SOFTWARE REPOSITORY MINING ANALYTICS TO ESTIMATE SOFTWARE COMPONENT RELIABILITY

André Freitas - freitas.andre@fe.up.pt

Dissertação realizada sob a orientação do *Prof. Rui Maranhão e Alexandre Perez*

1. Motivação

O Software desempenha um papel fundamental na nossa sociedade e na nossa rotina diária, pois dependemos de aplicações para comunicar, gerir informação, etc. O desenvolvimento de Software é relativamente complexo e o custo de corrigir bugs representa até 90% do custo do projeto [1].

Os repositórios de Software têm informação valiosa que pode ser explorada com técnicas de *Machine Learning* e de *Analytics* para suportar modelos de previsão de defeitos de *Software*.

O *Crowbar* [1] é uma ferramenta de localização automática de falhas, que após a execução de uma bateria de testes, estima os componentes faltosos. O algoritmo *Barinel* usado nesta ferramenta usa estimativas estáticas [2]. Estas podem ser substituídas por estimativas dinâmicas provenientes do resultado da previsão de defeitos, de maneira a melhorar a qualidade do diagnóstico das falhas.

2. Objectivos

Os principais objetivos são:

- Prever defeitos a partir de repositórios de *Software* ao aprender quais são as variáveis mais importantes a analisar e criar um modelo de previsão baseado nas técnicas existentes;
- Melhorar o diagnóstico de falhas no *Crowbar* com base nos resultados da previsão de defeitos.

3. Descrição do Trabalho

Foi implementada uma ferramenta em *Python* denominada *Schwa*, que é capaz de analisar repositórios *Git* e estimar a probabilidade de um determinado componente possuir defeitos. Por exemplo, o *Schwa* consegue estimar que um determinado ficheiro tem uma certa de probabilidade de ter um defeito. Caso o ficheiro for de código *Java*, a granularidade é até ao método, ou seja, é possível estimar a probabilidade de classes e métodos serem defeituosos.

3.1. Instalação

O *Schwa* está disponível livremente no *Github* [2] e pode ser facilmente instalado por:

```
pip install schwa --pre
```

3.2. Utilização da ferramenta

O *Schwa* pode ser invocado como um utilitário de linha de comandos com a seguinte sintaxe:

```
schwa git/repo/path [--commits COMMITS]
```

O número de *commits* é opcional para o caso de se pretender analisar apenas as últimas alterações. É apresentado o relatório na fig. 1.

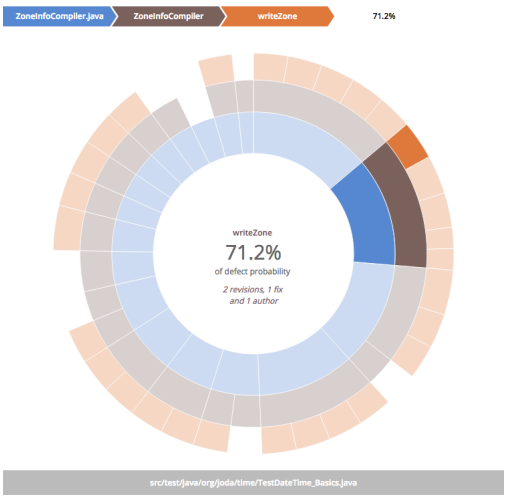

Fig. 1 – Relatório do Schwa

No gráfico da fig. 1 é possível inspecionar os componentes (ficheiros, classes ou métodos) de um modo hierárquico. Aparece no centro a estimativa da probabilidade de defeito juntamente com o os valores das métricas recolhidas, que são o número de revisões, correções e de autores do componente.

3.3. Extração

A primeira fase do processo do *Schwa* é a extração de dados do repositório que sejam relevantes para a análise. Com recurso à biblioteca *GitPython*, [3] em cada *commit* extraimos a mensagem, o autor, o *timestamp* e a lista dos componentes alterados.

[1] http://crowbar.io/
[2] https://github.com/andrefreitas/schwa
[3] https://github.com/gitpython-developers/GitPython

De maneira a interpretar as alterações de código *Java*, foi usada a biblioteca *Plyj*[4], que foi modificada para incluir a informação do número das linhas nas classes e métodos.

3.4. Análise

A análise é feita tendo em conta os seguintes princípios:

- **Princípio das revisões** Quantas mais revisões tem um componente, maior é a sua probabilidade de defeito[3];
- **Princípio das correções** Existe uma correlação entre defeitos do passado e defeitos no futuro [4];
- **Princípio dos autores** O número de autores é uma variável preditiva de defeitos [5, 6].

As variáveis utilizadas na análise são revisões, correções e autores e são recolhidas através do seguinte algoritmo:

```
For each commit in the repository
  twr = compute twr
  For each component in the commit
    component.revisions += twr
    IF commit is bug fix
        component.fixes += twr
    If is new author
        component.authors += twr
```

Em vez de se usar contadores, é usada a função *Time-Weighted-Risk* (TWR) que tem um valor máximo quando a alteração do componente é recente. Esta função recebe um *timestamp* normalizado [7].

$$twr(t_i) = \frac{1}{1 + e^{-12t_i+12}} \tag{1}$$

3.5. Modelo de previsão de defeitos

É calculado primeiro um *score* para cada componente:

$$\begin{aligned} score = revisions * revisions_{weight} \\ + fixes * fixes_{weight} \\ + authors * authors_{weight} \end{aligned} \tag{2}$$

O *score* é então normalizado:

$$defect_{probability} = 1 - e^{-score} \tag{3}$$

3.6. Experiências

Desenvolvemos um modelo de aprendizagem dos pesos das variáveis (revisões, correções e autores) com algoritmos genéticos. Após o *Schwa* aprender com projetos académicos, *Open Source* e empresariais, concluímos que os pesos variam de projeto para projeto.

[4] https://github.com/musiKk/plyj

Ao usarmos o *Schwa* no *Crowbar*, conseguimos reduzir o tempo de diagnóstico. No projecto *Joda Time* reduzimos de 1 hora para menos de 1 minuto.

4. Conclusões

Conseguimos criar uma técnica capaz de prever defeitos de componentes de *Software* com base na análise de repositórios *Git* com um relatório visual.

As técnicas atuais de revisão de código podem beneficiar da utilização do *Schwa*, pois permite aos programadores focarem os seus recursos onde os defeitos estão.

Referências

[1] F. Servant. Supporting bug investigation using history analysis. Em *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, páginas 754–757, Nov 2013.

[2] Rui Abreu, Peter Zoeteweij, e Arjan J. C. van Gemund. Spectrum-based multiple fault localization. Em *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, ASE '09, páginas 88–99, Washington, DC, USA, 2009. IEEE Computer Society.

[3] T.L. Graves, A.F. Karr, J.S. Marron, e H. Siy. Predicting fault incidence using software change history. *Software Engineering, IEEE Transactions on*, 26(7):653–661, Jul 2000.

[4] Thomas Zimmermann, Rahul Premraj, e Andreas Zeller. Predicting defects for eclipse. Em *Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, PROMISE '07, páginas 9–, Washington, DC, USA, 2007. IEEE Computer Society.

[5] Raimund Moser, Witold Pedrycz, e Giancarlo Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. Em *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, páginas 181–190, New York, NY, USA, 2008. ACM.

[6] Marco D'Ambros, Michele Lanza, e Romain Robbes. Evaluating defect prediction approaches: A benchmark and an extensive comparison. *Empirical Softw. Engg.*, 17(4-5):531–577, Agosto 2012.

[7] Chris Lewis, Zhongpeng Lin, Caitlin Sadowski, Xiaoyan Zhu, Rong Ou, e E. James Whitehead Jr. Does bug prediction support human developers? findings from a google case study. Em *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, páginas 372–381, Piscataway, NJ, USA, 2013. IEEE Press.