How (Much) Do Developers Test?

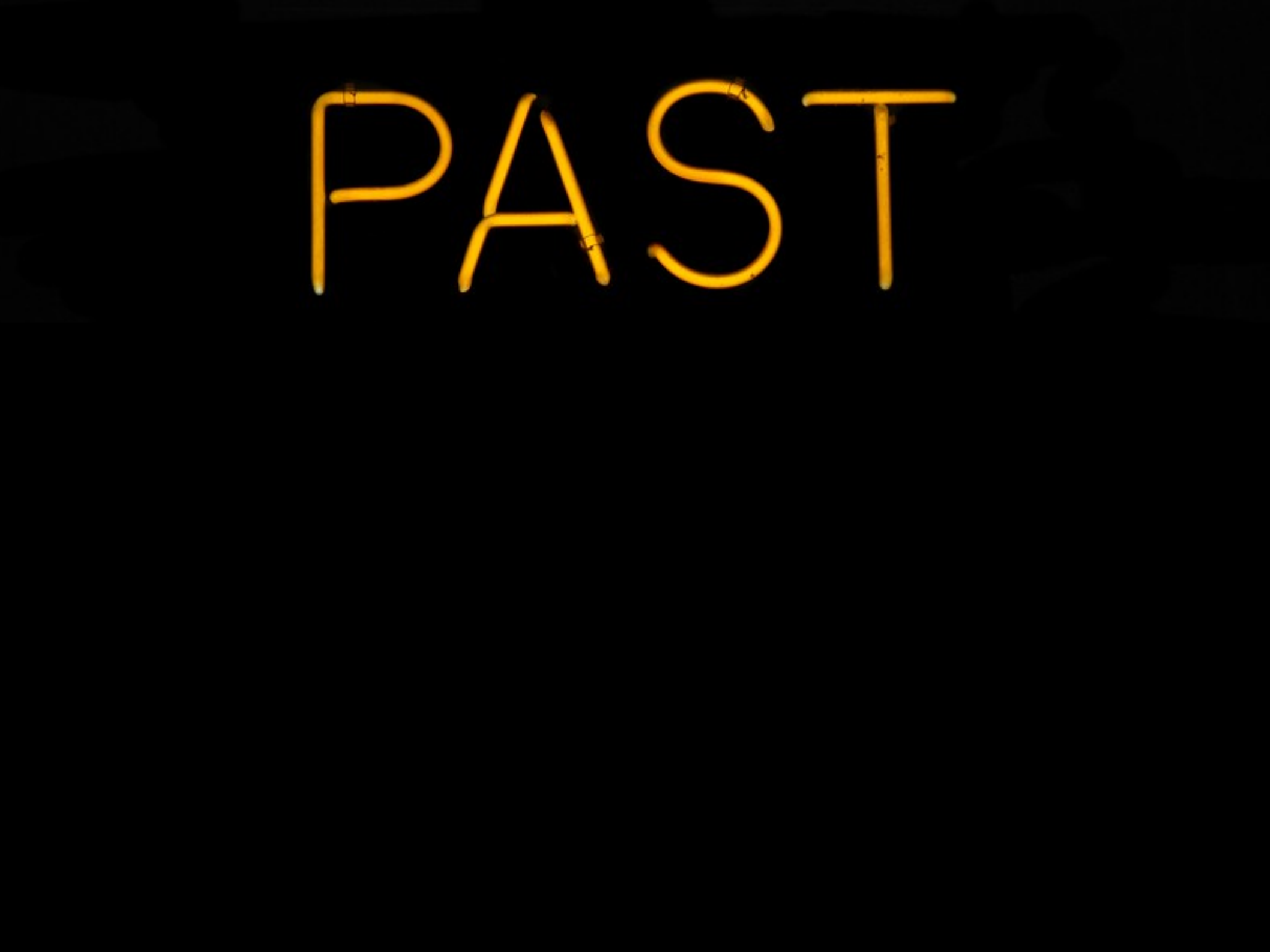
Moritz Beller

 @Inventitech

TU Delft

Delft University of Technology

Andy Zaidman, Georgios Gousios, Annibale Panichella, Igor Levaja


PAST

index

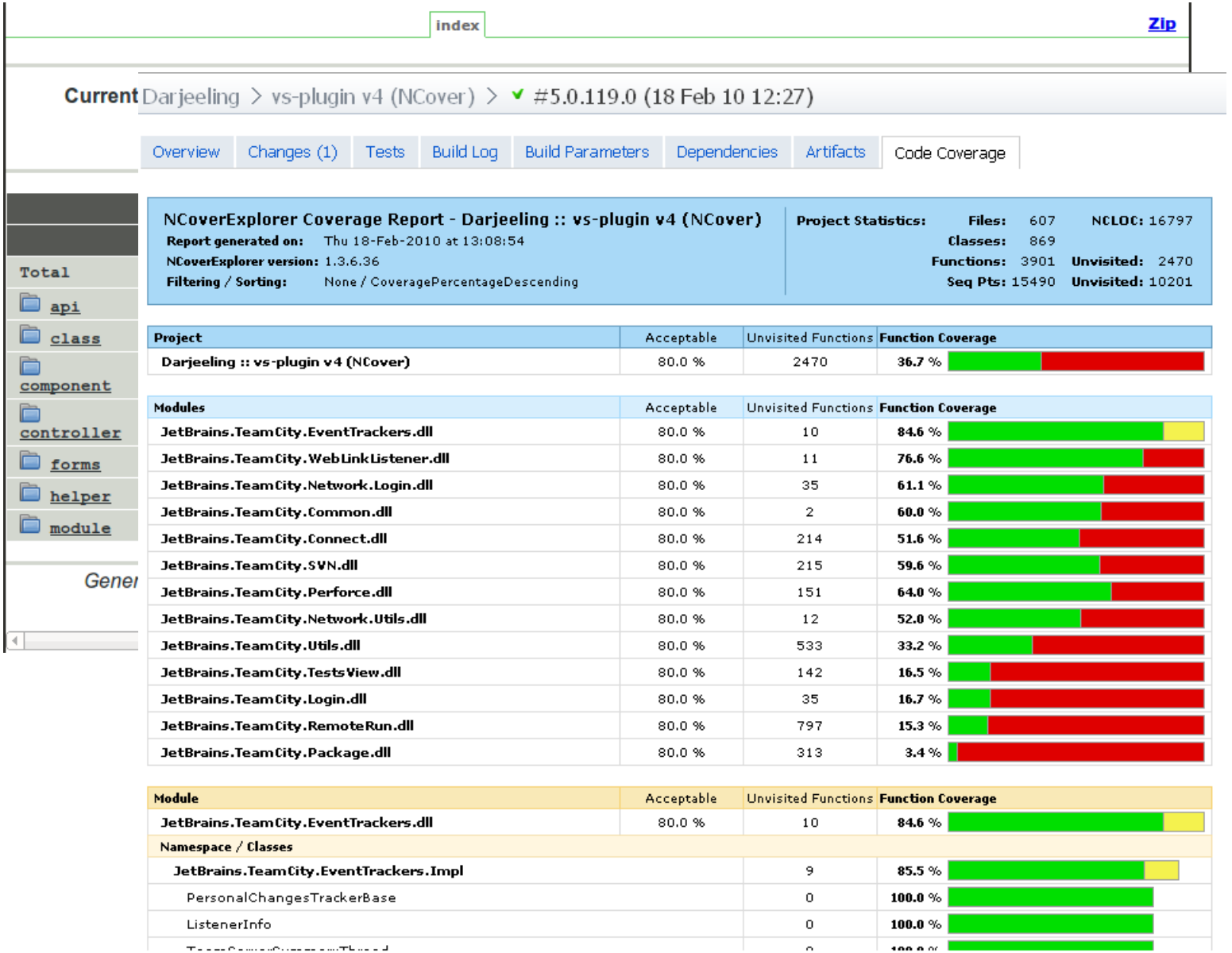
Zip

Current directory: /home/jenkins/workspace[redacted]
(dashboard)

Legend: Low: 0% to 35% | Medium: 35% to 70% | High: 70% to 100%

	Coverage								
	Lines			Functions / Methods			Classes		
Total		84.27%	3707 / 4399		81.08%	510 / 629		72.31%	94
api		77.83%	179 / 230		62.50%	20 / 32		50.00%	
class		98.68%	820 / 831		95.03%	153 / 161		94.59%	35
component		88.05%	140 / 159		85.71%	12 / 14		50.00%	
controller		72.19%	701 / 971		71.84%	74 / 103		42.86%	6
forms		81.65%	178 / 218		77.78%	7 / 9		80.00%	
helper		99.62%	260 / 261		97.92%	47 / 48		50.00%	
module		82.65%	1429 / 1729		75.19%	197 / 262		67.65%	46

Generated by PHP_CodeCoverage 1.1.2 using PHP 5.3.3 and PHPUnit 3.6.10 at Wed Jun 13 19:36:30 BST 2012.



index

Zip

Current Darjeeling > vs-plugin v4 (NCover) > #5.0.119.0 (18 Feb 10 12:27)

Overview | Changes (1) | Tests | Build Log | Build Parameters | Dependencies | Artifacts | Code Coverage

Total
api
class
component
controller
forms
helper
module

Gener

NCoverExplorer Coverage Report - Darjeeling :: vs-plugin v4 (NCover)

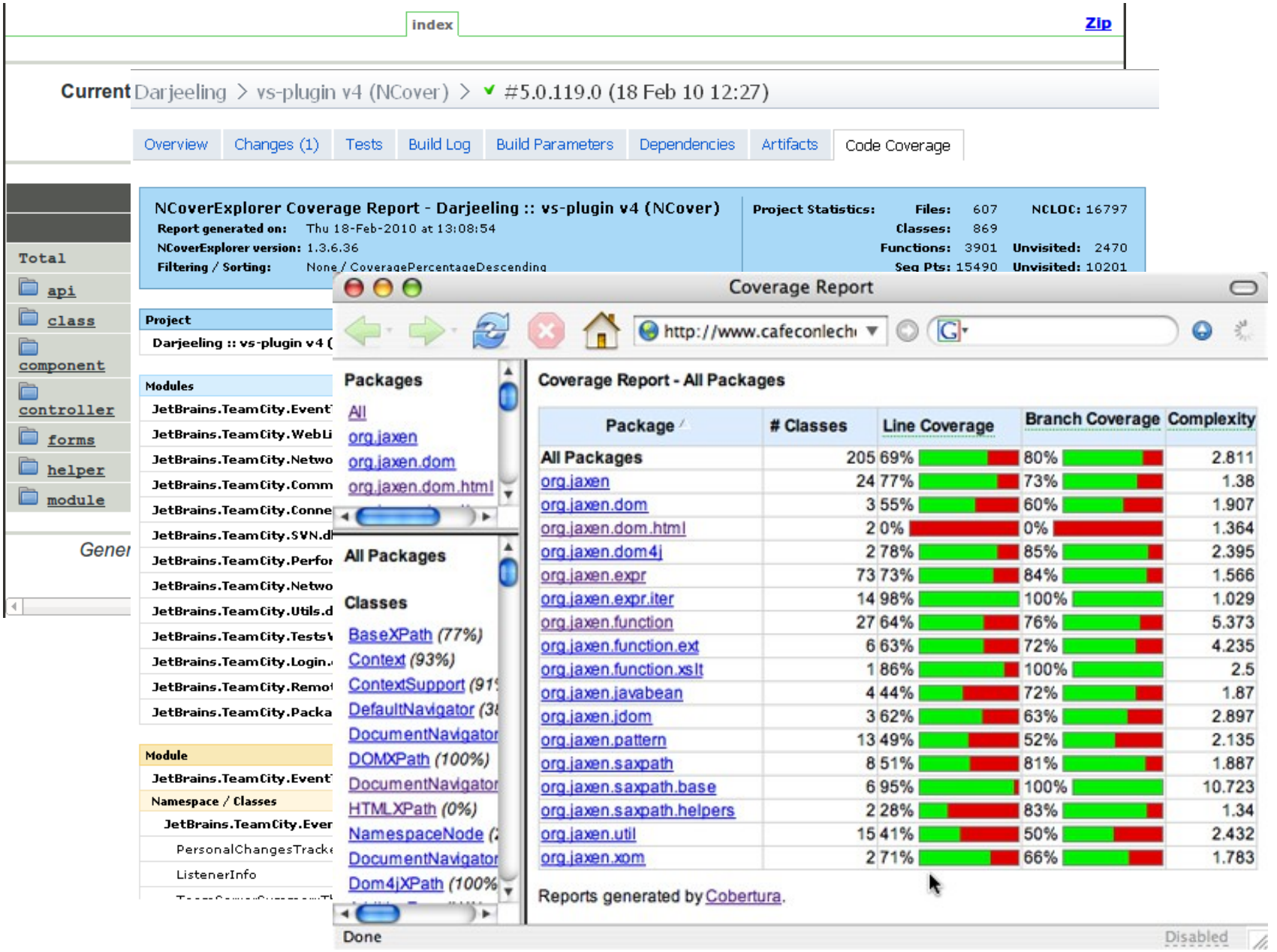
Report generated on: Thu 18-Feb-2010 at 13:08:54
NCoverExplorer version: 1.3.6.36
Filtering / Sorting: None / CoveragePercentageDescending

Project Statistics: **Files:** 607 **NCLOC:** 16797 **Classes:** 869 **Functions:** 3901 **Unvisited:** 2470 **Seq Pts:** 15490 **Unvisited:** 10201

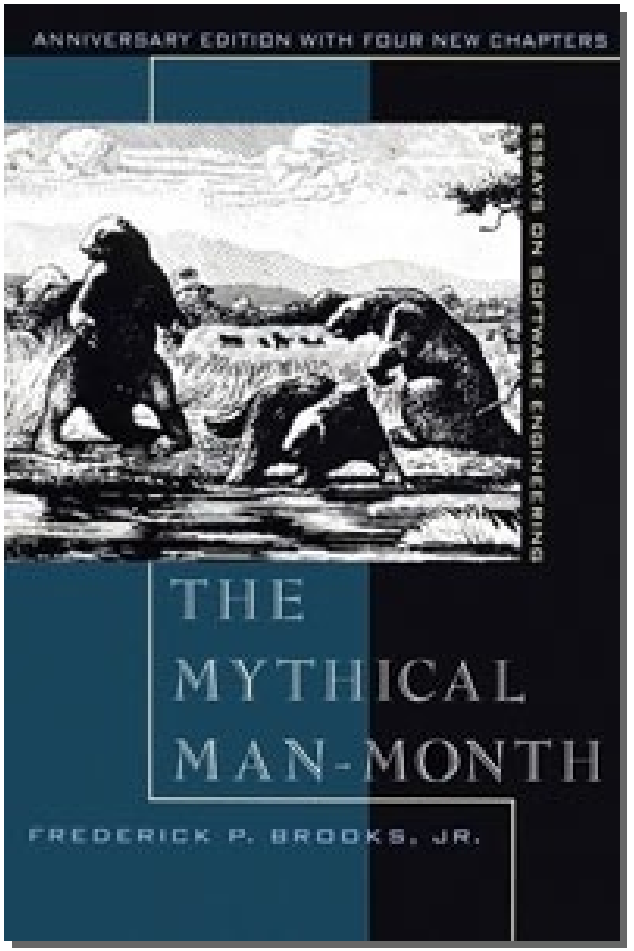
Project	Acceptable	Unvisited Functions	Function Coverage
Darjeeling :: vs-plugin v4 (NCover)	80.0 %	2470	36.7 %

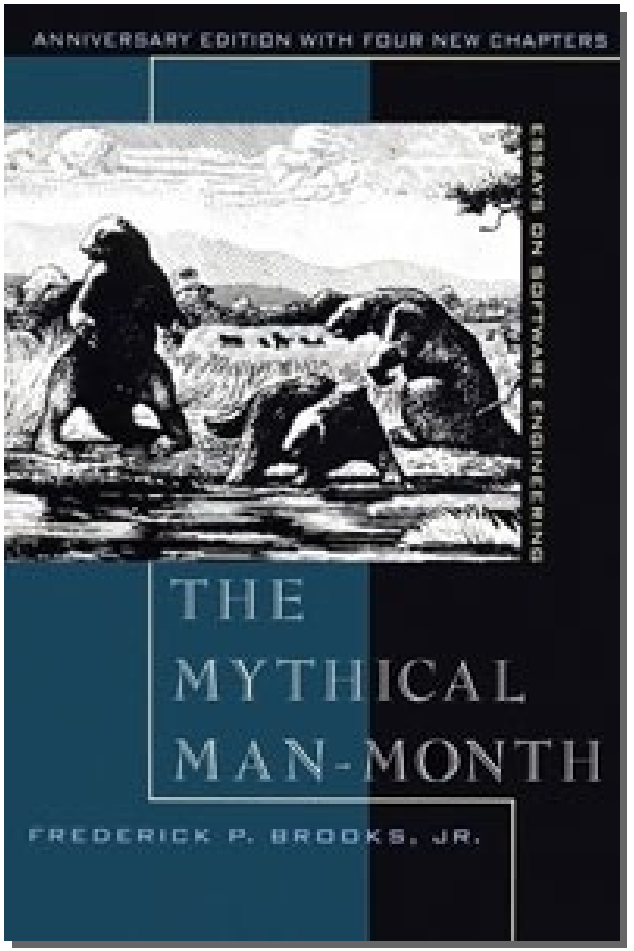
Modules	Acceptable	Unvisited Functions	Function Coverage
JetBrains.TeamCity.EventTrackers.dll	80.0 %	10	84.6 %
JetBrains.TeamCity.WebLinkListener.dll	80.0 %	11	76.6 %
JetBrains.TeamCity.Network.Login.dll	80.0 %	35	61.1 %
JetBrains.TeamCity.Common.dll	80.0 %	2	60.0 %
JetBrains.TeamCity.Connect.dll	80.0 %	214	51.6 %
JetBrains.TeamCity.SVN.dll	80.0 %	215	59.6 %
JetBrains.TeamCity.Perforce.dll	80.0 %	151	64.0 %
JetBrains.TeamCity.Network.Utils.dll	80.0 %	12	52.0 %
JetBrains.TeamCity.Utils.dll	80.0 %	533	33.2 %
JetBrains.TeamCity.TestsView.dll	80.0 %	142	16.5 %
JetBrains.TeamCity.Login.dll	80.0 %	35	16.7 %
JetBrains.TeamCity.RemoteRun.dll	80.0 %	797	15.3 %
JetBrains.TeamCity.Package.dll	80.0 %	313	3.4 %

Module	Acceptable	Unvisited Functions	Function Coverage
JetBrains.TeamCity.EventTrackers.dll	80.0 %	10	84.6 %
Namespace / Classes			
JetBrains.TeamCity.EventTrackers.Impl		9	85.5 %
PersonalChangesTrackerBase		0	100.0 %
ListenerInfo		0	100.0 %

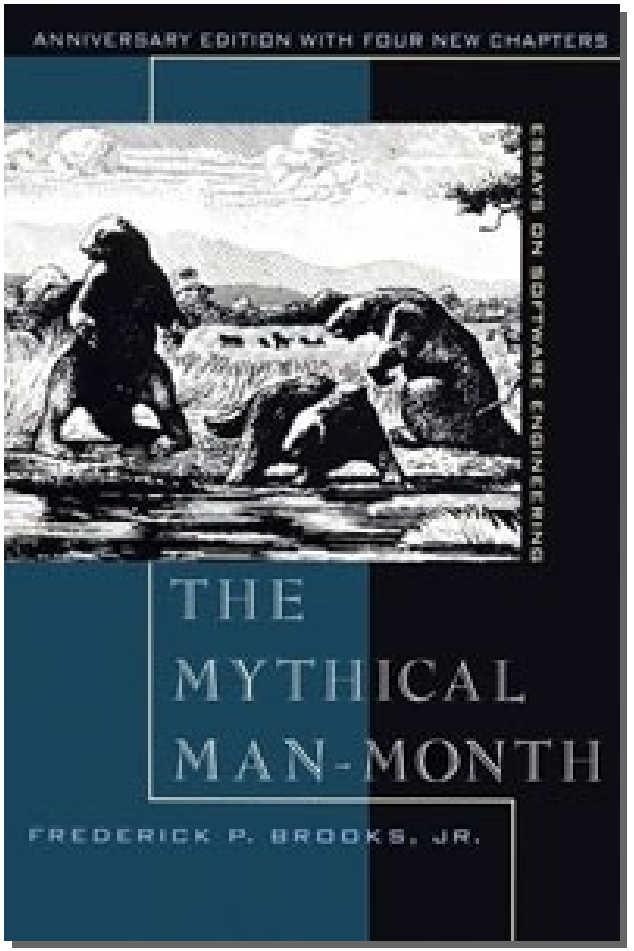

index
Zip
Current Darjeeling > vs-plugin v4 (NCover) > #5.0.119.0 (18 Feb 10 12:27)
Overview
Changes (1)
Tests
Build Log
Build Parameters
Dependencies
Artifacts
Code Coverage
NCoverExplorer Coverage Report - Darjeeling :: vs-plugin v4 (NCover)
Report generated on: Thu 18-Feb-2010 at 13:08:54
NCoverExplorer version: 1.3.6.36
Filtering / Sorting: None / CoveragePercentageDescending
Project Statistics: Files: 607 NCLOC: 16797
Classes: 869
Functions: 3901 Unvisited: 2470
Seq Pts: 15490 Unvisited: 10201
Total
api
class
component
controller
forms
helper
module
Project
Darjeeling :: vs-plugin v4 (
Modules
JetBrains.TeamCity.Event
JetBrains.TeamCity.WebLi
JetBrains.TeamCity.Netwo
JetBrains.TeamCity.Comm
JetBrains.TeamCity.Conne
JetBrains.TeamCity.SVN.d
JetBrains.TeamCity.Perfor
JetBrains.TeamCity.Netwo
JetBrains.TeamCity.Utils.d
JetBrains.TeamCity.Tests
JetBrains.TeamCity.Login.
JetBrains.TeamCity.Remot
JetBrains.TeamCity.Packa
Module
JetBrains.TeamCity.Event
Namespace / Classes
JetBrains.TeamCity.Ever
PersonalChangesTracke
ListenerInfo
Coverage Report
http://www.cafeconlech
Packages
All
org.jaxen
org.jaxen.dom
org.jaxen.dom.html
All Packages
Classes
BaseXPath (77%)
Context (93%)
ContextSupport (91
DefaultNavigator (3
DocumentNavigator
DOMXPath (100%)
DocumentNavigator
HTMLXPath (0%)
NamespaceNode (
DocumentNavigator
Dom4jXPath (100%
Coverage Report - All Packages
Package
Classes
Line Coverage
Branch Coverage
Complexity
All Packages 205 69% 80% 2.811
org.jaxen 24 77% 73% 1.38
org.jaxen.dom 3 55% 60% 1.907
org.jaxen.dom.html 2 0% 0% 1.364
org.jaxen.dom4j 2 78% 85% 2.395
org.jaxen.expr 73 73% 84% 1.566
org.jaxen.expr.iter 14 98% 100% 1.029
org.jaxen.function 27 64% 76% 5.373
org.jaxen.function.ext 6 63% 72% 4.235
org.jaxen.function.xslt 1 86% 100% 2.5
org.jaxen.javabean 4 44% 72% 1.87
org.jaxen.jdom 3 62% 63% 2.897
org.jaxen.pattern 13 49% 52% 2.135
org.jaxen.saxpath 8 51% 81% 1.887
org.jaxen.saxpath.base 6 95% 100% 10.723
org.jaxen.saxpath.helpers 2 28% 83% 1.34
org.jaxen.util 15 41% 50% 2.432
org.jaxen.xom 2 71% 66% 1.783
Reports generated by Cobertura.
Done
Disabled

Thou shalt not have less than 80% coverage


ANNIVERSARY EDITION WITH FOUR NEW CHAPTERS
ESSAYS ON SOFTWARE ENGINEERING
THE MYTHICAL MAN-MONTH
FREDERICK P. BROOKS, JR.


ANNIVERSARY EDITION WITH FOUR NEW CHAPTERS
ESSAYS ON SOFTWARE ENGINEERING
THE MYTHICAL MAN-MONTH
FREDERICK P. BROOKS, JR.

Brooks, 1975



50%

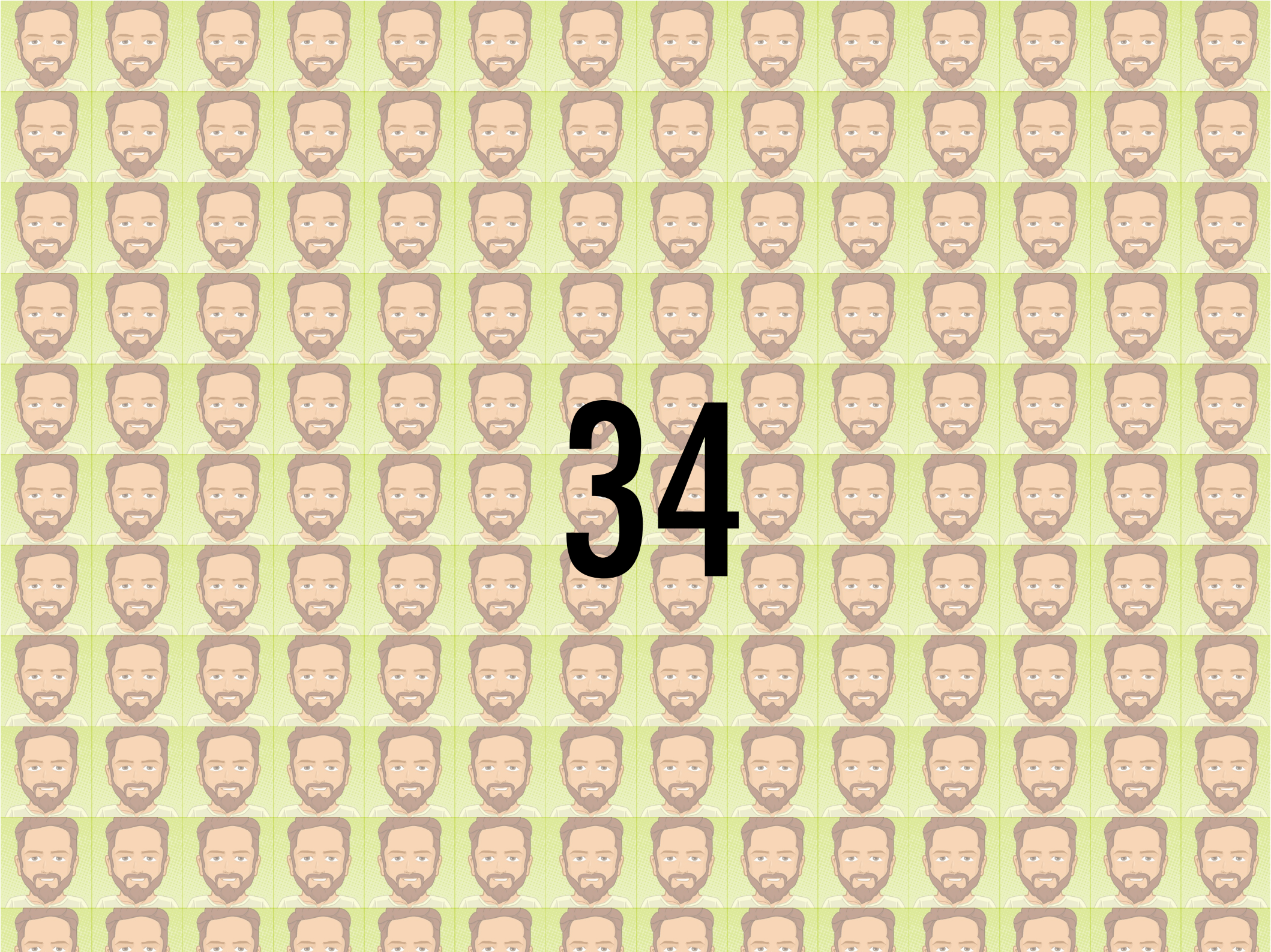
… of Project Effort

Brooks, 1975


PRESENT

How (Much) Do *You* Test?

WatchDog


34

How (Much) Do Developers Test?

Moritz Beller, Georgios Gousios, Andy Zaidman
Delft University of Technology,
The Netherlands
{m.m.beller, g.gousios, a.e.zaidman}@tudelft.nl

***Abstract*—What do we know about software testing in the real world? It seems we know from Fred Brooks' seminal work "The Mythical Man-Month" that 50% of project effort is spent on testing. However, due to the enormous advances in software engineering in the past 40 years, the question stands: Is this observation still true? In fact, was it ever true? The vision for our research is to settle the discussion about Brooks' estimation once and for all: How much do developers test? Does developers' estimation on how much they test match reality? How frequently do they execute their tests, and is there a relationship between test runtime and execution frequency? What are the typical reactions to failing tests? Do developers solve actual defects in the production code, or do they merely relax their test assertions? Emerging results from 40 software engineering students show that students overestimate their testing time threefold, and 50% of them test as little as 4% of their time, or less. Having proven the scalability of our infrastructure, we are now extending our case study with professional software engineers from open-source and industrial organizations.**

I. Introduction

Understanding how developers test and how to better support them in practice is crucial for the design of next-generation Integrated Development Environments (*IDEs*) and testing tools. Important questions, that need to be answered to increase our understanding, are:

1) How much time is spent on engineering test code versus production code? Do developers' estimation on how much they test match reality?
2) How frequently do developers execute their tests? Do they do it after each change to production code, or only a few times a day?
3) How long does a test run executed in the IDE take? Is there a relation between the frequency and the length of execution?
4) What are the typical reactions to failing tests? Do developers solve actual defects in the production code, or do they merely relax their test assertions?

In his seminal work on the mythical man-month from 1975, Brooks estimates that 50% of the development time of a software product is dedicated to testing [1]. In the 40 years since, software engineering has changed dramatically. New programming languages, intelligent IDEs, a more agile way of working and test-driven development (*TDD*) are but a few of these advances. In general, practitioners and researchers have gained a broader understanding of the importance and benefits of software testing [2], expressed in its wide adoption in practice [2], [3].

Despite these fundamental changes, however, there is a surprising absence of research that follows up on Brooks' estimation and examines how much time developers devote to testing and production code. Even in recent literature, Brooks' rough 50% estimate is still widely referenced [4], [5]. It comes as no surprise then that the question of how much time needs to be spent on testing is one of the grand research challenges in empirical software engineering [6].

As developer testing, we understand any activity the developer undertakes in his IDE related to testing the program [7]. This usually consists of writing and executing unit tests, but is widely complemented by integration or system testing [8]. On a project level, developer testing is often complemented by manual testing, dedicated test teams and automated test generation. In contrast to these quality assurance methods, testing in the IDE cannot be quantified with a traditional time measurement approach, as it is intertwined with developing production code, especially when using TDD.

By observing the fine-grained steps in which developers construct software in their IDEs, we are able to provide deep insights into the aforementioned questions. Our approach contrasts the traditional repository mining, which focuses on the final result of fine-grained developer activities in the IDE, but does not provide accurate timing information about them.

In this paper, we present a novel approach to tackle our empirical questions on developer testing in a large-scale case study and report its first emerging results. To this end, we have created the WatchDog project.[1] While the data collection from developers across several open-source and commercial organizations is ongoing, student data from a software engineering course at TU Delft gives interesting first results: It show that students use on average only 9% of their working time in the IDE for engineering tests. Moreover, they test three times less in reality than they think they do. In our ongoing work, we are complementing this study with a study on professional developers from commercial and open-source projects, and diving deeper into the reactions of individual developers to failing test cases.

II. Methods

To study our research questions, we could watch over the shoulders of developers and manually take notes on how they develop their software. However, when we want to perform the study on a larger scale, we need an automated way to reliably

[1] http://www.testroots.org/testroots_watchdog.html

How (Much) Do Developers Test?

Moritz Beller, Georgios Gousios, Andy Zaidman
Delft University of Technology,
The Netherlands
{m.m.beller, g.gousios, a.e.zaidman}@tudelft.nl

our research is to settle the discussion about Brooks' estimation once and for all: How much do developers test? Does developers' estimation on how much they test match reality? How frequently do they execute their tests, and is there a relationship between test runtime and execution frequency? What are the typical reactions to failing tests? Do developers solve actual defects in the production code, or do they merely relax their test assertions? Emerging results from 40 software engineering students show that students overestimate their testing time threefold, and 50% of them test as little as 4% of their time, or less. Having proven the scalability of our infrastructure, we are now extending our case study with professional software engineers from open-source and industrial organizations.

I. Introduction

Understanding how developers test and how to better support them in practice is crucial for the design of next-generation Integrated Development Environments (*IDEs*) and testing tools. Important questions, that need to be answered to increase our understanding, are:

1) How much time is spent on engineering test code versus production code? Do developers' estimation on how much they test match reality?
2) How frequently do developers execute their tests? Do they do it after each change to production code, or only a few times a day?
3) How long does a test run executed in the IDE take? Is there a relation between the frequency and the length of execution?
4) What are the typical reactions to failing tests? Do developers solve actual defects in the production code, or do they merely relax their test assertions?

In his seminal work on the mythical man-month from 1975, Brooks estimates that 50% of the development time of a software product is dedicated to testing [1]. In the 40 years since, software engineering has changed dramatically. New programming languages, intelligent IDEs, a more agile way of working and test-driven development (*TDD*) are but a few of these advances. In general, practitioners and researchers have gained a broader understanding of the importance and benefits of software testing [2], expressed in its wide adoption in practice [2], [3]. as no surprise then that the question of how much time needs to be spent on testing is one of the grand research challenges in empirical software engineering [6].

As developer testing, we understand any activity the developer undertakes in his IDE related to testing the program [7]. This usually consists of writing and executing unit tests, but is widely complemented by integration or system testing [8]. On a project level, developer testing is often complemented by manual testing, dedicated test teams and automated test generation. In contrast to these quality assurance methods, testing in the IDE cannot be quantified with a traditional time measurement approach, as it is intertwined with developing production code, especially when using TDD.

By observing the fine-grained steps in which developers construct software in their IDEs, we are able to provide deep insights into the aforementioned questions. Our approach contrasts the traditional repository mining, which focuses on the final result of fine-grained developer activities in the IDE, but does not provide accurate timing information about them.

In this paper, we present a novel approach to tackle our empirical questions on developer testing in a large-scale case study and report its first emerging results. To this end, we have created the WatchDog project.[1] While the data collection from developers across several open-source and commercial organizations is ongoing, student data from a software engineering course at TU Delft gives interesting first results: It show that students use on average only 9% of their working time in the IDE for engineering tests. Moreover, they test three times less in reality than they think they do. In our ongoing work, we are complementing this study with a study on professional developers from commercial and open-source projects, and diving deeper into the reactions of individual developers to failing test cases.

II. Methods

To study our research questions, we could watch over the shoulders of developers and manually take notes on how they develop their software. However, when we want to perform the study on a larger scale, we need an automated way to reliably

[1] http://www.testroots.org/testroots_watchdog.html

How (Much) Do ~~Developers~~ Test?

Moritz Beller, Georgios Gousios, Andy Zaidman
Delft University of Technology,
The Netherlands
{m.m.beller, g.gousios, a.e.zaidman}@tudelft.nl

our research is to settle the discussion about Brooks' estimation once and for all: How much do developers test? Does developers' estimation on how much they test match reality? How frequently do they execute their tests, and is there a relationship between test runtime and execution frequency? What are the typical reactions to failing tests? Do developers solve actual defects in the production code, or do they merely relax their test assertions? Emerging results from 40 software engineering students show that students overestimate their testing time threefold, and 50% of them test as little as 4% of their time, or less. Having proven the scalability of our infrastructure, we are now extending our case study with professional software engineers from open-source and industrial organizations.

I. Introduction

Understanding how developers test and how to better support them in practice is crucial for the design of next-generation Integrated Development Environments (*IDEs*) and testing tools. Important questions, that need to be answered to increase our understanding, are:

1) How much time is spent on engineering test code versus production code? Do developers' estimation on how much they test match reality?
2) How frequently do developers execute their tests? Do they do it after each change to production code, or only a few times a day?
3) How long does a test run executed in the IDE take? Is there a relation between the frequency and the length of execution?
4) What are the typical reactions to failing tests? Do developers solve actual defects in the production code, or do they merely relax their test assertions?

In his seminal work on the mythical man-month from 1975, Brooks estimates that 50% of the development time of a software product is dedicated to testing [1]. In the 40 years since, software engineering has changed dramatically. New programming languages, intelligent IDEs, a more agile way of working and test-driven development (*TDD*) are but a few of these advances. In general, practitioners and researchers have gained a broader understanding of the importance and benefits of software testing [2], expressed in its wide adoption in practice [2], [3]. as no surprise then that the question of how much time needs to be spent on testing is one of the grand research challenges in empirical software engineering [6].

As developer testing, we understand any activity the developer undertakes in his IDE related to testing the program [7]. This usually consists of writing and executing unit tests, but is widely complemented by integration or system testing [8]. On a project level, developer testing is often complemented by manual testing, dedicated test teams and automated test generation. In contrast to these quality assurance methods, testing in the IDE cannot be quantified with a traditional time measurement approach, as it is intertwined with developing production code, especially when using TDD.

By observing the fine-grained steps in which developers construct software in their IDEs, we are able to provide deep insights into the aforementioned questions. Our approach contrasts the traditional repository mining, which focuses on the final result of fine-grained developer activities in the IDE, but does not provide accurate timing information about them.

In this paper, we present a novel approach to tackle our empirical questions on developer testing in a large-scale case study and report its first emerging results. To this end, we have created the WatchDog project.[1] While the data collection from developers across several open-source and commercial organizations is ongoing, student data from a software engineering course at TU Delft gives interesting first results: It show that students use on average only 9% of their working time in the IDE for engineering tests. Moreover, they test three times less in reality than they think they do. In our ongoing work, we are complementing this study with a study on professional developers from commercial and open-source projects, and diving deeper into the reactions of individual developers to failing test cases.

II. Methods

To study our research questions, we could watch over the shoulders of developers and manually take notes on how they develop their software. However, when we want to perform the study on a larger scale, we need an automated way to reliably

[1] http://www.testroots.org/testroots_watchdog.html

How (Much) Do ~~Developers~~ Students Test?

Moritz Beller, Georgios Gousios, Andy Zaidman
Delft University of Technology,
The Netherlands
{m.m.beller, g.gousios, a.e.zaidman}@tudelft.nl

our research is to settle the discussion about Brooks' estimation once and for all: How much do developers test? Does developers' estimation on how much they test match reality? How frequently do they execute their tests, and is there a relationship between test runtime and execution frequency? What are the typical reactions to failing tests? Do developers solve actual defects in the production code, or do they merely relax their test assertions? Emerging results from 40 software engineering students show that students overestimate their testing time threefold, and 50% of them test as little as 4% of their time, or less. Having proven the scalability of our infrastructure, we are now extending our case study with professional software engineers from open-source and industrial organizations.

I. Introduction

Understanding how developers test and how to better support them in practice is crucial for the design of next-generation Integrated Development Environments (*IDEs*) and testing tools. Important questions, that need to be answered to increase our understanding, are:

1) How much time is spent on engineering test code versus production code? Do developers' estimation on how much they test match reality?
2) How frequently do developers execute their tests? Do they do it after each change to production code, or only a few times a day?
3) How long does a test run executed in the IDE take? Is there a relation between the frequency and the length of execution?
4) What are the typical reactions to failing tests? Do developers solve actual defects in the production code, or do they merely relax their test assertions?

In his seminal work on the mythical man-month from 1975, Brooks estimates that 50% of the development time of a software product is dedicated to testing [1]. In the 40 years since, software engineering has changed dramatically. New programming languages, intelligent IDEs, a more agile way of working and test-driven development (*TDD*) are but a few of these advances. In general, practitioners and researchers have gained a broader understanding of the importance and benefits of software testing [2], expressed in its wide adoption in practice [2], [3]. as no surprise then that the question of how much time needs to be spent on testing is one of the grand research challenges in empirical software engineering [6].

As developer testing, we understand any activity the developer undertakes in his IDE related to testing the program [7]. This usually consists of writing and executing unit tests, but is widely complemented by integration or system testing [8]. On a project level, developer testing is often complemented by manual testing, dedicated test teams and automated test generation. In contrast to these quality assurance methods, testing in the IDE cannot be quantified with a traditional time measurement approach, as it is intertwined with developing production code, especially when using TDD.

By observing the fine-grained steps in which developers construct software in their IDEs, we are able to provide deep insights into the aforementioned questions. Our approach contrasts the traditional repository mining, which focuses on the final result of fine-grained developer activities in the IDE, but does not provide accurate timing information about them.

In this paper, we present a novel approach to tackle our empirical questions on developer testing in a large-scale case study and report its first emerging results. To this end, we have created the WatchDog project.[1] While the data collection from developers across several open-source and commercial organizations is ongoing, student data from a software engineering course at TU Delft gives interesting first results: It show that students use on average only 9% of their working time in the IDE for engineering tests. Moreover, they test three times less in reality than they think they do. In our ongoing work, we are complementing this study with a study on professional developers from commercial and open-source projects, and diving deeper into the reactions of individual developers to failing test cases.

II. Methods

To study our research questions, we could watch over the shoulders of developers and manually take notes on how they develop their software. However, when we want to perform the study on a larger scale, we need an automated way to reliably

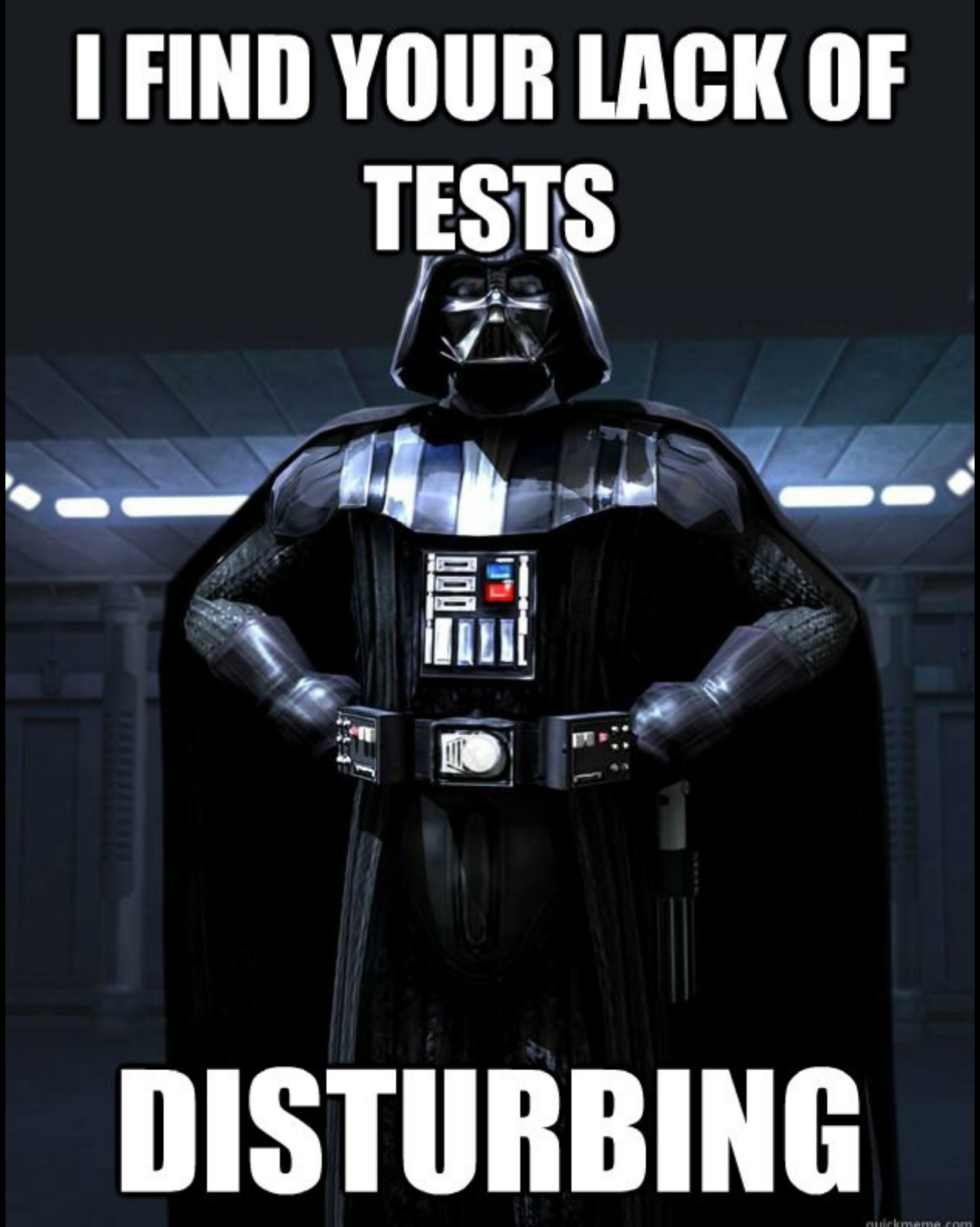
[1] http://www.testroots.org/testroots_watchdog.html

4.2 yr

OMG!!!

Histogram of Testing Time

50% of students test only 4% of their time, or less


I FIND YOUR LACK OF TESTS
DISTURBING

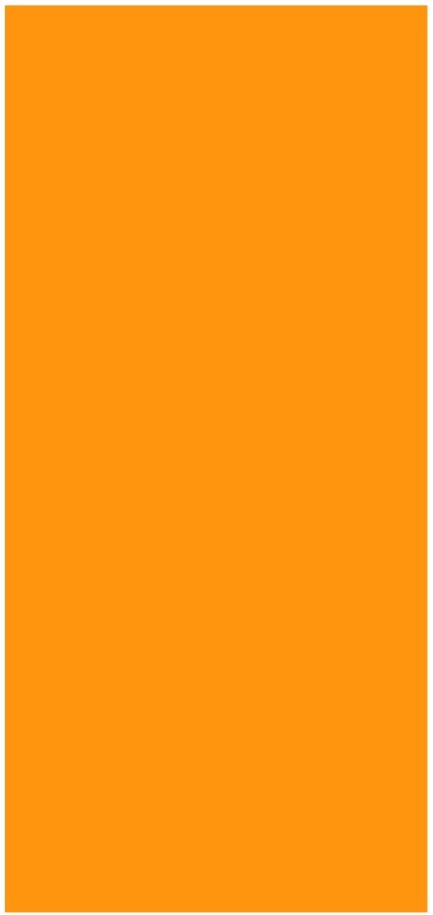
Thou shalt not have less than 75% coverage

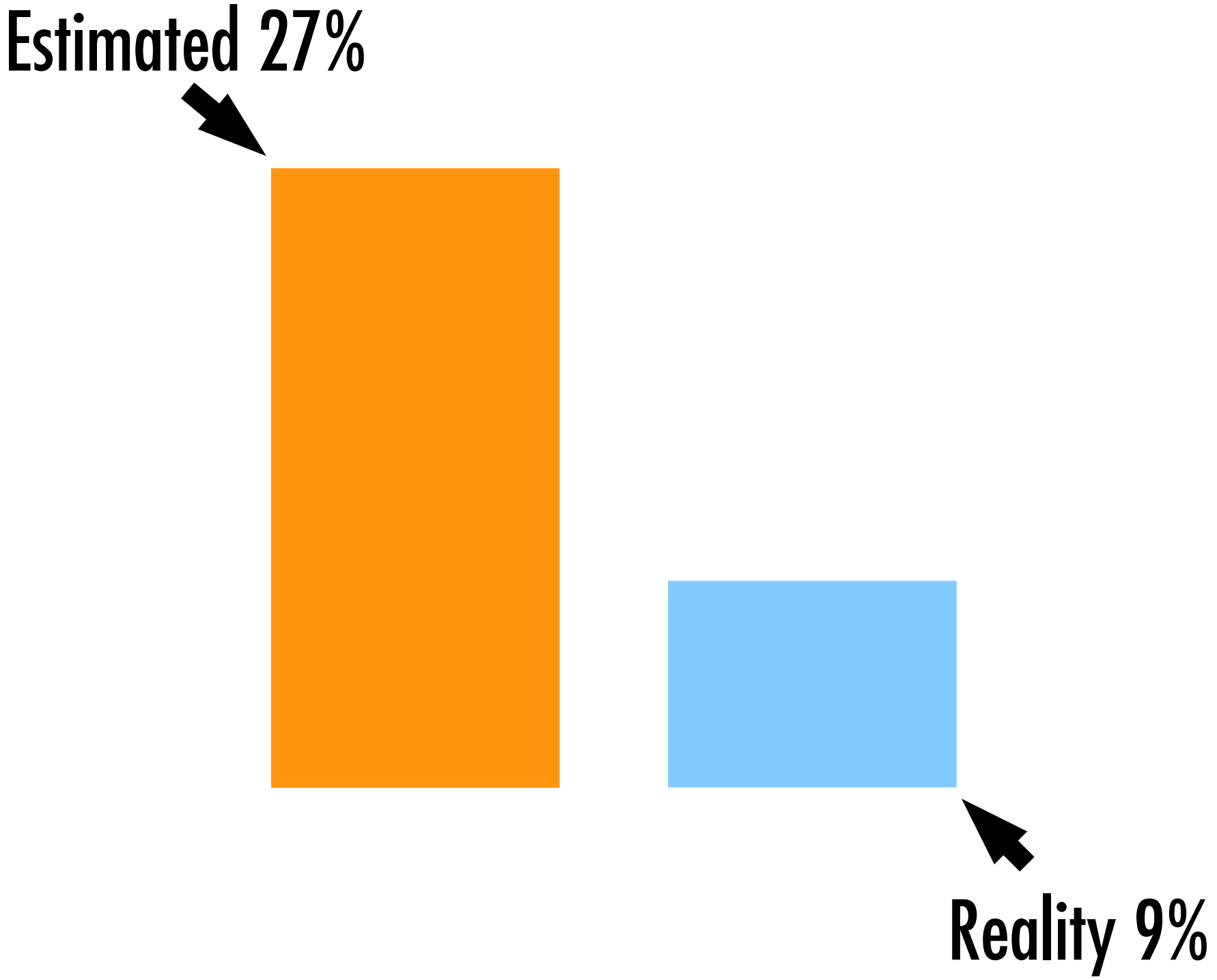
~~Thou~~ shalt not

Students have less than

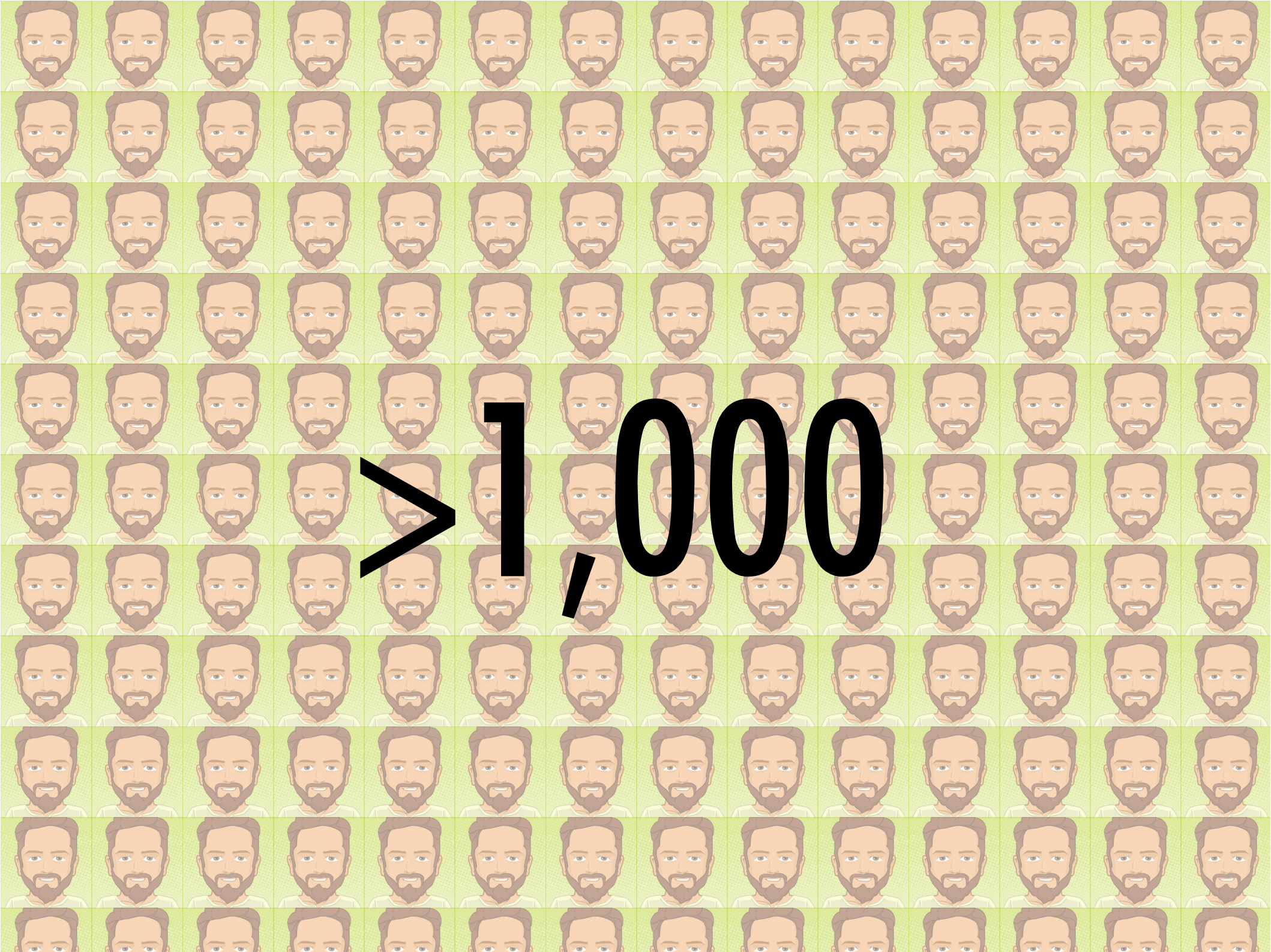
75% coverage

Estimated 27%

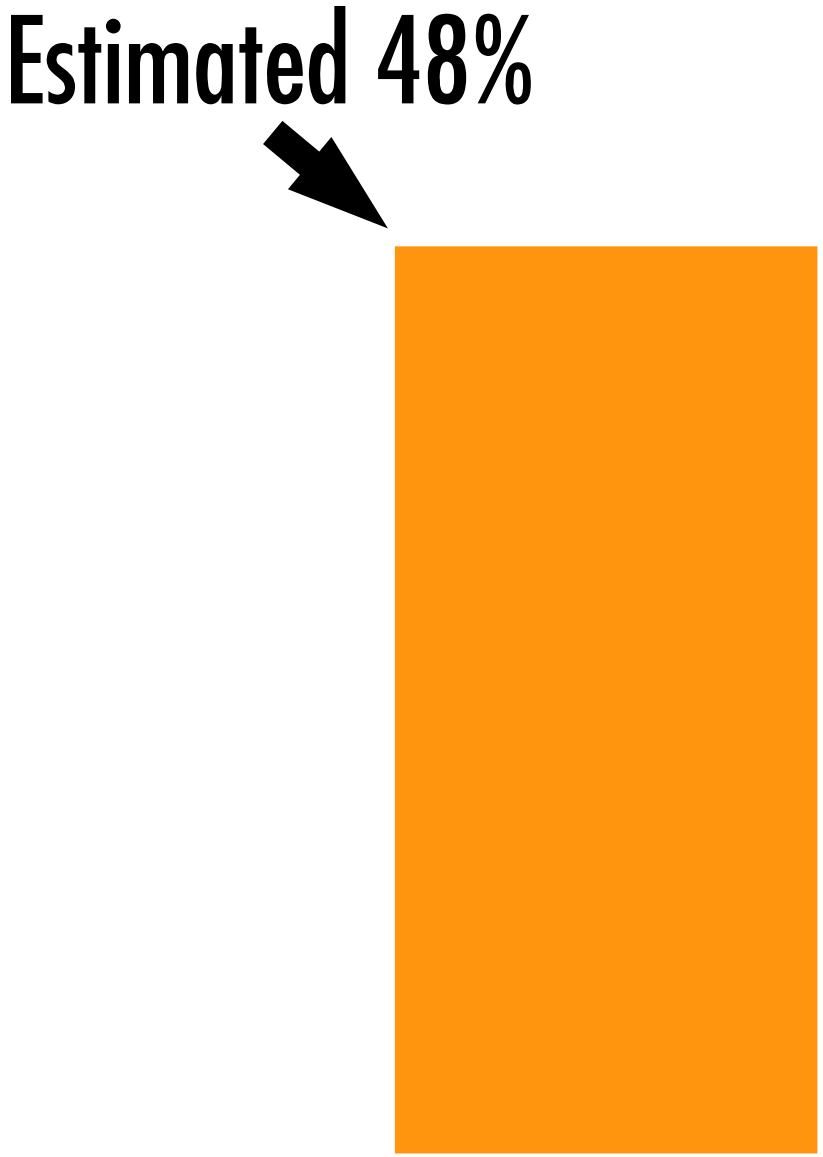


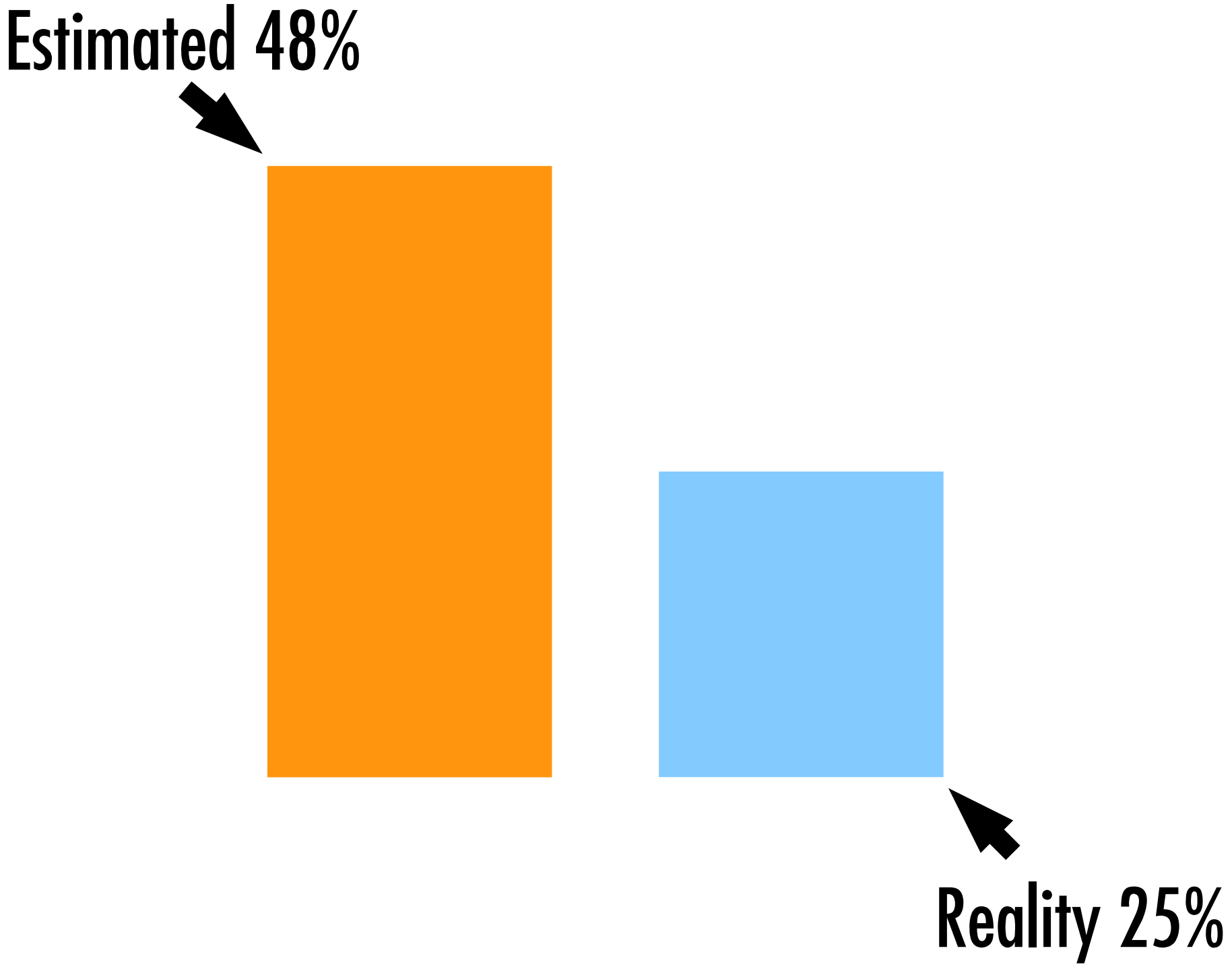

Estimated 27%
Reality 9%


iMac

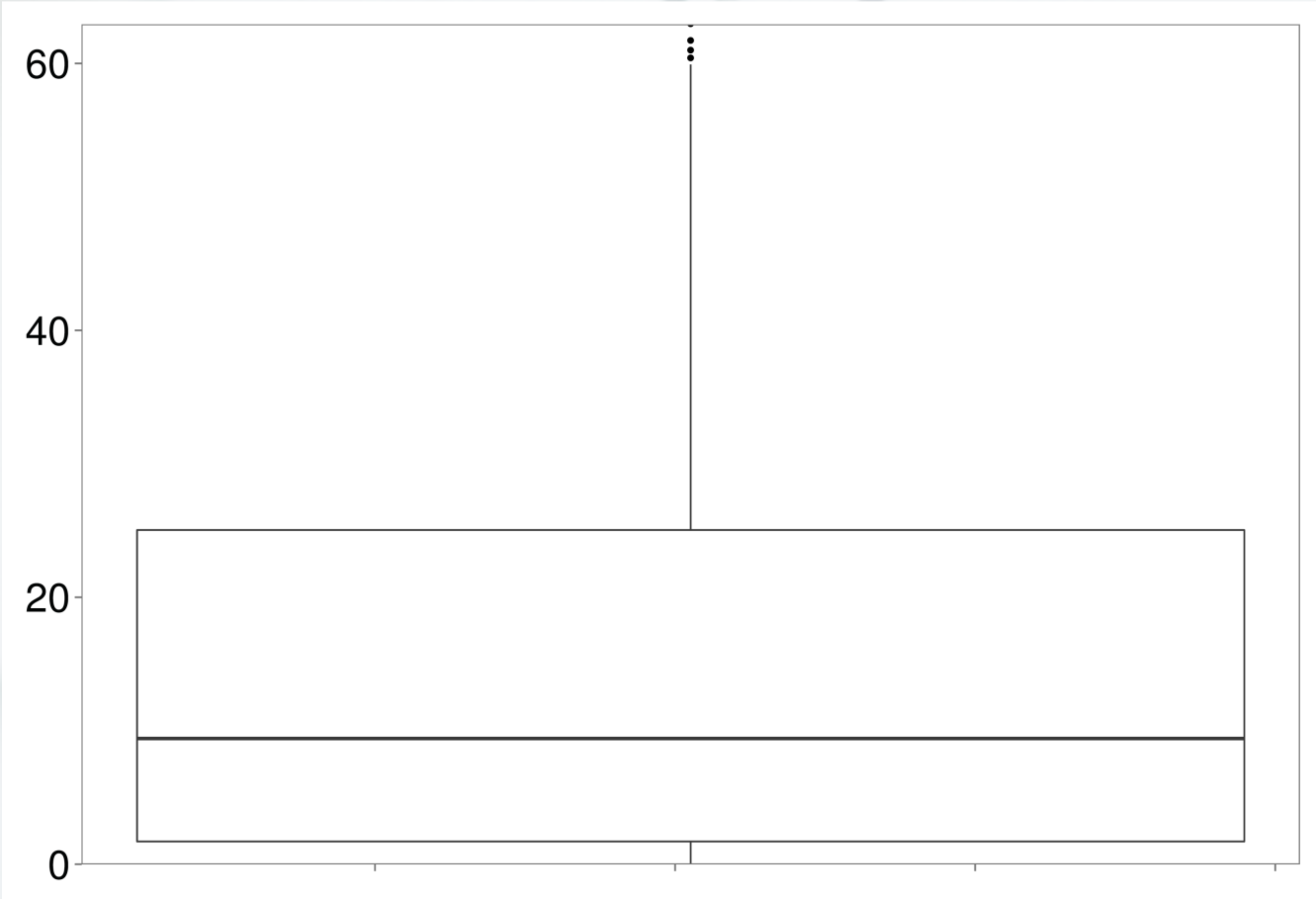

>1,000

13 yr

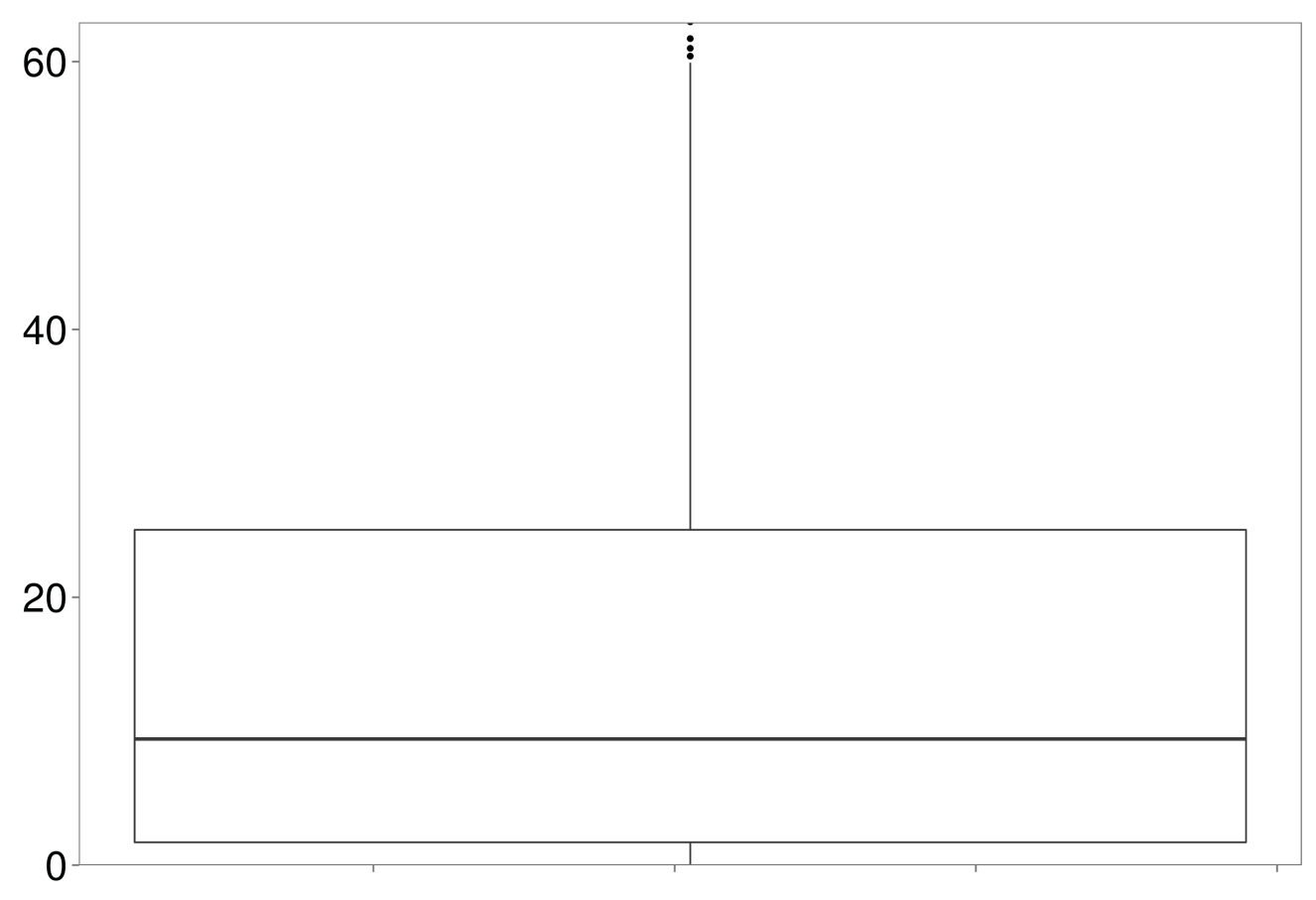

Estimated 48%


Estimated 48%
Reality 25%

65% of test executions fail.


60
40
20
0

50% of test failures are fixed within 10 minutes.



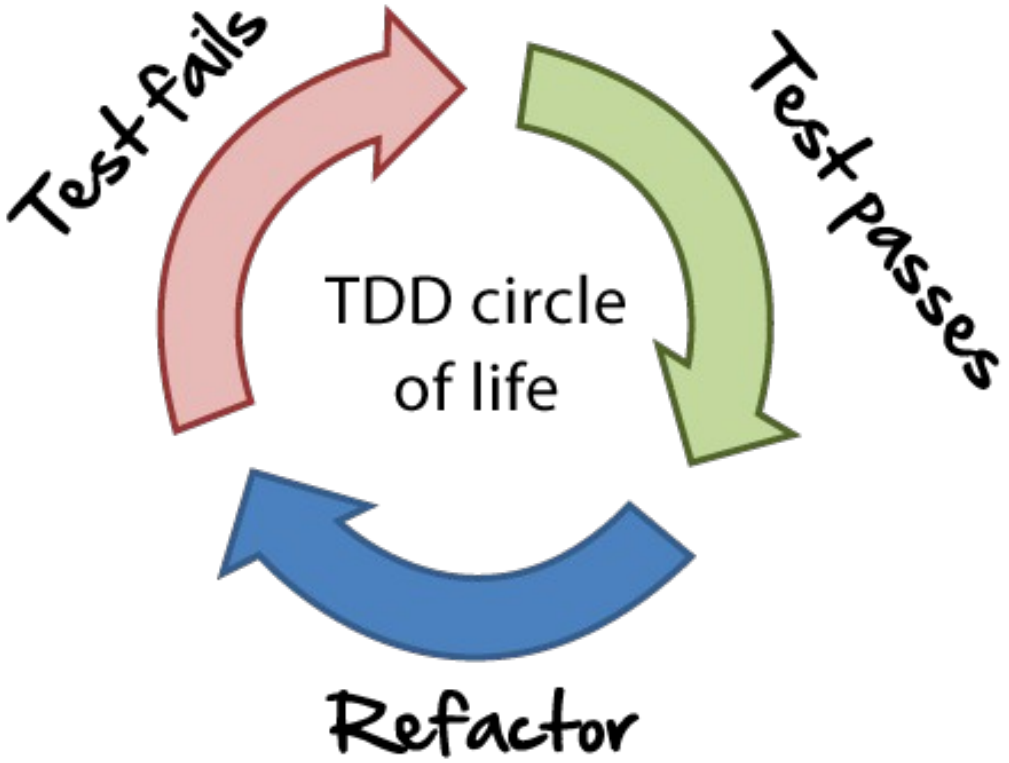


Image source: http://www.agilenutshell.com/test_driven_development

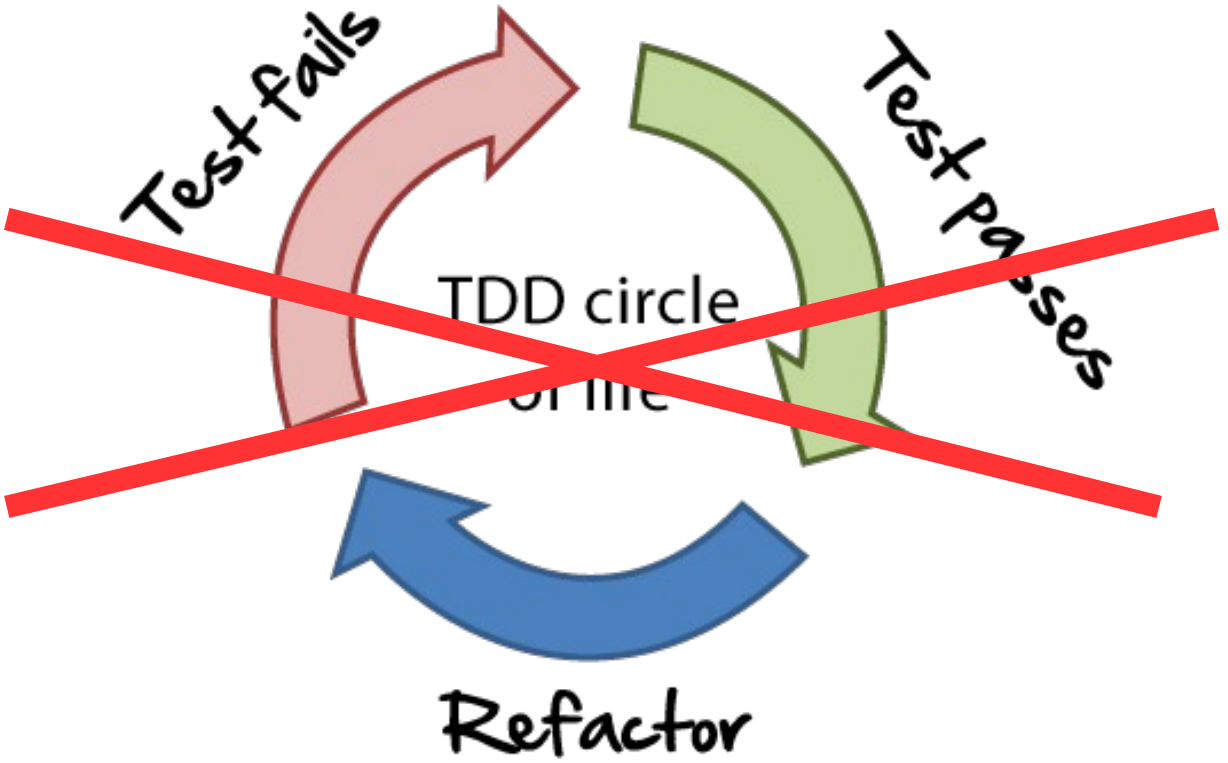
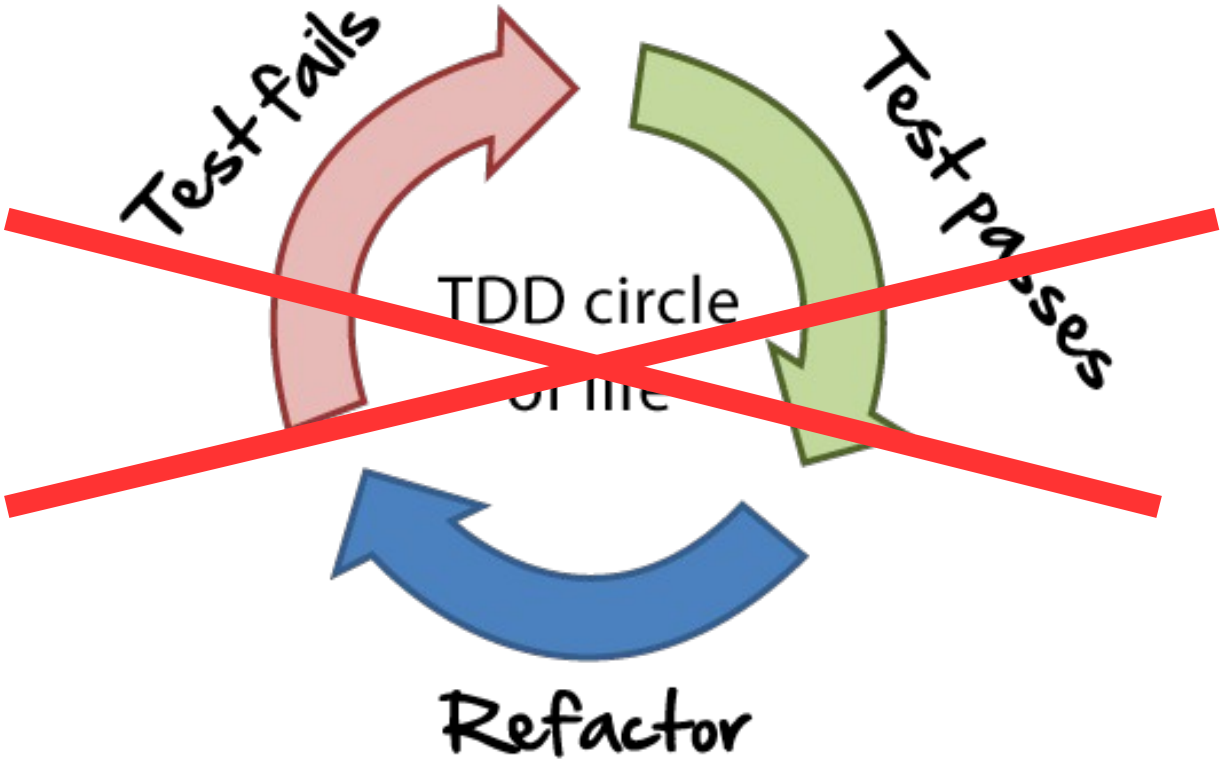


Image source: http://www.agilenutshell.com/test_driven_development



Nobody follows TDD (strictly).

Image source: http://www.agilenutshell.com/test_driven_development


FUTURE

WatchDog

DO YOU PROGRAM JAVA WITH ECLIPSE?

YOU CAN WIN AMAZING PRIZES WITH ZERO EXTRA WORK, AND HELP SCIENCE ALONG THE WAY. DOWNLOAD

WatchDog is an Eclipse plugin that monitors how you test, and it respects your privacy (and your company's policy).

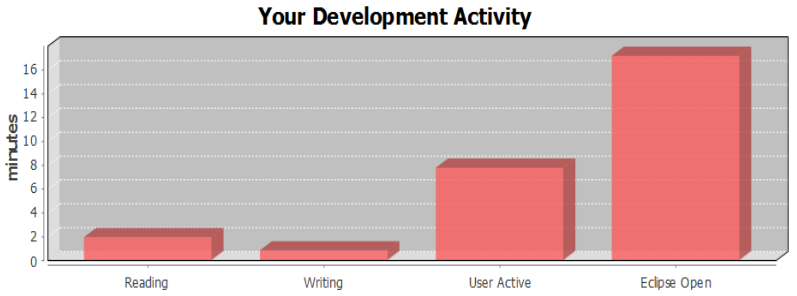
I want to install WatchDog!

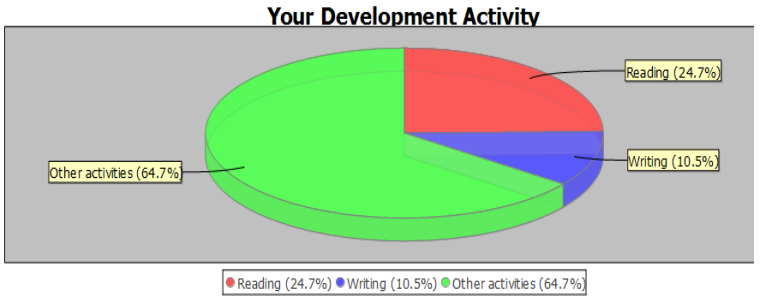
Fire up your Eclipse (we support 3.7, 3.8, 4.2, 4.3 and 4.4) and simply drag the install button into your Eclipse

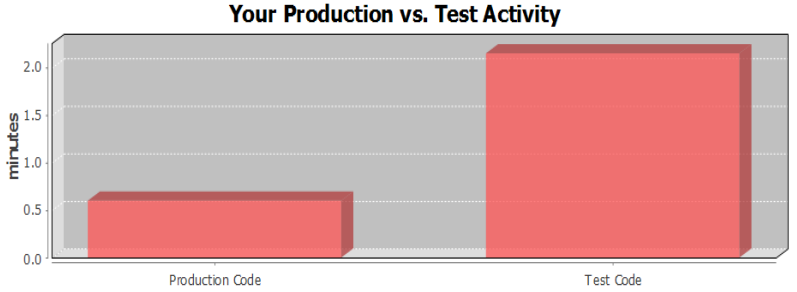
Install

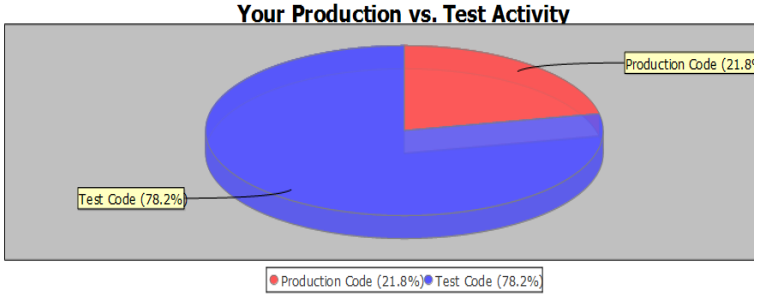
WatchDog

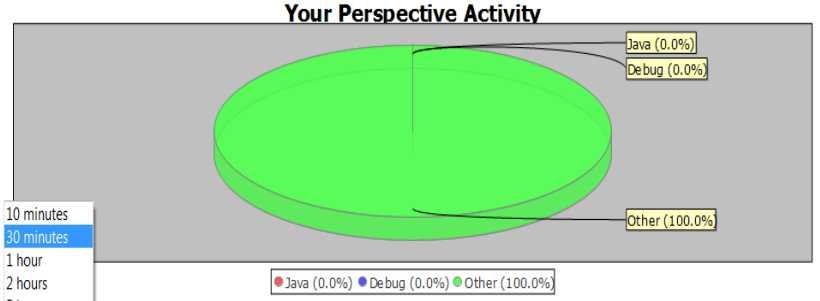
WD WatchDog Statistics

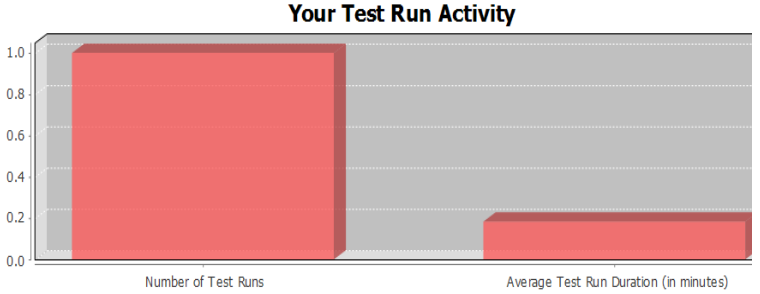












10 minutes
30 minutes
1 hour
2 hours
5 hours
8 hours
10 hours

2 11:07:22 CET 2015 to Mon Jan 12 11:34:53 CET 2015.

ervals.)

30 minutes

Refresh.

Report on Your Project "WatchDog"

This is the WatchDog Report based on your data submitted for your project "WatchDog" under the project id fe7982626ebea0c754b330a4be223f3cf3d45d3c.

Summary of Your Project "WatchDog"

You worked 17% of your time on testing, and 83% on production code (you originally estimated 30% testing, 70% production).
You executed your tests on average 21 times per day, and the average execution time for one such test execution was 5.4 seconds. 22% of your tests failed.
You do not follow test-driven development.

In the following, you can find more detailed statistics on your project.

Description	Your value	Average Ø	Ø in Peers
Total time in which WatchDog was active	**407h**	45h	351h
This makes an average of (assuming a 5-day work week)	**4h/day**	1h/day	5h/day

General Development Behavior

Active Eclipse Usage	**75%**	40%	51%

This number tells how long you actively worked in Eclipse while it was open. It can be indicative of how much you can focus on programming during your work, but many factors influence it: If you need to do a lot of work outside of your IDE, leave your IDE open while doing on other stuff, or leave your IDE running while attending a meeting, for example, this number is small.

Time spent Writing	**10%***	5%	7%

This means how much of your active time you were modifying text documents (Java, XML, ...) in Eclipse. A high number means you write a lot of text in your IDE.

Time spent Reading	**55%***	40%	42%

This means how much of your active time you were reading text documents (Java, XML, ...) in Eclipse. A high number means you read a lot of text in your IDE.

Remaining Time	**35%***	55%	51%

This can be accessing non-purely text-based files, performing refactorings, configuring Eclipse, or similar work not done in Eclipse's text editor.

* Of your Active Eclipse Usage.

WatchDog

WatchDog


WatchDog


eclipse

WatchDog


PAST
PRESENT
FUTURE

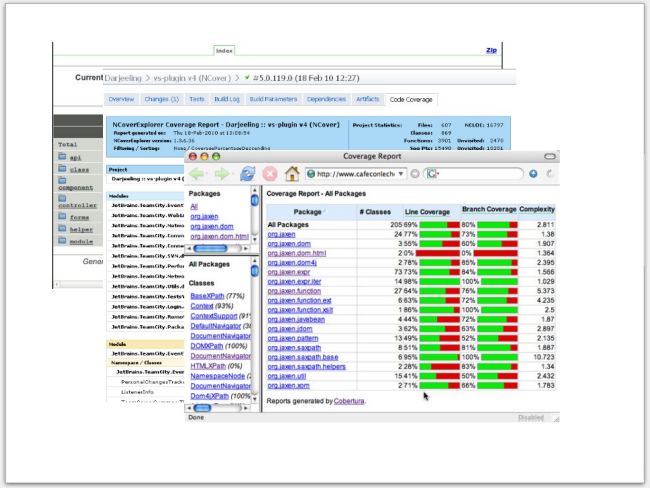
How (Much) Do Developers Test?

TestRoots.org

@Inventitech
Moritz Beller, TU Delft

How (Much) Do Developers Test?

TestRoots.org

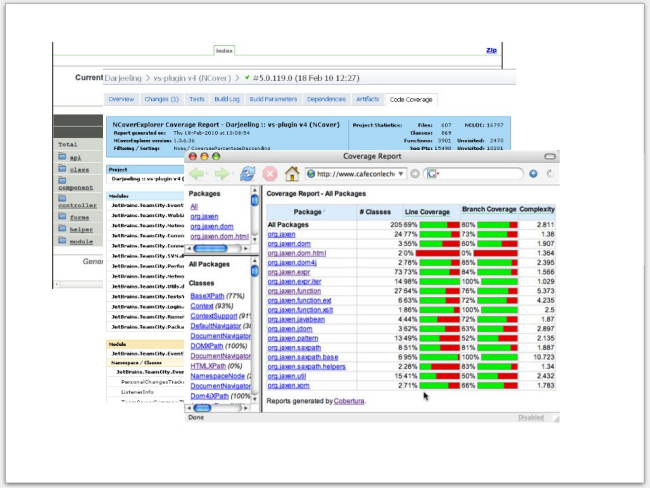


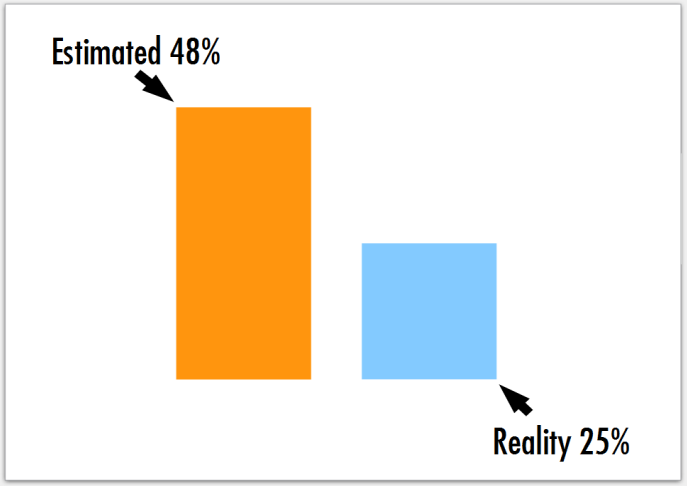
@Inventitech

Moritz Beller, TU Delft

How (Much) Do Developers Test?

TestRoots.org



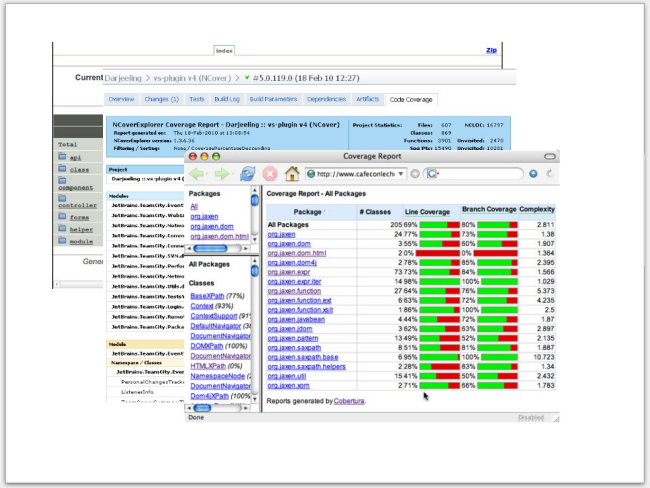


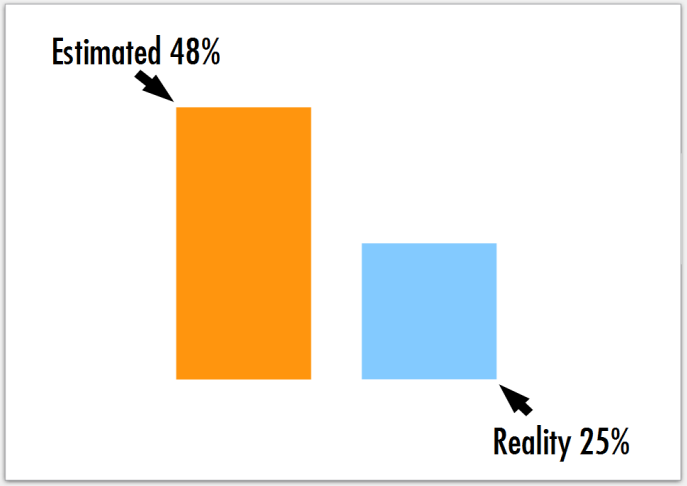
@Inventitech

Moritz Beller, TU Delft

How (Much) Do Developers Test?

TestRoots.org





@Inventitech

Moritz Beller, TU Delft