1.
Introdução

No contexto atual de desenvolvimento de software é comum utilizar ferramentas de controle de versão (SCM – *Software Configuration Management*) para auxiliar o desenvolvimento. Este tipo de ferramenta mantém um histórico das evoluções do código fonte do software, guardando estados de acordo com a política adotada pelos seus usuários, como por exemplo, estados em que o artefato é compilável e está correto. Esses estados, comumente chamados de revisões, são armazenados em um repositório implementado de acordo com o critério adotado pelo controle de versão. O que o estado identifica, ou seja, a unidade que é versionada, depende da política adotada pelo controle de versão: ferramentas de controle de versão comerciais [CEDERQVIST, 1993] [PILATO, 2004] [MICROSOFT, 2000] [MICROSOFT, 2007] [SWICEGOOD, 2008] [MERCURIAL, 2006] [POOL, 2007] [IBM, 2008] costumam definir versões por arquivo ou por grupo de arquivos.

Cada revisão criada possui um identificador único e guarda o incremento necessário para evoluir o código da revisão anterior para a em questão. Devido a este mecanismo é possível voltar ao estado de qualquer revisão com base neste identificador. Para determinar o incremento de cada revisão o controle de versão utiliza uma ferramenta para comparar o estado corrente com o estado da revisão que foi evoluída. Cada ferramenta de comparação possui uma unidade indivisível, que representa o elemento a ser comparado. Na maioria das ferramentas comerciais esta unidade costuma ser o parágrafo (linha de texto seguida de terminador de linha), o que permite tratar de todos os arquivos de texto da mesma forma independente do tipo de conteúdo de cada um. Esta ferramenta de comparação costuma ser chamada de *diff* [OPENGROUP, 2008], devido a um aplicativo UNIX com este nome que realiza a comparação de textos utilizando o parágrafo como unidade de comparação.

Uma das principais vantagens em utilizar um controle de versão é viabilizar a evolução simultânea do código fonte, permitindo que vários desenvolvedores

produzam evoluções no código ao mesmo tempo. Este recurso pode ser implementado utilizando uma política pessimista ou uma política otimista. A política pessimista é chamada de *lock* e funciona da seguinte forma: quando um usuário começa a implementar uma evolução no software todas as unidades versionadas (arquivos, componentes, classes, métodos, etc.) que ele alterar perdem a permissão de escrita por outros usuários até que este confirme as alterações ou explicitamente remova o *lock*. Porém, muitas vezes é necessário que dois usuários queiram e até necessitem alterar simultaneamente a mesma unidade versionada. Por exemplo, um cenário em que um arquivo foi pego por um desenvolvedor para uma tarefa que durará dias, e um segundo desenvolvedor deseja apenas fazer a correção de uma falha neste mesmo arquivo (possivelmente em um trecho de código irrelevante para o primeiro usuário). O segundo desenvolvedor terá que esperar o primeiro terminar sua tarefa para realizar a correção, porém esta solução não é aceitável em um ambiente de produção onde se busca um nível alto de produtividade a fim de reduzir o *time-to-market*, que neste caso é dificultado por limitações da ferramenta. A outra política, otimista, permite que dois usuários alterem o mesmo conteúdo versionado simultaneamente, e em um momento futuro estes possam ser unificados de forma automatizada ou semi-automatizada a partir de uma ferramenta de unificação. O procedimento realizado por estas ferramentas costumam ser chamadas de *merge* devido a uma aplicação UNIX com este nome capaz de unificar textos utilizando o parágrafo como unidade atômica.

Existem dois tipos de merge: *two-way-merge* e *three-way-merge*. Generalizando o conteúdo versionado como arquivos, temos que o primeiro tipo recebe como parâmetro dois arquivos a serem unificados e realiza o procedimento apenas com as informações inferidas na comparação destes. Já o segundo tipo, recebe como parâmetro dois arquivos (A e B) a serem unificados e um arquivo base (C), cujo conteúdo representa a versão comum na história dos outros dois. A unificação deste tipo procede da seguinte forma: 1 - Compara-se A e C, guardando o resultado como *Delta1*; 2 - Compara-se B e C, guardando o resultado como *Delta2*; 3 - Unifica-se A e B, utilizando os resultados *Delta1* e *Delta2* para inferir o resultado final. Este segundo tipo de merge é mais eficiente, porém exige uma terceira entrada que nem sempre existe no modelo do controle de versão utilizado.

Até o momento vimos as duas principais ferramentas de um controle de versão (comparador e unificador), e que a ferramenta de unificação utiliza a ferramenta de comparação para inferir as diferenças que ela irá trabalhar, logo, é dependente da precisão que esta consegue ter. Então, concluímos que buscar uma ferramenta de comparação mais precisa aumenta a qualidade do controle de versão, seja para exibir diferenças ao usuário como para realizar a unificação de versões.

Sabemos também que o uso de parágrafos como unidade de comparação produz um resultado impreciso e com poucas informações sobre o que realmente foi modificado, delegando ao usuário a tarefa de identificar o trecho exato e a semântica associada a ele, ou no caso de uma unificação, impedindo que esta seja realizada automaticamente.

Considerando estas dificuldades, o trabalho que será apresentado nesta dissertação procurou criar um mecanismo de comparação de documentos textuais capaz de identificar com precisão sintática as diferenças entre duas versões de um documento. Para tanto, o mecanismo de comparação se baseia nas estruturas sintáticas dos documentos, em vez do texto puro de cada um deles. Também foi construída uma ferramenta que implementa este mecanismo de forma genérica, a fim de suportar diferentes tipos de conteúdo. Como exemplo, a fim de avaliar a ferramenta sobre o conteúdo de um tipo de documento real, foi implementado suporte ao tipo de conteúdo específico da linguagem de programação C++.

Este documento está organizado da seguinte forma: na seção 1 contextualizamos o problema e apresentamos resumidamente as dificuldades; na seção 2 apresentamos as motivações para este trabalho; na seção 3 mostramos o estado da arte atual; na seção 4 descrevemos o modelo conceitual da solução; na seção 5 apresentamos a ferramenta de comparação de documentos produzida durante esta pesquisa; na seção 6 discutimos os experimentos realizados com a ferramenta produzida; na seção 7 concluímos esta dissertação; e finalmente, na seção 8, citamos trabalhos futuros.