabalho e a avaliação dos resultados.

2. Caracterização da Contribuição

Como foi apresentado na seção 1, serão identificados os ICs do sistema que merecem uma maior atenção em relação ao controle de concorrência. Esses itens serão chamados de *elementos críticos* do sistema. A granularidade desses elementos deve ser definida pelo engenheiro, e pode representar, por exemplo, pacotes, classes ou métodos. Através dessa definição, é possível analisar o sistema em diferentes níveis de abstração.

Com a identificação dos elementos críticos, de certa forma, estão sendo identificados também alguns riscos no desenvolvimento do sistema. Alguns conhecimentos existentes na área de gerência de riscos são úteis para a análise desses riscos. A IEEE Std 1540 [IEEE 2001] define risco como a probabilidade de um evento, perigo, ameaça ou situação ocorrer associado às conseqüências indesejáveis, ou seja, um problema potencial. Nesse contexto, o risco tratado por esse trabalho é a perda de produtividade. De acordo com a gerência de riscos, os riscos que devem receber maior atenção em um projeto de software são os que têm maior probabilidade de acontecer, e que, ao acontecerem, que têm grande potencial de causar um maior impacto no projeto. Assim, é calculado o valor referente à probabilidade de um dado risco ocorrer e o valor do impacto desse risco. Esses valores são multiplicados e um terceiro valor, chamado de exposição, é obtido. A exposição indica o quão preocupante é o risco [SOFTEX 2006].

Utilizando os conhecimentos de gerência de risco, para cada IC do sistema é calculado um valor que representa o quão crítico é esse item. Esse valor leva em consideração duas métricas: o nível de **concorrência** de um IC (semelhante à probabilidade do risco, pois quanto maior for a concorrência, maior será a probabilidade da necessidade de junção) e o nível de **dificuldade de junção** do IC (semelhante ao impacto do risco, pois quanto mais difícil for a junção, maior será o impacto na produtividade da equipe).

A primeira delas informa se esse item é, em muitos casos, modificado por mais de um desenvolvedor ao mesmo tempo, levando em consideração o tempo que o elemento ficou em posse dos desenvolvedores (tempo entre o *check-out* e *check-in*) e o tempo que este ficou em posse de mais de um desenvolvedor. Com isso, é calculado um número, entre 0 e 1, que indica qual o nível de concorrência de cada IC analisado.

A segunda métrica quantifica a dificuldade de se fazer a junção de duas versões de um dado IC. Ou seja, considerando que houve uma situação de conflito, é quantificada, entre 0 e 1, a dificuldade de resolução desse conflito. Essa métrica leva em consideração a quantidade de sub-elementos que sofreram conflitos e a dificuldade de resolvê-los. Os sub-elementos representam elementos do item em questão. Por exemplo, métodos e atributos podem ser considerados sub-elementos de um item classe.

Os cálculos das duas métricas apresentadas serão efetuados de forma automática, baseados no histórico do projeto, armazenado nos sistemas de GCS. Além disso, as fórmulas representam um ponto de variação da abordagem, podendo ser calibradas à medida que são analisadas.

De posse desses resultados, serão geradas representações visuais que possibilitem ao gerente de configuração obter uma visão geral dos elementos do sistema. A visualização será feita por meio da utilização de modelos UML e gráficos que apresentam estatísticas. Os modelos UML foram escolhidos por serem uma notação que os desenvolvedores já conhecem e que possibilita a representação de uma classe e seus relacionamentos de forma visual e sucinta.

Cada elemento analisado será representado no diagrama de visualização utilizando técnicas de coloração. Assim, será possível identificar as partes do sistema que merecem mais atenção, em relação ao controle de concorrência. Além disso, a

abordagem irá sugerir também a política de controle de concorrência mais indicada para cada item em questão.

Uma vez aplicada a abordagem, o engenheiro de software pode fazer alterações na política de controle de concorrência ou pode alocar melhor seus recursos, tentando evitar que muitos desenvolvedores trabalhem em um elemento crítico ao mesmo tempo, minimizando possíveis conflitos. Ou ainda, o engenheiro de software pode decidir fazer uma reestruturação nos elementos críticos do sistema para diminuir o problema de concorrência e de dificuldade de junção.

Outra contribuição desse trabalho é o desenvolvimento de uma ferramenta que implemente a abordagem proposta.

3. Trabalhos Relacionados

Atualmente, a visualização de software é aplicada para diferentes propósitos relacionados à engenharia de software, como, por exemplo, suporte ao desenvolvimento de software colaborativo [EICK et al. 1992], evolução de software [GERMAN et al. 2004] e testes de software [JONES et al. 2002]. A maioria das ferramentas que implementam essas abordagens utiliza os dados gerados pela GCS, especialmente os dados gerados pelo sistema de controle de versões. A seguir, são apresentadas duas dessas ferramentas.

Seesoft [EICK et al. 1992] foi uma das primeiras ferramentas que utilizou os dados de sistemas de controle de versões para a visualização de históricos de programas. É uma ferramenta genérica para a visualização de estatísticas associadas às linhas de um arquivo texto. Seesoft usa os dados do sistema de controle de versões relativos à autoria, data e descrição das revisões. Essa ferramenta adota uma visualização baseada em linhas que mapeia cada linha de código em uma linha de apenas um pixel na tela. A cor de cada linha representa o valor do atributo que está sendo visualizado, como, por exemplo, a autoria do código.

A ferramenta softChange [GERMAN et al. 2004] tem como objetivo mostrar aos engenheiros de software como o software evoluiu desde a sua concepção. softChange extrai as informações dos sistemas de controle de versões e dos sistemas de controle de modificações. Ela é composta de um componente gráfico provê dois tipos de visão: a primeira utiliza histogramas para apresentar estatísticas, permitindo uma visão geral da evolução do projeto; e a segunda apresenta gráficos que mostram arquivos, autores e seus relacionamentos, como, por exemplo, quais autores modificaram determinado arquivo.

Como foi mencionado anteriormente, essas ferramentas utilizam os dados gerados pelos sistemas de GCS para apoiar outras áreas, mas não para apoiar aspectos da própria GCS. Além disso, elas usam notações próprias, que podem ser de difícil compreensão por parte do desenvolvedor.

4. Considerações Finais

Com o desenvolvimento deste trabalho espera-se apoiar o controle de concorrência, visando um aumento na produtividade em um projeto de desenvolvimento de software.

Este trabalho foi dividido inicialmente em quatro etapas. A primeira delas já está parcialmente concluída e consiste em uma revisão bibliográfica para a fundamentação teórica, envolvendo os conceitos de GCS e Visualização de Software. A segunda etapa já foi iniciada e tem como objetivo definir como serão calculadas as duas métricas utilizadas pela abordagem proposta; identificar as informações geradas pelos sistemas de GCS importantes para o trabalho; e formular uma metodologia para a extração e análise dessas informações. A terceira etapa do trabalho envolve a definição da abordagem de visualização e o desenvolvimento da ferramenta que implementa essa abordagem. Por fim, a quarta e última etapa consiste em avaliar a abordagem e a ferramenta desenvolvida.

Tanto a abordagem quanto a ferramenta precisam ser avaliadas. Será utilizado um estudo de caso no qual serão analisados os arquivos gerados pelos sistemas de GCS de um projeto *open-source* disponibilizados livremente na internet. A abordagem será aplicada em um dado momento do passado e será verificado se as previsões obtidas nesse momento se concretizaram no futuro.

Referências

CEMIN, C. (2001) *Visualização de Informações Aplicada à Gerência de Software*, Tese de M.Sc., Instituto de Informática, UFRGS, Porto Alegre, Rio Grande do Sul, Brasil.

EICK, S., STEFFEN, J., SUMNER, E. (1992) "Seesoft-A Tool for Visualizing Line Oriented Software Statistics", *IEEE Transactions on Software Engineering*, v. 18, n. 11 (November), pp. 957-968.

ESTUBLIER, J. (2000) "Software Configuration Management: a Roadmap". In: *International Conference on Software Engineering (ICSE)*, pp. 279-289, Limerick, Ireland, June.

GERMAN, D., HINDLE, A., JORDAN, N. (2004) "Visualizing the evolution of software using softChange". In: *Internation Conference on Software Engineering and Knowledge Engineering (SEKE)*, pp. 336-341, Banff, Anbert, Canada, June.

IEEE (2001) *Std 1540 - IEEE Standard for Software Life Cycle Processes - Risk Management*, Institute of Electrical and Electronics Engineers.

JONES, J., HARROLD, M., STASKO, J. (2002) "Visualization of Test Information to Assist Fault Localization". In: *International Conference on Software Engineering (ICSE)*, pp. 467-477, Orlando, Florida, May.

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

LINTERN, R., MICHAUD, J., STOREY, M., et al. (2003) "Plugging-in Visualization: Experiences Integrating a Visualization Tool with Eclipse". In: *ACM Symposium on Software Visualization (SoftVis)*, p. 47-ff, San Diego, California, June.

MACKINLAY, J., SHNEIDERMAN, B. (2000) *Information Visualization, Using Vision to Think*, Morgan Kaufmann.

MURTA, L. (2006) *Gerência de Configuração no Desenvolvimento Baseado em Componentes*, Tese de D.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil.

SOFTEX (2006) "MPS.BR – Guia de Implementação, Parte 5: Nível C 1.0". In: http://www.softex.br, acessado em 01/03/07.