Test Case Recommender: um sistema de recomendação para alocação automática de testes baseada no perfil do testador

Breno Miranda[1], **Juliano Iyoda**[1], **Silvio Meira**[1]

[1]Centro de Informática
Universidade Federal de Pernambuco
Cidade Universitária – CEP 50740-540 Recife-PE – Brazil

{bafm,jmi,srlm}@cin.ufpe.br

Abstract. *Software testing is an arduous and expensive activity. In the context of manual testing, any effort to reduce the test execution time is helpful. Allocating tests in an effective way can be a good strategy to reduce the execution time and, consequently, increase test productivity. This paper describes a new approach to the allocation problem using a Recommender System. This work is developed in the context of an industrial cooperation with the BTC (Brazil Test Center), where test allocation is not a trivial task: at the beginning of a new test phase for certain product, the Test Manager is responsible for allocating hundreds of test cases among the testers available. We developed a tool to recommend a test allocation based on the tester's effectiveness and experience in executing a given test case. The usage of our tool and a comparison between the proposed approach and the manual allocation method are illustrated in a case study. Preliminary results showed a precision varying between* 43.08% *and* 44.69%.

Resumo. *A atividade de Teste de Software pode ser bastante árdua e custosa. No contexto de testes manuais, todo esforço é válido no sentido de reduzir o tempo de execução dos testes. Alocar os testes de maneira eficiente pode ser uma boa estratégia para reduzir este tempo de execução e, consequentemente, aumentar a produtividade. Este artigo apresenta uma nova abordagem para o problema de alocar testes a testadores utilizando Sistemas de Recomendação. Este trabalho foi desenvolvido no contexto de uma cooperação industrial com o BTC (Brazil Test Center), onde a alocação de testes não é uma tarefa trivial: ao início de uma nova fase de testes para determinado produto, o Gerente de Testes tem a tarefa de alocar algumas centenas de testes entre os testadores disponíveis. Uma ferramenta foi desenvolvida para automatizar o processo de alocação dos testes baseada em fatores como a efetividade e a experiência de cada testador naquele caso de teste. A utilização desta ferramenta e uma comparação do método proposto com o método de alocação manual são ilustradas em um estudo de caso. Resultados preliminares mostram uma precisão entre* $43,08\%$ *e* $44,69\%$.

1. Introdução

A atividade de Teste é fundamental para garantir o desenvolvimento de *software* com qualidade, uma vez que ela representa uma revisão das especificações, do design e

da codificação. Tendo em vista que esta atividade pode ser bastante árdua e custosa [Beizer 1990], todo esforço é válido no sentido de otimizar a criação, seleção, alocação e execução de testes. A proposta deste trabalho é abordar o problema de alocação de testes utilizando Sistemas de Recomendação [Konstan 2004].

Este trabalho faz parte do projeto *Brazil Test Center* (BTC), uma cooperação entre a Motorola e o Centro de Informática da Universidade Federal de Pernambuco (CIn/UFPE). O BTC executa testes caixa-preta em celulares, ou seja, os testes são criados a partir da especificação do *software* e não há acesso ao código-fonte. A maioria dos testes são executados manualmente.

Ao início de uma nova execução, o Gerente de Testes fica encarregado de alocar os testes entre os testadores disponíveis. Esta alocação é feita de forma manual e intuitiva, baseada na experiência do Gerente. Em alguns casos, o Gerente de Testes apenas informa quais testadores estarão envolvidos naquela execução e a quantidade de teste que cada um deve alocar para si próprio. Mais uma vez a alocação é feita de forma manual e intuitiva; desta vez, baseada na experiência e preferência dos próprios testadores.

Analisar centenas de testes manualmente e distribuí-los entre os testadores disponíveis de forma a otimizar a execução (executar mais testes em menos tempo identificando o maior número possível de defeitos) pode ser uma atividade bastante custosa. Este trabalho descreve uma ferramenta que faz uso de técnicas de Sistemas de Recomendação para alocar testes de forma automática de acordo com o perfil dos testadores: o *Test Case Recommender* (TCR). Esta ferramenta analisa a relevância de cada teste a ser executado para cada um dos testadores e sugere uma distribuição de modo que cada testador seja alocado para os testes mais relevantes de acordo com o perfil selecionado pelo Gerente de Testes.

As principais contribuições deste artigo são: a utilização de uma abordagem mais racional para a atividade de alocação de Testes utilizando técnicas de Sistemas de Recomendação; a definição de dois perfis (*effectiveness* e *expertise*) criados a partir da análise de dados históricos de execuções; a implementação de uma ferramenta para automatizar a atividade de alocação de testes e um estudo de caso ilustrando o uso da ferramenta com Planos de Testes reais executados pelo BTC. O perfil *effectiveness* retrata o testador baseado na sua capacidade de encontrar defeitos. O perfil *expertise* captura a experiência do testador. O estudo de caso rodou a ferramenta considerando-se os dois perfis e em duas alocações distintas (dois Planos de Testes). Resultados preliminares mostraram uma precisão (em comparação com a alocação do gerente) variando de $43,08\%$ a $44,69\%$.

Sistemas de recomendação têm sido aplicados a quase todas atividades da Engenharia de Software. Muitos trabalhos descrevem sistemas de recomendação de alocação de tarefas a pessoas [Anvik et al. 2006, Mockus and Herbsleb 2002, Minto and Murphy 2007, Pereira et al. 2010, Jeong et al. 2009], prevenção de defeitos e depuração [Li and Zhou 2005, Livshits 2005, Giger et al. 2010]. Poucos estão diretamente ligados a testes [Kpodjedo et al. 2008] e nenhum (até onde sabemos), relacionados a alocação de testes a testadores.

As próximas seções estão organizadas da seguinte forma: a Seção 2 apresenta a técnica utilizada para modelar os perfis dos testadores e o cálculo de similaridade entre um teste e um testador. A Seção 3 explica detalhes da implementação da ferramenta. Um

estudo de caso, que ilustra a nossa abordagem, é apresentado na Seção 4. A Seção 5 apresenta os trabalhos relacionados e a Seção 6 conclui.

2. Sistemas de Recomendação Aplicados a Alocação de Testes

Sistemas de Recomendação (SR) [Konstan 2004] buscam fornecer sugestões personalizadas baseadas nas preferências do usuário. Estes sistemas utilizam técnicas de filtragem da informação para recomendar novos itens a partir da sua comparação com o perfil do usuário. Estes sistemas têm sido utilizados, principalmente, em sites de comércio eletrônico através da indicação de produtos aos seus usuários, considerando suas necessidades específicas. Por exemplo, na livraria Amazon.com (http://www.amazon.com), ao selecionarmos um livro L, o site automaticamente recomenda outros livros comprados por clientes que compraram L. A princípio, estes livros estariam relacionados ao livro L e poderiam induzir o cliente a efetuar mais compras. Isso contribui para o aumento das vendas e da fidelidade dos clientes [Bezerra and de Carvalho 2010]. Outro exemplo, onde o uso dos SR tem proporcionado uma clara vantagem competitiva é no mercado de entretenimento [Schafer et al. 2001]. Como exemplo, pode-se citar os guias de programação personalizados em TVs digitais e recomendações de músicas em estações de rádio.

2.1. Construindo o Perfil Simbólico Modal do Testador

Utilizamos uma técnica similar àquela apresentada por [Bezerra and de Carvalho 2010] para a criação de dois perfis de testador: de *expertise* e de *effectiveness*. Porém, antes de descrever o algoritmo de recomendação proposto neste trabalho, é conveniente introduzir a terminologia utilizada no BTC e em Sistemas de Recomendação. Introduziremos tais conceitos através de um exemplo simples.

Uma *feature* é uma funcionalidade coesa e identificável [Turner et al. 1998]. No caso do BTC, por exemplo, "suportar múltiplas listas de contato no applicativo de mensagens instantâneas" é uma *feature*. Cada *feature* no BTC possui um identificador numérico único e cada teste está associado a uma ou mais *features*. Um *componente* é o agrupamento das diversas *features* de um mesmo domínio. Cada teste está também associado a um ou mais componentes. Existem outros atributos do teste, mas, para ilustrar a técnica, utilizaremos apenas *feature* e componente. Um atributo será chamado aqui de *variável*.

A Tabela 1 mostra um *item* do nosso sistema de recomendação. O item, no nosso caso, é o caso de teste. No caso da Amazon.com, por exemplo, o item é o livro. A Tabela 1 mostra um caso de teste que testa as *features* fictícias 015 e 112 e os componentes, também fictícios, *Component A*, *Component B* e *Component C*. As demais variáveis foram omitidas por simplicidade.

Tabela 1. Descrição do item CT25 (caso de teste 25).

Variável	Valor
Feature	*015, 112*
Componente	*Component A*, *Component B*, *Component C*
...	...

Cada item será representado por variáveis cujos valores terão pesos uniformes. Esta é uma simplificação deste trabalho. Uma alternativa para trabalho futuro é consultar

testadores experientes para julgar o melhor peso de cada valor. A Tabela 2 mostra como os pesos são calculados para um dado teste. Cada variável tem peso total 1 e cada valor de uma variável multivalorada tem peso igualmente distribuído entre seus valores. Por exemplo, *Component A*, *Component B* e *Component C* têm, cada um, $1/3$ de peso.

Tabela 2. Descrição simbólica modal do item CT25.

Variável	Valores e Pesos
Feature	(015, $1/2$), (112, $1/2$)
Componente	(*Component A*, $1/3$), (*Component B*, $1/3$), (*Component C*, $1/3$)
...	...

Coletamos os pesos de todos os testes e compilamos o resultado em uma tabela similar à Tabela 3. Note que, nesta tabela, apenas as variáveis *feature* e componente aparecem e, por simplicidade, utilizaremos valores como F1, F2, ... para *features* e C1, C2, ... para componentes. Neste projeto (e no restante deste documento), iremos trabalhar apenas com estas duas variáveis. Pretendemos utilizar mais variáveis em trabalhos futuros.

Tabela 3. Testes com suas respectivas descrições de conteúdo.

Caso de Teste	*Feature*	Componente
CT01	(F1, $1/2$), (F5, $1/2$)	(C1, $1/3$), (C3,$1/3$), (C4,$1/3$)
CT02	(F2, $1/3$), (F4, $1/3$), (F8, $1/3$)	(C3, $1/2$), (C5, $1/2$)
CT03	(F3, $1/2$), (F5, $1/2$)	(C4, 1)
CT04	(F3, $1/3$), (F6, $1/3$), (F7, $1/3$)	(C2, $1/2$), (C5, $1/2$)
...	...	...

Precisamos agora definir como medir as *especialidades* de um testador. Ou seja, qual métrica utilizar para descrever, por exemplo, que o Testador03 é mais especialista no teste CT02 que o Testador01. Nesta seção, *especialidade* será medida pela experiência que o testador tem em executar testes olhando-se para as variáveis *feature* e componente. O *perfil* que captura todas as características dos testes executados por determinado testador será chamado de perfil *expertise*. No estudo de caso apresentado na Seção 4, trabalharemos tanto com o perfil *expertise*, quanto com o perfil *effectiveness*, que leva em conta a eficácia do testador em achar defeitos executando um determinado teste. A Tabela 4 ilustra um histórico de execuções.

Tabela 4. Histórico de Execuções.

	CT01	CT02	CT03	CT04	CT05
Testador01	**X**		**X**	**X**	
Testador02	**X**	**X**			**X**
Testador03		**X**	**X**	**X**	
...	...	...	...	...	...

A Tabela 5 mostra a descrição simbólica do perfil *expertise* do Testador03 montada levando em conta o histórico de execuções mostrado na Tabela 4 e a descrição dos testes apresentada na Tabela 3.

Tabela 5. Descrição Modal Simbólica do *Testador03* (Perfil *Expertise*).

Testador	*Feature*	Componente
...	...	...
Testador03	(F2, $1/9$), (F3, $5/18$), (F4, $1/9$), (F5,$1/6$), (F6, $1/9$), (F7, $1/9$), (F8, $1/9$)	(C2, $1/6$), (C3, $1/6$), (C4, $1/3$), (C5, $1/3$)
...	...	...

Como o Testador03 executou os casos de teste CT02, CT03 e CT04, calculamos os pesos de cada *feature* e componente associados a estes testes. Por exemplo, calculamos os pesos das *features* F2, F3, F4, F5, F6, F7 e F8, pois elas estão presentes nos testes CT02, CT03 e CT04 (Tabela 3). O peso de uma *feature* para um testador é a média dos pesos desta *feature* entre os testes executados. Por exemplo, os pesos de F3 são 0 (CT02), $1/2$ (CT03) e $1/3$ (CT04). Somamos $0 + 1/2 + 1/3$ e dividimos pelo número de testes sendo avaliados (no nosso caso, 3 testes). Portanto, o peso de F3 no perfil *expertise* para o Testador03 é

$$\frac{1}{3} \cdot \left(0 + \frac{1}{2} + \frac{1}{3}\right) = \frac{1}{3} \cdot \frac{3+2}{6} = \frac{1}{3} \cdot \frac{5}{6} = \frac{5}{18}.$$

O peso das demais *features* e dos componentes é calculado da mesma forma.

No estudo de caso da Seção 4, trabalhamos também com o perfil *effectiveness*. A única diferença entre o perfil *expertise* e o perfil *effectiveness* é que, no último, apenas os testes que foram executados pelo testador e *falharam* é que são considerados para a criação da descrição do perfil do testador.

Com relação ao perfil *effectiveness*, é importante ressaltar que, no contexto de testes manuais, a especificação nem sempre é completa ou livre de ambiguidades: valores de entrada não são bem definidos ou passos de testes dão margem à diferentes execuções. Como consequência, testadores diferentes podem chegar a resultados diferentes executando o mesmo teste (dependendo dos valores de entrada que foram utilizados). Além disso, o repositório de testes utilizado neste trabalho conta com diferentes tipos de testes, incluindo testes de exploratório. Neste caso, a responsabilidade de encontrar um defeito é mais do testador que do teste.

2.2. Gerando a Lista de Recomendações

Esta seção descreve como calcular o grau de similaridade entre um teste e um testador. O grau de similaridade é um número de 0 a 1. Quanto mais próximo de 1, mais recomendável é alocar o teste ao testador. Calcularemos a similiaridade do caso de teste CT06 (Tabela 6) com o Testador03 (Tabela 5).

Tabela 6. Caso de teste CT06.

Caso de Teste	*Feature*	Componente
CT06	(F3, $1/2$), (F7, $1/2$)	(C2, $1/3$), (C5,$1/3$), (C7,$1/3$)

Seja $C = \{C2, C5\}$ o conjunto dos componentes comuns a CT06 e Testador03. Sejam $CT = \{C7\}$ e $T = \{C3, C4\}$ os conjuntos dos componentes exclusivos de CT06 e exclusivos de Testador03, respectivamente. Calcularemos quatro somatórios de pesos:

α, β, γ e δ. O somatório α é a soma dos pesos de CT06 para os elementos de C, enquanto o somatório β é a soma dos pesos do Testador03 para os elementos de C.

$$\alpha = \frac{1}{3} + \frac{1}{3} = \frac{2}{3} \qquad\qquad \beta = \frac{1}{6} + \frac{1}{3} = \frac{1+2}{6} = \frac{1}{2}.$$

O somatório γ é a soma dos pesos do Testador03 para os elementos de T e o somatório δ é a soma dos pesos de CT06 para os elementos de CT.

$$\gamma = \frac{1}{6} + \frac{1}{3} = \frac{1+2}{6} = \frac{1}{2}. \qquad\qquad \delta = \frac{1}{3}.$$

Note que α e β são componentes comuns que capturam a concordância entre o perfil do Testador03 com o caso de teste CT06. Já os somatórios γ e δ capturam a discordância entre o perfil do Testador03 e o caso de teste CT06. O cálculo de *dissimilaridade* em relação aos componentes é dado pela equação

$$\frac{1}{2} \cdot \left(\frac{\gamma + \delta}{\alpha + \gamma + \delta} + \frac{\gamma + \delta}{\beta + \gamma + \delta} \right) = \frac{1}{2} \cdot \left(\frac{\frac{1}{2} + \frac{1}{3}}{\frac{2}{3} + \frac{1}{2} + \frac{1}{3}} + \frac{\frac{1}{2} + \frac{1}{3}}{\frac{1}{2} + \frac{1}{2} + \frac{1}{3}} \right) \cong 0,59.$$

O cálculo de dissimilaridade para a variável *feature* é feito da mesma forma. Neste caso, o resultado (omitido aqui por simplicidade), é aproximadamente $0,50$. A dissimilaridade total é a média das dissimilaridades parciais: $(0,59 + 0,50)/2 = 0,55$. A similaridade é o complemento deste valor: $1 - 0,55 = 0,45$. Portanto, o Testador03 e o caso de teste CT06 têm similaridade de 45%. [Bezerra and de Carvalho 2010] descrevem em detalhes todas as fórmulas utilizadas no nosso sistema de recomendação.

3. A Ferramenta

O *Test Case Recommender* (TCR) foi desenvolvido em *Python* [Lutz 2006] com suporte do ambiente de desenvolvimento integrado *Eclipse* [Holzner 2004]. A escolha desta linguagem se deu, principalmente, pela facilidade de comunicação com o repositório de testes através de consultas SQL. Outras características como a facilidade de implementação e manutenção do código, também contaram positivamente para a escolha da linguagem.

O primeiro passo executado pelo TCR é a criação dos perfis dos testadores. Este passo se dá através da comunicação com o repositório de testes para a coleta de dados referentes às execuções anteriores. A partir da manipulação destes dados, é possível criar tanto o perfil *expertise*, quanto o perfil *effectiveness* de cada testador. O perfil *expertise* considera todos os testes executados por um testador e, a partir daí, monta o perfil para as variáveis *feature* e componente.

Para a execução do segundo passo, o TCR recebe como entrada (do Gerente de Testes) o Plano de Testes que será executado, a lista dos testadores disponíveis para aquela execução e o perfil que será considerado para a alocação dos testes (*expertise* ou *effectiveness*). Com estes dados, o *Test Case Recommender* verifica se já existe na base de dados local uma descrição de todos os testes existentes naquele Plano de Testes. Caso algum teste ainda não possua a descrição, o TCR se comunica mais uma vez com o repositório de testes para coletar os dados necessários para a sua criação. Em seguida, o TCR calcula

a relevância de cada teste (através do grau de similaridade) para cada um dos testadores disponíveis para aquela execução.

O terceiro e último passo é a distribuição dos testes (alocação) entre os testadores disponíveis. Neste passo, o Gerente de Testes pode optar pela distribuição igualitária dos testes ou especificar a quantidade exata de testes que deve ser alocada para cada testador. Isto permite uma flexibilidade maior para o Gerente de Testes nas situações em que há restrições: um determinado testador está alocado na execução de mais de um plano de testes simultaneamente, por exemplo. Um pré-processamento é executado para definir a ordem na qual o TCR irá distribuir os testes entre os testadores.

O pré-processamento é executado da seguinte forma: as combinações (teste x testador) são ordenadas de acordo com o grau de similaridade (obtido no passo anterior). Em seguida, o teste da combinação (teste x testador) que possui *maior* grau de similaridade, e portanto maior relevância, é alocado para o testador daquela combinação. Veja um exemplo na Tabela 7, onde os testadores são chamados de T01, T02, etc. A primeira alocação é (CT02,T03). O algoritmo segue buscando o par de maior similaridade ainda não alocado. Por exemplo, a próxima alocação é (CT10,T01). Note que o par (CT02,T01) é descartado pois CT02 foi alocado anteriormente a T03. Estes passos são repetidos até que cada um dos testadores disponíveis tenha recebido o seu primeiro teste. Após a primeira alocação, o algoritmo reinicia em busca do segundo teste para cada testador. Caso o Gerente de Testes tenha definido diferentes quantidades de teste para os testadores disponíveis, uma vez que o número desejado de testes é atingido, o testador é removido da fila de alocação e a distribuição continua para os testadores remanescentes.

Tabela 7. Definindo a ordem para distribuição dos testes.

1o.	2o.	3o.	4o.	5o.	6o.	7o.	8o	9o.
45,78%	42,75%	39,65%	37,19%	35,17%	33,63%	31,46%	29,44%	27,45%
CT02	CT03	CT04	CT02	**CT10**	CT01	**CT06**	CT09	**CT07**
T03	T03	T03	T01	**T01**	T01	**T04**	T01	**T02**

4. Estudo de Caso

O estudo de caso apresentado nesta seção tem como principal objetivo comparar o método proposto na Seção 2 com o método de alocação manual realizado pelo Gerente de Testes do BTC. Foram definidos os seguintes objetivos, perguntas e métricas (*Goal, Question, Metric* [Basili et al. 1994]):

Goal: Comparar se a alocação automática realizada pelo *Test Case Recommender* é igual à alocação manual realizada pelo Gerente de Testes.

Question 1: Os casos de teste alocados pelo TCR utilizando o perfil *expertise* são os mesmos selecionados pelo Gerente?

Question 2: Os casos de teste alocados pelo TCR utilizando o perfil *effectiveness* são os mesmos selecionados pelo Gerente?

Metric: Sejam F e G os conjuntos de pares *(testador, teste)* alocados pela ferramenta e pelo gerente, respectivamente. Para medirmos o quão parecidas foram as

alocações de ambos, calcularemos a precisão da ferramenta através do tamanho da interseção destes conjuntos:

$$\frac{\#(F \cap G)}{\#F}$$

O estudo de caso foi executado de acordo com as seguintes etapas:

Seleção dos Planos de Teste. Para a realização deste estudo de caso, dois Planos de Teste executados pelo BTC foram escolhidos aleatoriamente: PT1 e PT2. O plano PT1 possui 51 casos de teste que foram executados por dois testadores. Já o Plano PT2 possui 70 casos de teste que foram executados por 4 testadores.

Coleta de dados da alocação manual. Nesta etapa, os dados referentes à alocação manual foram coletados para os dois planos previamente escolhidos. Estes dados servem para a comparação posterior entre alocação manual e alocação automática, bem como, para definir a lista de testadores que deve ser utilizada como entrada para a execução da ferramenta.

Execução da Ferramenta. A execução foi feita em 3 etapas:

1. *Criação dos Perfis dos Testadores.* Para cada um dos planos selecionados, a data de início da execução foi coletada e os perfis dos testadores foram gerados daquela data para trás.
2. *Seleção dos Testadores Disponíveis.* Para garantir uma comparação mais precisa, a ferramenta recebeu como entrada a mesma lista de testadores que estiveram envolvidos durante a execução real daquele plano. O mesmo foi feito para definir as quantidades de teste que cada testador deveria receber.
3. *Geração das Alocações.* Definimos inicialmente a quantidade de testes que cada testador deveria receber e, por fim, executamos a geração das alocações nos planos PT1 e PT2.

4.1. Análise dos resultados

A maior interseção entre alocação manual e automática foi de 49,02%, observada no perfil *effectiveness* para o Plano de Testes PT1. A menor interseção ocorreu entre o perfil *effectiveness* e o Plano de Testes *PT2* e foi de 37,14%. A interseção média foi de 43,08% para o perfil *effectiveness* e de 44,69% para o perfil *expertise*. Este estudo de caso oferece indícios, ainda que preliminares, de que a alocação manual feita pelo gerente e a alocação automática diferem consideravelmente. Uma maior precisão pode ser conseguida se considerarmos mais variáveis de teste e outros perfis de testadores.

5. Trabalhos Relacionados

[Anvik et al. 2006] utilizam sistemas de aprendizagem de máquina para alocar programadores a *bugs*. Observando alocações no passado, o sistema infere qual o melhor programador para consertar um defeito. A precisão alcançada foi de 57% para o projeto Eclipse e 64% para o Firefox. [Mockus and Herbsleb 2002] propuseram um sistema de identificação de *expertise*. O sistema identifica, através de análises no histórico de mudanças do sistema, quais são os especialistas em determinadas ferramentas e tecnologias. [Pereira et al. 2010] propuseram um *framework* para alocação de pessoas geograficamente distribuídas para desenvolver componentes juntos. Esta alocação de times se dá em um contexto de desenvolvimento de Linhas de Produto de Software.

PR-Miner [Li and Zhou 2005] é uma ferramenta que identifica regras (padrões) de programação implícitas. Por exemplo, a chamada das funções *lock()* e *unlock()* acontece sempre em pares. A ferramenta não apenas identifica estes padrões recorrentes, como também detecta violações deles. Tais violações são potenciais *bugs* sendo introduzidos. DynaMine [Livshits 2005] minera o histórico de mudanças e, assim como PR-Miner, extrai padrões de programação e suas violações. [Giger et al. 2010] analisam o histórico de correção de defeitos para estimar quanto tempo um determinado defeito pode levar para ser consertado. [Jeong et al. 2009] propuseram um modelo baseado em Cadeias de Markov para reduzir a quantidade de vezes que um *bug* é realocado para uma pessoa. Nos projetos Mozilla e Eclipse, entre 37% e 44% dos *bugs* são realocados para outro desenvolvedor, implicando em perda de tempo na correção. [Kpodjedo et al. 2008] propuseram uma métrica que mede, em um sistema orientado a objetos, que classes merecem mais a atenção do testador (ou do gerente na alocação de esforço de teste). Uma classe é crítica se ela sofre mudanças frequentes e se tem alta conectividade.

Os trabalhos acima são uma amostra pequena de sistemas de recomendação aplicados a alocação de pessoas, prevenção de defeitos ou correção de defeitos. Não há, até onde sabemos, sistemas de recomendação para alocação de testes a testadores.

6. Conclusão e Trabalhos Futuros

Neste trabalho, descrevemos uma nova abordagem para o problema de alocação de testes utilizando sistemas de recomendação. Após a realização do estudo de caso, ficou evidente que a alocação proposta pela ferramenta é diferente daquela realizada pelo Gerente de Testes. Apesar de não ser possível afirmar que a alocação proposta pela ferramenta é mais efetiva, pode-se ao menos garantir que ela proporciona uma análise mais detalhada tanto dos testes, quanto dos testadores. Tal análise torna-se inviável de ser realizada manualmente para uma quantidade grande de casos de teste.

Para avaliarmos a proposta apresentada neste trabalho pretendemos utilizar duas métricas amplamente empregadas na avaliação de Sistemas de Recomendação: *precison* (ilustrada no estudo de caso) e *recall*, que mede a quantidade de pares (teste x testador) corretamente sugeridos pela ferramenta em relação ao conjunto de todos os pares recomendados. Pretendemos comparar diferentes variações do algoritmo apresentado em um experimento controlado. Também como trabalho futuro, pretendemos adicionar novas variáveis aos testes e novos perfis aos testadores.

Agradecimentos

Os autores agradecem o apoio do Instituto Nacional de Ciência e Tecnologia para Engenharia de Software (INES). Juliano Iyoda agradece à FACEPE pelo apoio financeiro APQ-0074-1.03/07.

Referências

Anvik, J., Hiew, L., and Murphy, G. C. (2006). Who should fix this bug? In *Proceedings of the International Conference on Software Engineering*, pages 361–370, New York, NY, USA. ACM.

Basili, V. R., Caldiera, G., and Rombach, H. D. (1994). *Goal Question Metric paradigm. Encyclopedia of Software Engineering.* John Wiley & Sons.

Beizer, B. (1990). *Software testing techniques (2nd ed.)*. Van Nostrand Reinhold Co., New York, NY, USA.

Bezerra, B. L. D. and de Carvalho, F. T. (2010). Symbolic data analysis tools for recommendation systems. *Knowledge and Information Systems*, pages 1–34. 10.1007/s10115-009-0282-3.

Giger, E., Pinzger, M., and Gall, H. (2010). Predicting the fix time of bugs. In *Proceedings of the International Workshop on Recommendation Systems for Software Engineering*, pages 36–40, Cape Town, South Africa.

Holzner, S. (2004). *Eclipse*. O'Reilly Media, Inc.

Jeong, G., Kim, S., and Zimmermann, T. (2009). Improving bug triage with bug tossing graphs. In van Vliet, H. and Issarny, V., editors, *Proceedings of the European Software Engineering Conference and the International Symposium on Foundations of Software Engineering*, pages 111–120. ACM.

Konstan, J. A. (2004). Introduction to recommender systems: Algorithms and evaluation. *ACM Transactions on Information Systems*, 22(1):1–4.

Kpodjedo, S., Ricca, F., Galinier, P., and Antoniol, G. (2008). Not all classes are created equal: Toward a recommendation system for focusing testing. In *Proceedings of the International Workshop on Recommendation Systems for Software Engineering*, Atlanta, Georgia, USA.

Li, Z. and Zhou, Y. (2005). PR-Miner: automatically extracting implicit programming rules and detecting violations in large software code. In *Proceedings of the European Software Engineering Conference and the International Symposium on Foundations of Software Engineering*, pages 306–315. ACM.

Livshits, B. (2005). Dynamine: Finding common error patterns by mining software revision histories. In *Proceedings of the European Software Engineering Conference and the International Symposium on Foundations of Software Engineering*, pages 296–305. ACM Press.

Lutz, M. (2006). *Programming Python.* O'Reilly Media.

Minto, S. and Murphy, G. C. (2007). Recommending emergent teams. In *MSR*, page 5. IEEE Computer Society.

Mockus, A. and Herbsleb, J. D. (2002). Expertise browser: a quantitative approach to identifying expertise. In *Proceedings of the International Conference on Software Engineering*, pages 503–512, New York, NY, USA. ACM.

Pereira, T. A. B., dos Santos, V. S., Ribeiro, B. L., and Elias, G. (2010). A recommendation framework for allocating global software teams in software product line projects. In *Proceedings of the International Workshop on Recommendation Systems for Software Engineering*, pages 36–40, Cape Town, South Africa.

Schafer, J. B., Konstan, J. A., and Riedl, J. (2001). E-commerce recommendation applications. *Data Min. Knowl. Discov.*, 5(1-2):115–153.

Turner, C. R., Wolf, A. L., Fuggetta, A., and Lavazza, L. (1998). Feature engineering. *Proceedings of the 9th International Workshop on Software Specification and Design*, page 192.