Distribuição de Conhecimento de Código em Times de Desenvolvimento - uma Análise Arquitetural

Mívian M. Ferreira[1], **Kecia Aline M. Ferreira**[2], **Marco Tulio Valente**[1]

[1] Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte, MG – Brasil

[2]Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG)
Belo Horizonte, MG – Brasil

{mivian.ferreira,mtov}@dcc.ufmg.br, kecia@decom.cefetmg.br

Abstract. *Understanding the distribution of knowledge about the code among the members of a development team is an important task for project management. Truck factor is a metric that indicates the number of developers whose departure impairs or complicates the survival of the project. Currently the algorithms proposed for computing this metric take into account only the amount of files authored by a developer, but not the importance of these files to the system. This study investigates whether the set developers indicated by such algorithms are those who actually have knowledge of the most complex components of the system. The results indicate that the analysis of the truck factor associated with the importance of the classes in a system brings relevant information about the influence of developers in the system.*

Resumo. *Saber como se dá a distribuição de conhecimento do código entre os membros de um time de desenvolvimento é uma questão importante para o gerenciamento de projetos. Truck factor é uma métrica que indica o número de desenvolvedores cuja saída dificulta ou inviabiliza a continuidade do projeto. Os algoritmos propostos para o cálculo dessa métrica levam em consideração apenas a quantidade de arquivos nos quais um desenvolvedor possui autoria, e não a importância desses arquivos para o sistema. O presente trabalho investiga se o conjunto de desenvolvedores indicados por tais algoritmos são aqueles que de fato possuem conhecimento dos componentes mais complexos do sistema. Os resultados sugerem que a análise do truck factor considerando a importância das classes de um sistema traz detalhamentos relevantes acerca da influência dos desenvolvedores no sistema.*

1. Introdução

Truck factor é uma métrica que tem por objetivo detectar como o conhecimento sobre o código-fonte está distribuído entre os membros de uma equipe de um projeto de software. Segundo Martin Bowler[1], essa métrica indica o número de desenvolvedores que, ao serem desligados da equipe de desenvolvimento, fazem com que o projeto entre em sérios problemas. *Truck factor* pode ser utilizada para identificar e prevenir possíveis riscos decorrentes da dependência do projeto em relação a certos desenvolvedores [Ricca et al. 2011]. Essa aplicação se faz importante principalmente em projetos

[1]http://www.agileadvice.com/2005/05/15/agilemanagement/truck-factor/

de desenvolvimento de software nos quais a taxa de saída e entrada (*i.e.*, *turnover*) de desenvolvedores tende a ser alta.

Existem na literatura alguns algoritmos para o cálculo de *truck factor* [Zazworka et al. 2010, Avelino et al. 2016]. Esses algoritmos baseiam-se na premissa de que todas as classes possuem a mesma importância para o sistema. Sendo assim, o *truck factor* de um sistema será dado pelo menor conjunto de desenvolvedores cuja remoção da equipe de desenvolvimento fará com que a maior parte dos arquivos do sistema percam todos os seus autores (*i.e.,* desenvolvedores que possuem conhecimento sobre o arquivo). Entretanto, não existem na literatura algoritmos que levem em consideração a importância das classes para o sistema, por exemplo, a quantidade de falhas associadas às classes ou às complexidades das mesmas. O presente trabalho visa contribuir nesse contexto.

Especificamente, este trabalho tem por objetivo analisar como o conhecimento dos desenvolvedores que compõem o *truck factor* de um sistema está distribuído em sua arquitetura. Buscaremos identificar uma associação mais realista entre o conhecimento dos desenvolvedores e as classes presentes nos componentes mais críticos do sistema. Para tanto, é apresentada uma avaliação empírica com quatro sistemas Java. A arquitetura dos sistemas será representada pelo modelo *Little House* [Ferreira 2011], que fornece a visualização macroscópica e genérica da arquitetura de um software.

O restante deste trabalho está organizado da seguinte forma: Seção 2 apresenta os principais conceitos e trabalhos relacionados a este trabalho; Seção 3 relata a metodologia aplicada para desenvolvimento do trabalho; Seção 4 apresenta os resultados obtidos; Seção 5 apresenta uma breve discussão sobre os achados do trabalho; Seção 6 discute as ameaças à validade do trabalho; e Seção 7 traz as conclusões e indicações de trabalhos futuros.

2. Trabalhos Relacionados

2.1. Truck Factor

Existem na literatura poucos trabalhos relacionados ao tema *truck factor*. Alguns trabalhos têm por objetivo propor abordagens para o cálculo de *truck factor* [Zazworka et al. 2010, Cosentino et al. 2015, Avelino et al. 2016]. A abordagem proposta por Zazworka et al. [2010] calcula, para cada combinação possível de desenvolvedores, o número de arquivos que continuariam com autores caso aquele conjunto de desenvolvedores fosse retirado do sistema. Porém, essa abordagem não escala para projetos com mais de 30 desenvolvedores [Ricca et al. 2011]. Já na abordagem proposta por Consentino et al. [2015], os desenvolvedores utilizados para o cálculo de *truck factor* são aqueles que possuem um determinado nível de conhecimento sobre o sistema. A utilização dessa abordagem é dificultada pela necessidade de inserção de parâmetros e métricas de conhecimento sobre código-fonte. A proposta de Avelino et al. [2016] consiste em estabelecer os autores de cada arquivo do sistema através da fórmula de *Degree of Authorship* [Fritz et al. 2014]. Após estabelecer os autores do sistema, retiram-se da lista de avaliação os autores com maior número de arquivos. As retiradas são realizadas até que mais de 50% dos arquivos estejam sem autores. Os dados referentes ao *truck factor* usados na avaliação empírica deste artigo foram obtidos pela aplicação da abordagem desenvolvida por Avelino et al. [2016].

Aplicando a abordagem desenvolvida por Zazworka et al. [2010], alguns trabalhos fazem uso do conceito de *truck factor* para identificar: *thresholds* para o uso da métricas [Torchiano et al. 2011]; dificuldades inerentes ao cálculo da métrica [Ricca et al. 2011] e a complexidade computacional do algoritmo [Hannebauer and Gruhn 2014]. Entretanto, no melhor no nosso conhecimento, não existem trabalhos que, como este, busquem correlacionar os dados obtidos através do *truck factor* de um sistema com sua arquitetura.

2.2. O Modelo *Little House*

Little House é um modelo que tem por objetivo proporcionar a visualização macroscópica e genérica da arquitetura de um software [Ferreira 2011]. Proposto com base no modelo Bow-tie ([Broder et al. 2000] apud [Ferreira 2011]) que representa o macrografo da Web, o modelo *Little House* retrata a componentização do grafo de dependências de um software desenvolvidos em Java.

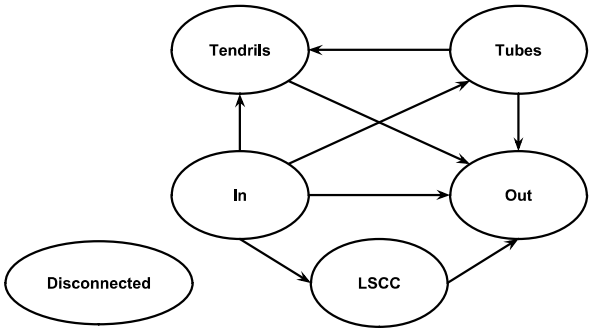

Figura 1. Modelo *Little House* [Ferreira 2011]

A Figura 1 ilustra os componentes do modelo. As classes de um sistema são agrupadas nos componentes de acordo com a seguinte regra: Uma classe A usa uma classe B, se em A existe uma referência a B. Os componentes de *Little House* são caracterizados conforme descrito a seguir:

Tubes: agrupa as classes que utilizam classes de *Tendrils* ou *Out* e são utilizadas por classes presentes no próprio componente ou em *In*.
Tendrils: agrupa classes que fazem uso de classes do componente *Out* e podem ser utilizadas por classes presentes em *In* ou *Tubes*.
In: reúne classes que usam classes de quaisquer outros componentes do modelo, exceto *Disconnected*, mas que não são utilizadas por classes existentes nos outros componentes.
Out: agrupa classes que não fazem uso de classes de outro componente e são utilizadas por classes presentes em qualquer outro componente do modelo, exceto *Disconnected*.
LSCC: é o maior componente fortemente conectado do modelo. As classes presentes neste componente podem ser alcançadas de forma direta ou indireta por quaisquer outras classes existentes neste componente.
Disconnected: aglomera classes que não são utilizadas ou fazem uso de quaisquer outras classes agrupadas nos demais componentes.

Estudos realizados com o modelo indicam que *Out* e *LSSC* são os componentes que: apresentam maior degradação ao longo da evolução do software do ponto de vista

de métricas [Ferreira et al. 2012a]; possuem piores valores de métricas quando comparados aos demais componentes [Ferreira et al. 2012b]; e são os componentes que agrupam classes com maior impacto de propagação de modificação [Ferreira et al. 2015]. Neste trabalho, busca-se identificar se os desenvolvedores apontados pelo algoritmo de cálculo de *truck factor* possuem de fato conhecimento dos componentes mais complexos do software.

3. Metodologia

O *dataset* inicial considerado utilizado neste trabalho foi o proposto por Avelino et al. [2016]. Ele é composto pelos 133 sistemas mais populares do GitHub, distribuídos em seis linguagens de programação: Java Script, Python, Ruby, C/C++, Java e PHP. Além disso, os sistemas possuem número de arquivos, número de desenvolvedores, tamanho e histórico de desenvolvimento significativos.

Os sistemas analisados neste trabalho atendem aos seguintes critérios: são escrito em Java; possuem seus *bytecodes* disponíveis ou disponibilizam mecanismos que facilitem a obtenção dos *bytecodes* (*i.e.*, presença do arquivo *pom.xml*). Esses critérios atendem às premissas de funcionamento da ferramenta *Connecta* [Ferreira 2011], que é responsável por gerar os dados do modelo *Little House*. Após aplicação dos critérios, foram selecionados quatro dos 22 sistemas Java disponíveis no *dataset* original: *Android Annotations*, um *framework* Android; *Dropwizard*, um *framework* web; *Titan*, um banco de dados de grafos; e *Glide*, um gerenciador de dependências.

Os dados relacionados ao *truck factor* (TF) foram obtidos através da ferramenta *Truck-Factor*[2] [Avelino et al. 2016]. Com essa ferramenta, foram coletados os seguintes dados: valor de TF do sistema, conjunto de desenvolvedores que formam o TF; e lista de arquivos nos quais esses desenvolvedores do TF têm autoria. A ferramenta *Connecta*[3] foi utilizada para obtenção dados referentes ao modelo *Little House* [Ferreira 2011]. A partir dos *bytecodes* do sistema, *Connecta* fornece uma lista contendo as classes do sistema e os componentes aos quais elas pertencem. Como última etapa, foi desenvolvido um *script* Java que concatena os dados das listas fornecidas pelas ferramentas *Truck-Factor* e *Connecta*. Além disso, o *script* é capaz de computar o percentual de classes que um desenvolvedor possui em um componente.

4. Resultados

Nesta seção são apresentados e analisados os resultados obtidos para os quatro sistemas estudados. Os dados obtidos estão apresentados em tabelas que reportam o total de classes por componente (Classes/Comp.), o número de classes que cada autor possui em cada componente (#Classes) e o percentual que esse número significa em relação ao total de classes do componente. Por questão de privacidade dos desenvolvedores, eles serão identificados pela sigla TFDev#.

Android Annotations

O sistema *Android Annottions* é um *framework* para desenvolvimento Android, possuindo *truck factor* 2. TFDev1 e TFDev2 são os desenvolvedores apontados como

[2]https://github.com/aserg-ufmg/Truck-Factor

[3]http://homepages.dcc.ufmg.br/ kecia/connecta_portugues.htm

críticos para o sistema, e possuem, respectivamente, 24,81% e 15,76% dos arquivos Java do sistema. A Tabela 1 apresenta os resultados para *Android Annotations*.

Tabela 1. Resultados *Android Annotations*

		TFDev1		TFDev2	
Componente	**Classes/Comp.**	**#Classes**	**%Autoria**	**#Classes**	**%Autoria**
LSCC	10	1	10%	9	90%
In	161	3	2%	89	55%
Out	10	2	20%	2	20%
Tendrils	71	10	14%	1	1%
Tubes	32	8	25%	1	3%
Disconnected	655	209	32%	46	7%

Embora TFDev1 seja indicado como o desenvolvedor cuja saída poderá gerar maior impacto para o sistema, ele não possui maior percentual de autoria nos componentes mais críticos do sistema. Isso fica a cargo do desenvolvedor TFDev2. Esse desenvolvedor possui maior percentual de autoria em dois dos três componentes mais críticos do sistema (LSCC e In). Sendo assim, ao considerar a importância das classes, o desenvolvedor cuja saída causaria maior impacto é TFDev2.

Titan

Titan é um banco de dados de grafos, otimizado para armazenar e consultar grafos com bilhões de vértices e arestas. Esse sistema possui *truck factor* 2. Os dois desenvolvedores críticos para esse sistema, TFDev3 e TFDev4, são responsáveis por 10% e 39% dos arquivos .java o sistema. Todavia, 53% dos arquivos de TFDev3 não são .java. Já no caso do desenvolvedor TFDev4, 99,7% dos arquivos dos quais ele possui autoria são .java. Conforme reporta a Tabela 2, esse sistema possui características análogas ao *Android Annotations*. Embora TFDev3 seja indicado como desenvolvedor com maior impacto no sistema, os dados indicam que TFDev4 possui maior percentual de autoria nos componentes mais críticos do sistema.

Tabela 2. Resultados *Titan*

		TFDev3		TFDev4	
Componente	**Classes/Comp.**	**#Classes**	**% Autoria**	**#Classes**	**%Autoria**
LSCC	74	1	1%	50	68%
In	214	66	31%	108	50%
Out	402	20	5%	163	41%
Tendrils	277	25	9%	48	17%
Tubes	57	4	7%	19	33%
Disconnected	371	27	7%	160	43%

Dropwizard e Glide

Os sistemas *Dropwizard* e *Glide* possuem *truck factor* 1. As Tabelas 3 e 4 apresentam, respectivamente, os resultados dos sistemas. Diferente dos resultados obtidos nos sistemas descritos *Android Annotations* e *Titan*, nos sistemas que possuem *truck factor* iguais a 1, observa-se que os desenvolvedores indicados pelo *truck factor* possuem maior percentual de autoria nos componentes mais complexos do sistema. No sistema

Dropwizard, o desenvolvedor TFDev5 possui maior percentual absoluto de autoria nos componentes LSCC e In e mais $1/4$ das classes do componente Out. Já no sistema *Glide*, o principal desenvolvedor, TFDev6, possui percentual de autoria significativa nos três componentes mais críticos do sistema.

Tabela 3. Resultados *Dropwizard*

		TFDev5	
Componente	**Classes/Comp.**	**#Classes**	**%Autoria**
LSCC	6	5	83%
In	46	24	52%
Out	106	40	38%
Tendrils	406	112	28%
Tubes	47	13	28%
Disconnected	293	89	30%

Tabela 4. Resultados *Glide*

		TFDev6	
Componente	**Classes/Comp.**	**#Classes**	**% Autoria**
LSCC	32	23	72%
In	5	2	40%
Out	220	126	57%
Tendrils	40	19	48%
Tubes	1	0	0%
Disconnected	93	44	47%

5. Discussão

Este trabalho investiga como é distribuído, na arquitetura do sistema, o conhecimento dos desenvolvedores que compõe seu *truck factor*. O objetivo deste estudo é identificar uma associação mais realista entre o conhecimento dos desenvolvedores e as classes presentes nos componentes mais críticos do sistema. Para isso, a arquitetura dos quatro sistemas Java avaliados foi representada pelo modelo *Little House*.

Embora a quantidade de software estudados seja pequena, os sistemas apresentados no estudo são relevantes. Foram utilizados quatro sistemas de código aberto, desenvolvidos em Java e cotados entre os mais populares do repositório GitHub. Além disso, a amostra conta com uma diversidade de domínios. Essas características são relevantes, uma vez que o estudo está relacionado ao conhecimento dos desenvolvedores sobre o código-fonte dos sistemas onde atuam.

Os resultados encontrados neste trabalho indicam conclusões importantes a respeito do cálculo de *truck factor*. Existem indícios de que, a importância das classes para o sistema seja um fator a ser considerado no cálculo da métrica. Como observado nos sistemas com *truck factor* 2, os desenvolvedores apontados como tendo o segundo maior impacto no sistema, foram aqueles que apresentam maior percentual de autoria nos componentes mais críticos sistema. Entretanto, a mesma afirmação não é válida para os projetos cujo *truck factor* é 1. Nesses sistemas, o desenvolvedor apontado pelo *truck factor* é de fato quem possui maior autoria (*i.e.,* conhecimento) nos componentes críticos.

A conclusão principal a que se chega a partir deste estudo é que os desenvolvedores responsáveis pelos componentes mais críticos da arquitetura do software são indicados dentre os desenvolvedores listados pelo *truck factor*. Portanto, a análise realizada pelo *truck factor* necessita ser aprimorada para que possa capturar com mais fidelidade a influência dos desenvolvedores no projeto, especialmente no que tange às questões arquiteturais.

6. Ameaças à Validade

Validade Externa. Para a realização da avaliação empírica, foram utilizados quatro sistemas *opensource* desenvolvidos em Java. Embora a amostra não seja grande, foram utilizados sistemas relevantes, de alta complexidade e de diferentes domínios, o que propicia maior representatividade para o *dataset* utilizado. Como o estudo se baseou apenas em sistemas *opensource*, não é possível afirmar que os resultados aqui obtidos possam ser generalizados para sistemas proprietários.

Validade de Construção. Neste trabalho consideramos apenas os arquivos .java de um sistema. Entretanto, é comum a existência de arquivos com outras extensões em um sistema. Sendo assim, podem existir arquivos tão importantes quanto os arquivos Java, mas que não foram incluídos na análise.

7. Conclusões

Truck factor é uma métrica utilizada para identificar a distribuição de conhecimento em projetos de desenvolvimento de software. De maneira geral, a métrica tem como objetivo identificar o conjunto de desenvolvedores cuja saída faz com que o desenvolvimento ou a manutenção do software fiquem em risco. No entanto, os algoritmos existentes para o cálculo da métrica não levam em consideração a importância relativa das classes de um sistema. Este trabalho teve por objetivo identificar como o conhecimento dos desenvolvedores, apontados como *truck factor* do sistema, está distribuído na arquitetura do sistema.

Neste trabalho foi realizada uma análise empírica quatro sistema Java. Buscou-se identificar se os desenvolvedores apontados com *truck factor* são aqueles que possuem maior percentual de autoria nos componentes mais críticos da arquitetura do sistema. Para tanto, a arquitetura do sistema foi representada através no modelo *Little House*. Os resultados foram obtidos através da comparação entre os dados fornecidos pelo modelo e os dados fornecidos pela ferramenta *Truck-Factor*.

Android Annotations e *Titan*, apresentaram *truck factor* 2. Nesses sistemas, detectou-se que o desenvolvedor apontado pela ferramenta *Truck-Factor* como aquele cuja saída causaria maior impacto no sistema, não é aquele que de fato que possui maior conhecimento nos componentes mais críticos dos sistemas. Em ambos os sistemas, os dados obtidos para os segundos desenvolvedores indicam que eles são os desenvolvedores críticos no caso de arquivos .java. Já nos sistemas *Dropwizard* e *Glide*, com *truck factor* 1, identificou-se que os desenvolvedores indicados são de fato aqueles que possuem maior percentual de autoria nos componentes críticos da arquitetura utilizada para a avaliação. Esses resultados indicam que a análise do *truck factor* associada à importância da classe do ponto de vista arquitetural pode revelar informações mais precisas acerca da influência dos desenvolvedores no sistema.

Como trabalhos futuros são propostos: *survey* com os desenvolvedores dos sistema para validação dos dados obtidos no estudo; proposta de uma abordagem que faça uso da relevância das classes de um sistema no cálculo de *truck factor*.

Agradecimentos

Essa pesquisa foi apoiada pela FAPEMIG e CAPES.

Referências

Avelino, G., Passos, L., Hora, A., and Valente, M. T. (2016). A novel approach for estimating truck factors. In *24th International Conference on Program Comprehension (ICPC)*, pages 1–10.

Cosentino, V., Izquierdo, J., and Cabot, J. (2015). Assessing the bus factor of git repositories. In *22nd International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 499–503.

Ferreira, K. A. M. (2011). *Um modelo de predição de amplitude da propagação de modificações contratuais em software orientado por objetos*. PhD thesis, Universidade Federal de Minas Gerais.

Ferreira, K. A. M., Moreira, R. C. N., Bigonha, M. A. S., and Bigonha, R. S. (2012a). The evolving structures of software systems. In *3rd International Workshop on Emerging Trends in Software Metrics (WETSoM)*, pages 28–34.

Ferreira, K. A. M., Moreira, R. C. N., Bigonha, M. A. S., and Bigonha, R. S. (2012b). A generic macroscopic topology of software networks - a quantitative evaluation. In *26th Brazilian Symposium on Software Engineering (SBES)*, pages 161–170.

Ferreira, M. M., Ferreira, K. A. M., and Neto, M.-H. T. (2015). Mapping the potential change impact in object-oriented software. In *30th ACM Symposium on Applied Computing (ACM-SAC)*, pages 1654–1656.

Fritz, T., Murphy, G. C., Hill, M.-E., Ou, J., and Hill, E. (2014). Degree-of-knowledge: modeling a developer's knowledge of code. *Software Engineering and Methodology*, 23(2):14:1–14:42.

Hannebauer, C. and Gruhn, V. (2014). Algorithmic complexity of the truck factor calculation. In *Product-Focused Software Process Improvement*, pages 119–133. Springer.

Ricca, F., Marchetto, A., and Torchiano, M. (2011). On the difficulty of computing the truck factor. In *12th International Conference on Product-focused Software Process Improvement (PROFES)*, pages 337–351.

Torchiano, M., Ricca, F., and Marchetto, A. (2011). Is my project's truck factor low?: theoretical and empirical considerations about the truck factor threshold. In *2nd International Workshop on Emerging Trends in Software Metrics (WETSoM)*, pages 12–18.

Zazworka, N., Stapel, K., Knauss, E., Shull, F., Basili, V. R., and Schneider, K. (2010). Are developers complying with the process: an xp study. In *4th International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 14:1–14:10.