2. Motivação

Conforme apresentado na introdução, este trabalho procura melhorar a qualidade das diferenças identificadas na comparação de duas versões de um documento. O termo *qualidade* empregado diz respeito à **semântica** e à **precisão sintática** nos limites de cada diferença identificada. Para exemplificar que as ferramentas tradicionais apresentam um nível de qualidade insatisfatório, serão apresentados exemplos onde as diferenças identificadas por elas não refletem a semântica das operações realizadas.

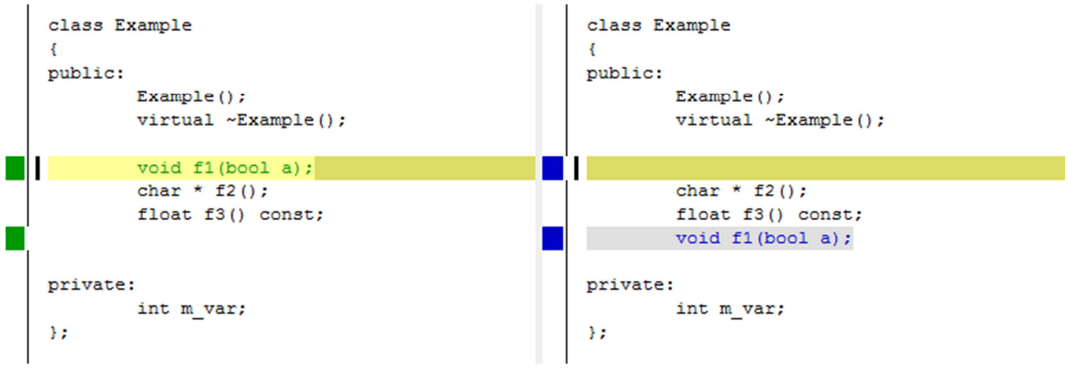
Como primeiro exemplo, considere o cenário em que a evolução consistiu em trocar a posição de um parágrafo, que representa a declaração de um método (Figura 1).

```
class Example
{
public:
	Example();
	virtual ~Example();

	void f1(bool a);
	char * f2();
	float f3() const;
private:
	int m_var;
};
```

```
class Example
{
public:
	Example();
	virtual ~Example();

	char * f2();
	float f3() const;
	void f1(bool a);
private:
	int m_var;
};
```

Figura 1 – Exemplo de código de duas versões de um documento onde a ordem das declarações é modificada.

Uma ferramenta de comparação baseada em texto acusaria (Figura 2) uma operação de remoção (a linha de onde o parágrafo foi retirado) e uma operação de inserção (a linha onde o parágrafo foi recolocado), quando o que realmente aconteceu foi uma mera reorganização de uma declaração.



```
class Example
{
public:
        Example();
        virtual ~Example();

        void f1(bool a);
        char * f2();
        float f3() const;


private:
        int m_var;
};
```

```
class Example
{
public:
        Example();
        virtual ~Example();


        char * f2();
        float f3() const;
        void f1(bool a);

private:
        int m_var;
};
```

Figura 2 – Exemplo de comparação textual sobre duas versões de um documento onde a ordem das declarações é modificada.

Em outro exemplo, considere o cenário onde a definição de um método sofreu uma reengenharia e parte dele foi levada para um novo método, disposto abaixo do original (Figura 3).

```
int MyClass:myMethod()
{
    int returnValue = 0;

    if(doSomething()) {
        int value1 = firstCall();
        int value2 = secondCall();
        returnValue += value1 * value2;
    }

    if(doMoreThings()) {
        int value3 = thirdCall();
        returnValue += value3 / M_PI;
    }

    return returnValue;
}
```

```
int MyClass:myMethod()
{
    int returnValue = 0;

    if(doSomething()) {
    int value1 = firstCall();
    int value2 = secondCall();
    returnValue += value1 * value2;
    }

    return myMethod2(returnValue);
}

int MyClass:myMethod2(int amount)
{
    if(doMoreThings()) {
    int value3 = thirdCall();
        amount += value3 / M_PI;
    }

    return amount;
}
```

Figura 3 – Exemplo de código de duas versões de um documento onde ocorre a extração de um trecho de código para um novo método.

A ferramenta de comparação baseada em texto, em alguns casos, não respeitará o limite dos escopos e indicará a diferença em uma mistura de ambos os métodos, possivelmente indicando inserções e remoções de parágrafos que nem foram alterados (Figura 4).

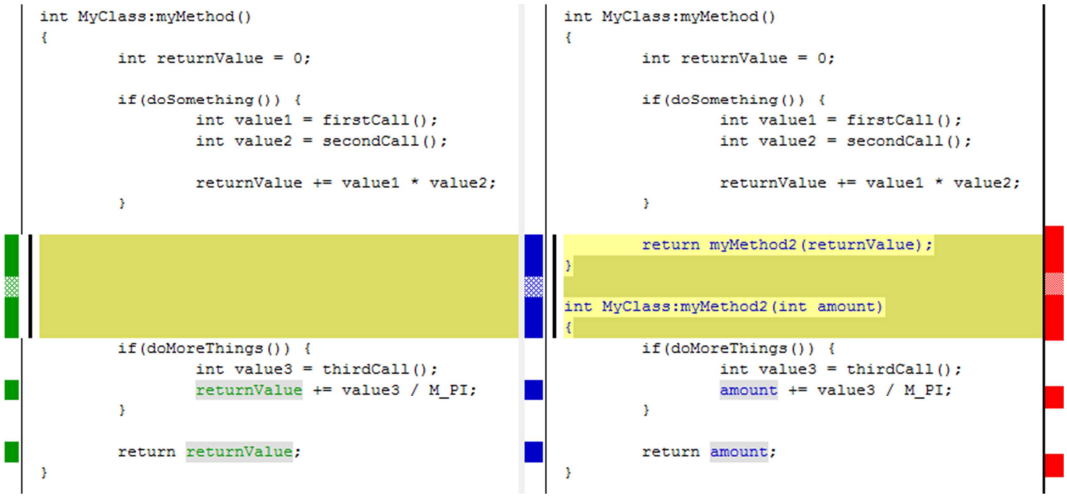


Figura 4 – Exemplo de comparação textual sobre duas versões de um documento onde um trecho de código é extraído para um novo método.

Novamente, o que realmente aconteceu foi a troca de um conjunto de operações pela chamada de um método, e a inserção de um novo método. E também, não menos importante, em cenários onde a formatação sofreu pequenas modificações, estas são mostradas como evoluções no documento, o que é perfeitamente natural, porém semanticamente não deveriam ser consideradas com o mesmo peso de uma modificação sintática.

Na literatura encontramos alguns artigos que tratam do tema de versionamento [PIETROBON, 1995] [WESTFECHTEL, *et al.*, 2001] [NGUYEN, *et al.*, 2004] [AMSTEL, BRAND e PROTIC, 2008] [JUNQUEIRA, BITTAR e FORTES, 2008], comparação [YANG, 1991] [GODFREY, 2005] [KIM e NOTKIN, 2006] e unificação [WESTFECHTEL, 1991] [GRASS, 1992] [YANG, HORWITZ e REPS, 1992] [MUNSON e DEWAN, 1994] [BUFFENBARGER, 1995] [CHAWATHE, *et al.*, 1996] [CHAWATHE e GARCIA-MOLINA, 1997] [LINDHOLM, 2001] [ZÜNDORF, WADSACK e ROCKEL, 2001] [MENS, 2002] [KLIER, 2005] de documentos utilizando unidades com granularidade mais fina, mais adequada ao tipo do conteúdo que será manipulado. Considerando o contexto de linguagens de programação como exemplo, e considerando que nesse contexto a unidade de comparação pode assumir diferentes tipos (declaração, definição, expressão, controle, comentário, etc.), ao comparar as estruturas sintáticas que representam o conteúdo de arquivos de código em vez de comparar arquivos de texto puro, é possível identificar as modificações com maior precisão sintática e também parte da semântica associada a ela. Para os exemplos citados,

idealmente a ferramenta de comparação deveria indicar: para o primeiro cenário, a movimentação de um símbolo sem consequência lógica; para o segundo cenário a troca de operações e a adição de um novo método; e, para o último, uma notificação explicita que a modificação consistiu em uma troca de formatação.

A partir desta especificação mais precisa é possível construir ferramentas que reduzam o esforço do usuário final, como por exemplo, uma ferramenta para visualização das diferenças, que apresente cada uma com precisão e indique a semântica associada a ela. Esta ferramenta também pode fornecer ao usuário mecanismos para ocultar temporariamente determinados tipos de modificações, como formatação, renomeação de símbolos, reordenação de métodos, atributos e parâmetros, etc. Ainda em relação a esta ferramenta de visualização, é possível mostrar as diferenças em formatos não textuais, como por exemplo, em diagramas. A ferramenta utilizada para unificar revisões também pode ser beneficiada por dispor de mais informações para tomar as decisões automáticas, reduzindo o esforço desprendido pelo usuário para esta tarefa.

Outra possibilidade viabilizada por esta abordagem é a criação de um mecanismo de *plugins* que operem diretamente sobre a estrutura sintática do código. Estes *plugins* podem ser disparados em operações do controle de versão tais como *commit*, *merge*, *checkout*, etc. Como exemplo, podemos criar um validador de estilo de código que valide, a partir de um padrão previamente configurado, as modificações a serem versionadas em uma operação de *commit*, e, caso sejam encontrados trechos de código fora do padrão, a criação da nova revisão é abortada e o usuário é notificado para fazer as devidas correções.

E, considerando que o controle de versão trabalha em cima de uma representação sintática, é possível determinar a formatação dos documentos quando linearizados para edição, permitindo que cada usuário utilize a formatação de sua preferência sem afetar o mecanismo de diferenças do controle de versão.

Em suma, trabalhar com a representação sintática do documento produz resultados mais precisos e mais informativos ao usuário. Porém, para alcançar estes benefícios é necessário que o mecanismo de comparação seja baseado na estrutura sintática do documento e não na sua forma textual pura. A principal dificuldade desta abordagem é que cada tipo de conteúdo apresenta uma estrutura sintática diferente, e consequentemente, a ferramenta deve ser capaz de interpretar

este conteúdo para gerar sua representação estrutural, em seguida identificar as diferenças, e finalmente extrair a sua semântica.