8.
Trabalhos futuros

Classificamos os trabalhos futuros em três categorias: melhorias no mecanismo de comparação, melhorias na ferramenta e novas ferramentas baseadas no resultado deste trabalho. As ideias relacionadas a cada uma dessas categorias será apresentada a seguir.

8.1
Melhorias no mecanismo de comparação

A primeira melhoria, e mais almejada, é a identificação de movimentações entre níveis. Como já foi comentado, o mecanismo de comparação só identifica modificações e movimentações entre elementos no mesmo nível da árvore sintática. Este recurso é suficiente para muitos tipos de escopo, porém, existem situações onde este recurso pode aumentar a precisão na identificação das modificações, como por exemplo, considerando o contexto de linguagens de programação, um trecho de código que é movido de dentro para fora de uma estrutura *If*. A solução proposta para identificar este tipo de diferença é adicionar no emparelhamento não apenas os elementos inseridos e removidos (resultado do LCS) do nível corrente, mas também os elementos das diferenças já identificadas na comparação dos sub-elementos.

Outra melhoria interessante é a criação de heurísticas para inferir operações de união, separação e clone de trechos do documento. Nesta mesma linha, também podem ser criadas heurísticas identificar operações de refatoração e realimentar o algoritmo de comparação com estas informações para desconsiderá-las.

O mecanismo de comparação proposto é restrito a documentos contendo informações textuais. Uma evolução significativa seria generalizá-lo para trabalhar com qualquer tipo de documento cuja informação pudesse ser estruturada, textual ou não. A principal característica que limita esta generalização é o sistema de coordenadas adotado para localizar estruturas dentro do documento, a *seleção*. Se criarmos uma camada de abstração onde o mecanismo

de comparação atua sobre um sistema de coordenadas abstrato, conseguimos generalizá-la.

Outra evolução é permitir que um documento seja constituído por mais de uma linguagem. Como exemplo, considere documentos do tipo JSP, que costumam apresentar em um mesmo arquivo conteúdos HTML, Java e Javascript. Devido à característica híbrida do mecanismo proposto, comparar este tipo de documento consiste em criar algoritmos de comparação de elementos adequados para cada tipo de elemento de cada linguagem, porém, é necessário também estudar quais novas informações devem ser inseridas na estrutura canônica. Com exemplo, uma ideia é criar um nó intermediário entre nós de conteúdos diferentes, a fim de informar ao mecanismo de comparação quais elementos pertencem a cada contexto, e, consequentemente, quais elementos não devem ser comparados entre si.

Outra ideia para melhorar a qualidade do resultado final é aplicar técnicas de *machine learning* [ALPAYDIN, 2004] onde uma entidade aprende a identificar padrões de diferenças com base em um conhecimento prévio e sugeri-las ao comparador, como uma fonte alternativa de informações, para auxiliá-lo a tomar decisões. Este conhecimento pode ser aprendido pelo comparador durante a sua própria utilização no dia-a-dia.

8.2 Melhorias na ferramenta

Considerando agora apenas o contexto da ferramenta, a expectativa natural é construir novos componentes para tratar de outros tipos de documento. Estipulando uma lista, as primeiras sintaxes a serem suportadas seriam das linguagens: Java, Lua, XML, HTML, Javascript, C#, HTML, etc.

Outro ponto importante é a facilidade de extensão da ferramenta: a adição de novos componentes que tratem outros tipos de documento é realizada em tempo de compilação. Uma possível solução é implementar um mecanismo para carregar bibliotecas dinâmicas (DLLs) com uma interface conhecida. Neste caso, a interface conhecida é a própria interface *DocumentTypePlugin*. Porém, ao longo do desenvolvimento sentiu-se necessidade de outros recursos comuns em linguagens de mais alto nível, como Java ou C#. Inicialmente estas linguagens não foram escolhidas por questões de desempenho, porém, uma sugestão é reescrever

a ferramenta em uma destas linguagens, mantendo os algoritmos escritos na linguagem de programação C++. Acreditamos que este passo facilite as próximas evoluções da ferramenta.

Uma ideia que surgiu ao longo do desenvolvimento foi criar um modo de visualização, específico para linguagens de programação, que permitisse ao usuário percorrer o resultado da comparação em formatos não textuais, como por exemplo em diagramas de classe, DFDs, etc.

Uma melhoria interna é eliminar as subclasses de *DocumentElement*, e substituí-las por meta-estruturas. As definições de cada meta-estrutura passariam a ser armazenadas em um arquivo de configuração. Esta melhoria facilita o suporte a um novo tipo de documento, evitando que este tenha que criar a fábrica e as subclasses de *DocumentElement* necessárias.

8.3 Novas ferramentas

Uma das motivações deste trabalho foi, a longo prazo, construir um controle de versão orientado a sintaxe. Este controle de versão deve, internamente, trabalhar apenas com a estrutura sintática dos documentos versionados, e, somente linearizá-los para edição. Este trabalho construiu as duas ferramentas principais para realizar este projeto, sendo a primeira um conversor para uma estrutura canônica, utilizada internamente pelo controle de versão. E a segunda ferramenta foi o comparador de documentos. Para concluir este projeto é necessário criar uma ferramenta de unificação de documentos e uma ferramenta para linearizar estruturas.