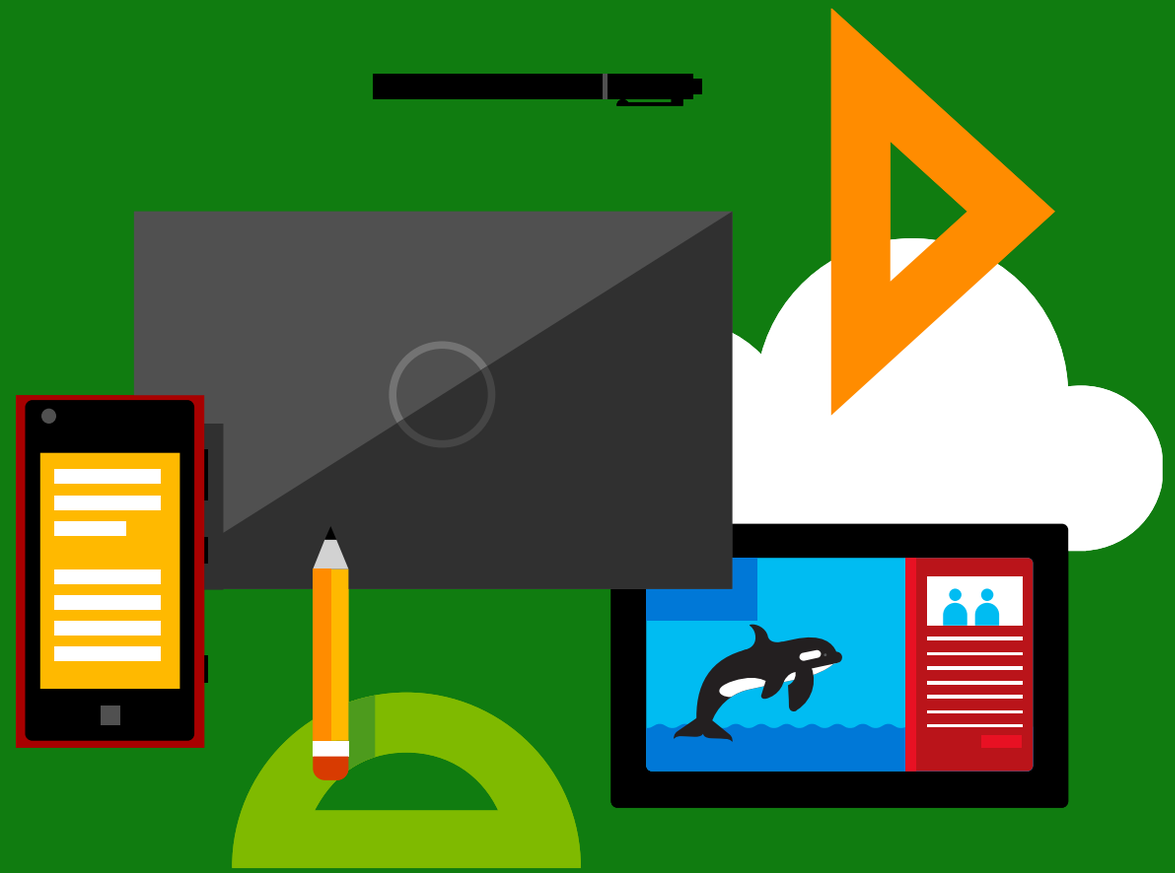
Verificação e Validação

Técnicas para Geração de Casos de Testes

Júlio Pereira Machado (julio.machado@pucrs.br)

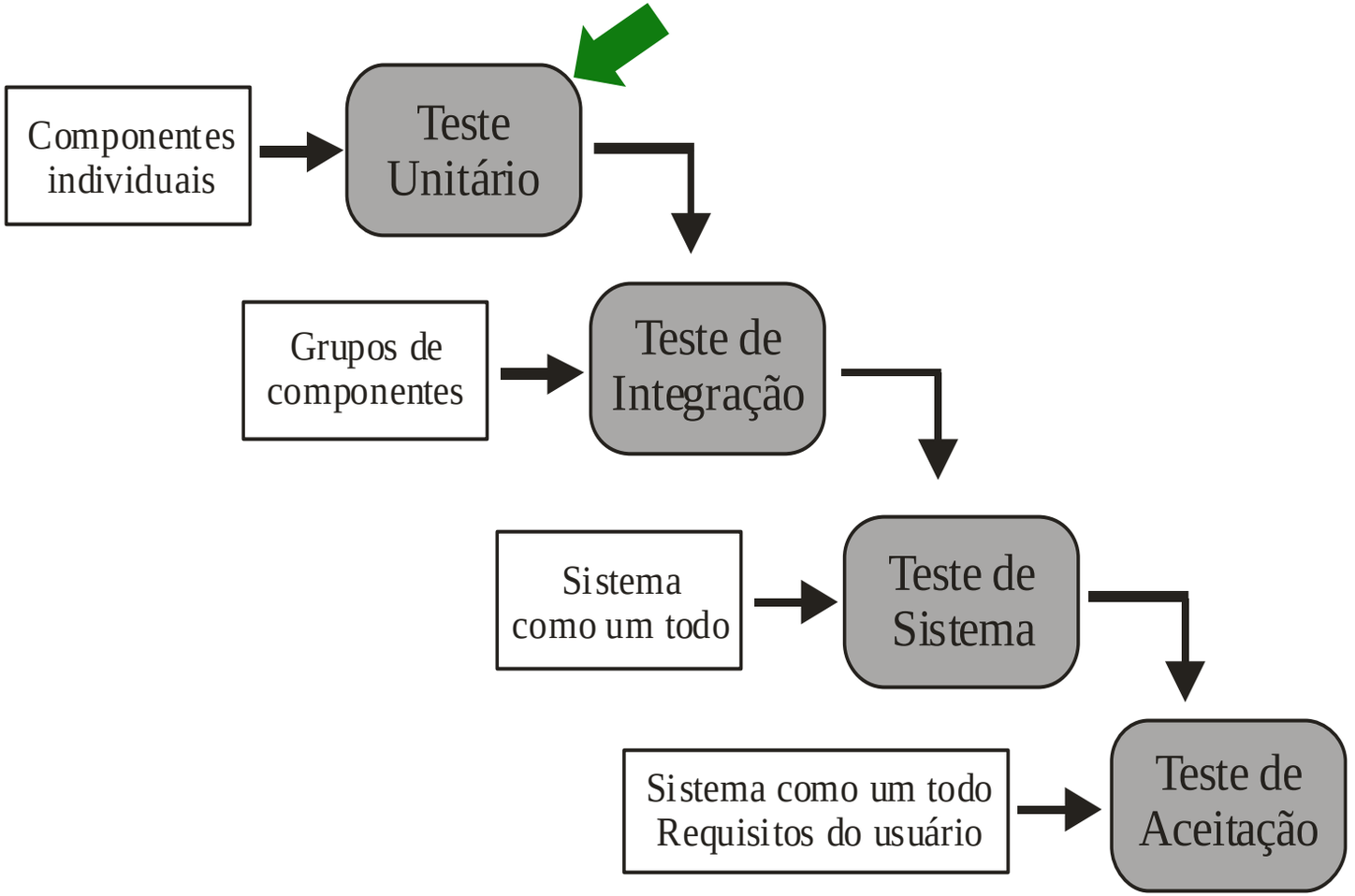
Introdução



Teste funcional

Níveis de testes

Níveis de testes



How Developers Engineer Test Cases: An Observational Study

Maurício Aniche*, Christoph Treude†, Andy Zaidman*
*Delft University of Technology - The Netherlands
†University of Melbourne - Australia
m.f.aniche@tudelft.nl, christoph.treude@unimelb.edu.au, a.e.zaidman@tudelft.nl

Abstract—One of the main challenges that developers face when testing their systems lies in engineering test cases that are good enough to reveal bugs. And while our body of knowledge on software testing and automated test case generation is already quite significant, in practice, developers are still the ones responsible for engineering test cases manually. Therefore, understanding the developers' thought- and decision-making processes while engineering test cases is a fundamental step in making developers better at testing software. In this paper, we observe 13 developers thinking-aloud while testing different real-world open-source methods, and use these observations to explain how developers engineer test cases. We then challenge and augment our main findings by surveying 72 software developers on their testing practices. We discuss our results from three different angles. First, we propose a general framework that explains how developers reason about testing. Second, we propose and describe in detail the three different overarching strategies that developers apply when testing. Third, we compare and relate our observations with the existing body of knowledge and propose future studies that would advance our knowledge on the topic.

Index Terms—software engineering, software testing, developer testing.

1 Introduction

Software testing is an important and challenging software development activity. No wonder large software companies such as Google [1, 2], Microsoft [3], and Facebook [4] have long been investing in software testing and ensuring that their developers master different techniques. However, while we know that testing accounts for a large part of the software development process, time-wise and cost-wise [5], many developers do not see testing as their favourite task [6] and, worse, do not feel productive when testing [7].

The difficulty in testing lies in devising test cases that are good enough to reveal bugs. To mitigate this, the software testing research community has been working on different approaches for a long time. The current advances in the areas of random testing [8], search-based software testing [9], or even more recently, neural bug finding [10], have already been helping developers in finding bugs they cannot do manually. However, these tools are not yet able to fully replace humans due to hard problems (e.g., the oracle problem [11]) that researchers still need to overcome. In practice, this means that software developers are still majorly responsible for testing their software systems.

Since the advent of agile methodologies [12] and, more specifically, software development methodologies that value technical aspects such as Extreme Programming [13], the popularity of (automated) unit testing among software developers has increased drastically. In fact, 67% of the 1,112 developers that completed the 14th Annual State of the Agile Report survey [14] affirmed to perform unit testing, the most applied engineering practice among the surveyed ones. The idea of "developer testing", as suggested by Meszaros [15] and Tadinger [16], where developers are also the ones responsible for testing, indeed became a prominent practice.

The popularity of the practice has led to an explosion of books on the topic. A simple search on Amazon for "software testing" returns more than 1,000 books. Well-known practitioners such as Fowler [17], Beck [13, 18], Martin [19], Freeman and Pryce [20], and Hunt and Thomas [21], have written about the advantages of automated unit tests as well as how to pragmatically test software systems. Academics have also been proposing software testing theories, practices, and techniques, in forms of books. Among them, we mention the books of Myers et al. [22], Pezzè and Young [23], and Mathur [24]. These books largely focus on explaining our current body of knowledge on how to perform domain testing, equivalence partitioning, boundary testing, and structural testing (e.g., [25, 26, 27, 28, 29, 30, 31]).

While our body of knowledge on software testing is already quite significant, we again argue that developers are still the ones responsible for putting all these techniques together. Therefore, understanding the developers' thought- and decision-making processes on, e.g., how they reason about what test cases to write, which techniques to apply, what types of questions they face when testing, and how they decide it is time to stop, is a fundamental step in making developers better at testing software.

In this paper, we observe 13 developers thinking-aloud while testing different real-world open source methods, and use these observations to explain *how developers engineer test cases*. We then challenge and augment our main findings by surveying 72 software developers on their testing practices.

We discuss our results from three different angles. First, we propose a general framework that explains how developers reason about testing. The framework contains six

Testes Sistemáticos!

- Desenvolvedores raramente são sistemáticos
- Desenvolvedores ou utilizam os requisitos ou o próprio código para organizar os casos de teste, mas raramente ambos

Casos de teste

- Um caso de teste é um par ordenado: $(d, S(d))$
  - $d \in D$ é um elemento de um domínio D
  - $S(d)$ é o resultado esperado para uma dada unidade de especificação quando d é utilizado como entrada
- Usualmente representados sob forma de tabela:

Valores de Entrada	Resultados Esperados
10032 25 "Jorge Andrade"	"Usuario cadastrado"
10033 0 "Luis Santos"	"Idade inválida – usuário não cadastrado"

Casos de teste

- A verificação completa de um programa P envolve o teste de P com um conjunto de casos de teste T que compreende todos os elementos do domínio de entrada D
- Como usualmente D é infinito ou muito grande utiliza-se um conjunto de técnicas (ou critérios) para geração de casos de teste que auxiliam o testador a organizar casos de teste potencialmente reveladores de defeitos

Casos de teste

- Exemplo: double computeMedia(int a, int b, int c)
  - Em uma máquina de 32-bits, cada variável possui cerca de 4 bilhões de valores possíveis
  - Acima de 80 octilhões (10^{27}) de testes possíveis
  - Praticamente, o espaço de valores de entrada é considerado infinito

Técnicas para geração de casos de teste

- Já foi visto que o teste exaustivo não é possível
- Deve-se buscar determinar o menor conjunto de casos de teste que seja potencialmente revelador de erros
- As estratégias de geração de casos de teste tem exatamente este propósito: oferecer norteadores para a seleção de casos de teste potencialmente reveladores de defeitos

Técnicas para geração de casos de teste

- Existem várias técnicas de geração de casos de teste
- Cabe ao desenvolvedor/testador analisar a situação e buscar identificar qual ou quais as técnicas mais adequadas
- Quanto mais “missão crítica” for a unidade a ser testada, maior o cuidado que se deve ter com a seleção e aplicação das técnicas

Técnicas para geração de casos de teste

Técnicas de teste caixa-preta: usam apenas a especificação como base para a geração dos casos de teste

Técnicas de teste caixa-branca: usam o código fonte da unidade a ser testada como base para a geração dos casos de teste

Técnicas que serão apresentadas

- Teste baseado na especificação
- Teste de valor limite
- Teste estrutural
- Teste baseado em modelos
- Teste baseado em contratos
- Teste de propriedades