The University Of Sheffield.

AUTOMATED UNIT TEST GENERATION DURING SOFTWARE DEVELOPMENT

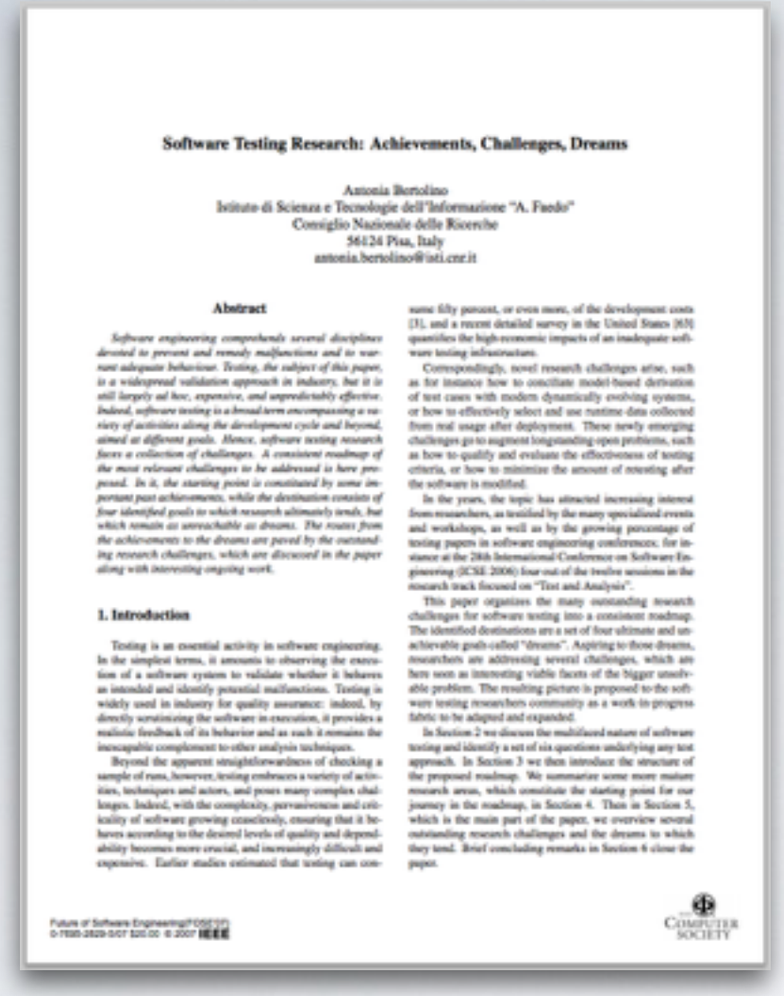
A Controlled Experiment and Think-aloud Observations

ISSTA 2015

José Miguel Rojas

j.rojas@sheffield.ac.uk

Joint work with Gordon Fraser and Andrea Arcuri



Software Testing Research: Achievements, Challenges, Dreams

Antonia Bertolino
Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"
Consiglio Nazionale delle Ricerche
56124 Pisa, Italy
antonia.bertolino@isti.cnr.it

Abstract

Software engineering comprehends several disciplines devoted to prevent and remedy malfunctions and to warrant adequate behaviour. Testing, the subject of this paper, is a widespread validation approach in industry, but it is still largely ad hoc, expensive, and unpredictably effective. Indeed, software testing is a broad term encompassing a variety of activities along the development cycle and beyond, aimed at different goals. Hence, software testing research faces a collection of challenges. A consistent roadmap of the most relevant challenges to be addressed is here proposed. In it, the starting point is constituted by some important past achievements, while the destination consists of four identified goals to which research ultimately tends, but which remain as unreachable as dreams. The routes from the achievements to the dreams are paved by the outstanding research challenges, which are discussed in the paper along with interesting ongoing work.

1. Introduction

Testing is an essential activity in software engineering. In the simplest terms, it amounts to observing the execution of a software system to validate whether it behaves as intended and identify potential malfunctions. Testing is widely used in industry for quality assurance: indeed, by directly scrutinizing the software in execution, it provides a realistic feedback of its behavior and as such it remains the inescapable complement to other analysis techniques.

Beyond the apparent straightforwardness of checking a sample of runs, however, testing embraces a variety of activities, techniques and actors, and poses many complex challenges. Indeed, with the complexity, pervasiveness and criticality of software growing ceaselessly, ensuring that it behaves according to the desired levels of quality and dependability becomes more crucial, and increasingly difficult and expensive. Earlier studies estimated that testing can consume fifty percent, or even more, of the development costs [3], and a recent detailed survey in the United States [63] quantifies the high economic impacts of an inadequate software testing infrastructure.

Correspondingly, novel research challenges arise, such as for instance how to conciliate model-based derivation of test cases with modern dynamically evolving systems, or how to effectively select and use runtime data collected from real usage after deployment. These newly emerging challenges go to augment longstanding open problems, such as how to qualify and evaluate the effectiveness of testing criteria, or how to minimize the amount of retesting after the software is modified.

In the years, the topic has attracted increasing interest from researchers, as testified by the many specialised events and workshops, as well as by the growing percentage of testing papers in software engineering conferences; for instance at the 28th International Conference on Software Engineering (ICSE 2006) four out of the twelve sessions in the research track focused on "Test and Analysis".

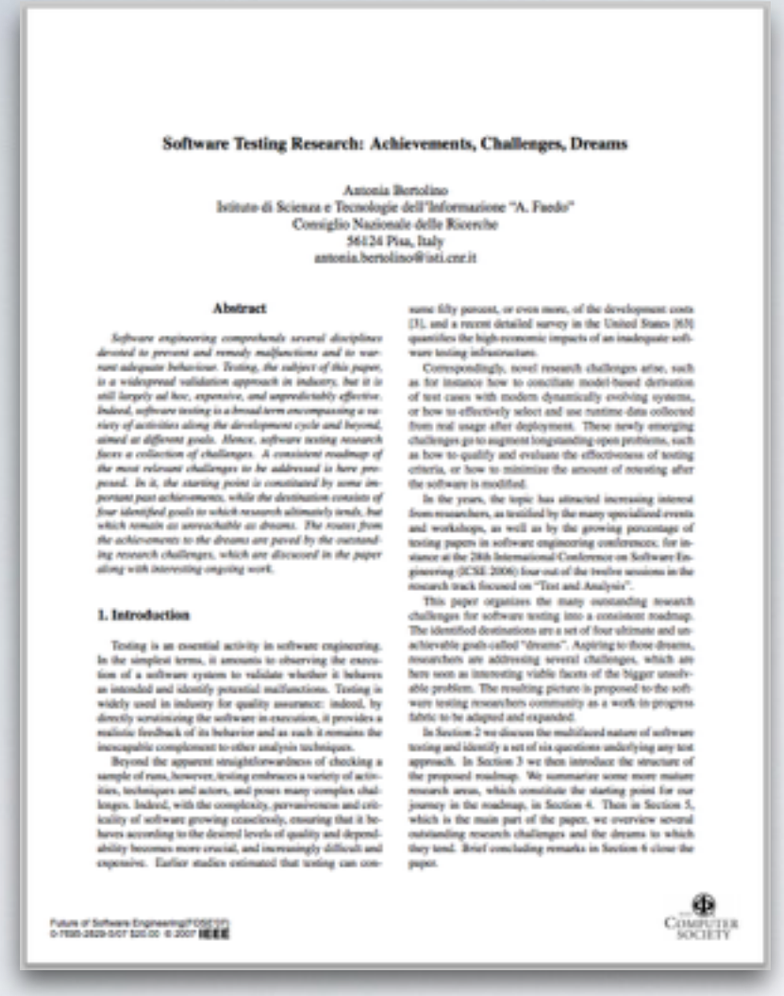
This paper organizes the many outstanding research challenges for software testing into a consistent roadmap. The identified destinations are a set of four ultimate and unachievable goals called "dreams". Aspiring to those dreams, researchers are addressing several challenges, which are here seen as interesting viable facets of the bigger unsolvable problem. The resulting picture is proposed to the software testing researchers community as a work-in-progress fabric to be adapted and expanded.

In Section 2 we discuss the multifaced nature of software testing and identify a set of six questions underlying any test approach. In Section 3 we then introduce the structure of the proposed roadmap. We summarize some more mature research areas, which constitute the starting point for our journey in the roadmap, in Section 4. Then in Section 5, which is the main part of the paper, we overview several outstanding research challenges and the dreams to which they tend. Brief concluding remarks in Section 6 close the paper.

Future of Software Engineering(FOSE'07)
0-7695-2829-5/07 $20.00 © 2007 IEEE

"Testing is a widespread validation approach in industry, but it is still **largely ad hoc**, **expensive**, and **unpredictably effective**."

"Software Testing Research: Achievements, Challenges, Dreams,"
A. Bertolino. *Future of Software Engineering*. IEEE . 2007.



Software Testing Research: Achievements, Challenges, Dreams

Antonia Bertolino
Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"
Consiglio Nazionale delle Ricerche
56124 Pisa, Italy
antonia.bertolino@isti.cnr.it

Abstract

Software engineering comprehends several disciplines devoted to prevent and remedy malfunctions and to warrant adequate behaviour. Testing, the subject of this paper, is a widespread validation approach in industry, but it is still largely ad hoc, expensive, and unpredictably effective. Indeed, software testing is a broad term encompassing a variety of activities along the development cycle and beyond, aimed at different goals. Hence, software testing research faces a collection of challenges. A consistent roadmap of the most relevant challenges to be addressed is here proposed. In it, the starting point is constituted by some important past achievements, while the destination consists of four identified goals to which research ultimately tends, but which remain as unreachable as dreams. The routes from the achievements to the dreams are paved by the outstanding research challenges, which are discussed in the paper along with interesting ongoing work.

1. Introduction

Testing is an essential activity in software engineering. In the simplest terms, it amounts to observing the execution of a software system to validate whether it behaves as intended and identify potential malfunctions. Testing is widely used in industry for quality assurance: indeed, by directly scrutinizing the software in execution, it provides a realistic feedback of its behavior and as such it remains the inescapable complement to other analysis techniques.

Beyond the apparent straightforwardness of checking a sample of runs, however, testing embraces a variety of activities, techniques and actors, and poses many complex challenges. Indeed, with the complexity, pervasiveness and criticality of software growing ceaselessly, ensuring that it behaves according to the desired levels of quality and dependability becomes more crucial, and increasingly difficult and expensive. Earlier studies estimated that testing can consume fifty percent, or even more, of the development costs [3], and a recent detailed survey in the United States [63] quantifies the high economic impacts of an inadequate software testing infrastructure.

Correspondingly, novel research challenges arise, such as for instance how to conciliate model-based derivation of test cases with modern dynamically evolving systems, or how to effectively select and use runtime data collected from real usage after deployment. These newly emerging challenges go to augment longstanding open problems, such as how to qualify and evaluate the effectiveness of testing criteria, or how to minimize the amount of retesting after the software is modified.

In the years, the topic has attracted increasing interest from researchers, as testified by the many specialised events and workshops, as well as by the growing percentage of testing papers in software engineering conferences; for instance at the 28th International Conference on Software Engineering (ICSE 2006) four out of the twelve sessions in the research track focused on "Test and Analysis".

This paper organizes the many outstanding research challenges for software testing into a consistent roadmap. The identified destinations are a set of four ultimate and unachievable goals called "dreams". Aspiring to those dreams, researchers are addressing several challenges, which are here seen as interesting viable facets of the bigger unsolvable problem. The resulting picture is proposed to the software testing researchers community as a work in progress fabric to be adapted and expanded.

In Section 2 we discuss the multifaced nature of software testing and identify a set of six questions underlying any test approach. In Section 3 we then introduce the structure of the proposed roadmap. We summarize some more mature research areas, which constitute the starting point for our journey in the roadmap, in Section 4. Then in Section 5, which is the main part of the paper, we overview several outstanding research challenges and the dreams to which they tend. Brief concluding remarks in Section 6 close the paper.

Future of Software Engineering(FOSE'07)
0-7695-2829-5/07 $20.00 © 2007 IEEE

COMPUTER SOCIETY

"Testing is a widespread validation approach in industry, but it is still **largely ad hoc**, **expensive**, and **unpredictably effective**."

"Software Testing Research: Achievements, Challenges, Dreams,"
A. Bertolino. *Future of Software Engineering*. IEEE . 2007.

"Test case generation has a **strong impact** on the effectiveness and efficiency of testing."

"…one of the most active research topics in software testing for several decades, resulting in **many different approaches and tools**."

"An orchestrated survey of methodologies for automated software test case generation," S. Anand, E. K. Burke, T.Y. Chen, J. Clark, M.B. Cohen, W. Grieskamp, M. Harman, M.J. Harrold, P. McMinn. *J. Systems and Software*. Elsevier. 2013.

The Journal of Systems and Software 86 (2013) 1978–2001

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

An orchestrated survey of methodologies for automated software test case generation

Saswat Anand[a], Edmund K. Burke[b], Tsong Yueh Chen[c], John Clark[d], Myra B. Cohen[e], Wolfgang Grieskamp[f], Mark Harman[g], Mary Jean Harrold[h], Phil McMinn[i]

[a] Stanford University, USA
[b] University of Stirling, Scotland, UK
[c] Swinburne University of Technology, Australia
[d] University of York, UK
[e] University of Nebraska–Lincoln, USA
[f] Microsoft Research, Redmond, USA
[g] University College London, UK
[h] Georgia Institute of Technology, USA
[i] University of Sheffield, UK

Orchestrators and Editors,
Antonia Bertolino[j,*], J. Jenny Li[k,2], Hong Zhu[l,3]

[j] ISTI-CNR, Italy
[k] Avaya Labs Research, USA
[l] Oxford Brookes University, UK

ARTICLE INFO

Article History:
Received 11 February 2013
Accepted 11 February 2013
Available online 17 April 2013

Keywords:
Adaptive random testing
Combinatorial testing
Model-based testing
Orchestrated survey
Search-based software testing
Software testing
Symbolic execution
Test automation
Test case generation

ABSTRACT

Test case generation is among the most labour-intensive tasks in software testing. It also has a strong impact on the effectiveness and efficiency of software testing. For these reasons, it has been one of the most active research topics in software testing for several decades, resulting in many different approaches and tools. This paper presents an orchestrated survey of the most prominent techniques for automatic generation of software test cases, reviewed in self-standing sections. The techniques presented include: (a) structural testing using symbolic execution, (b) model-based testing, (c) combinatorial testing, (d) random testing and its variant of adaptive random testing, and (e) search-based testing. Each section is contributed by world-renowned active researchers on the technique, and briefly covers the basic ideas underlying the method, the current state of the art, a discussion of the open research problems, and a perspective of the future development of the approach. As a whole, the paper aims at giving an introductory, up-to-date and (relatively) short overview of research in automatic test case generation, while ensuring a comprehensive and authoritative treatment.

ISSTA 2015 Keynote Address by Prof. Yuanyuan Zhou, U.C.S.D.

The gap between research and practice

In the last two decades, considerable research has been conducted in software testing and debugging. Unfortunately most research innovations did not make into practice; and software development and testing in the real world still lag behind in tooling support and automation. In this talk, I will share my limited experience and lessons learned in taking our research ideas into commercial tools as well as deploying them in large companies. I will discuss a few assumptions that we often make in our research but can significantly limit their adoption in practice. Additionally,I will also present some open problems that I have observed from interacting with customers and understanding their typical software testing workflows.

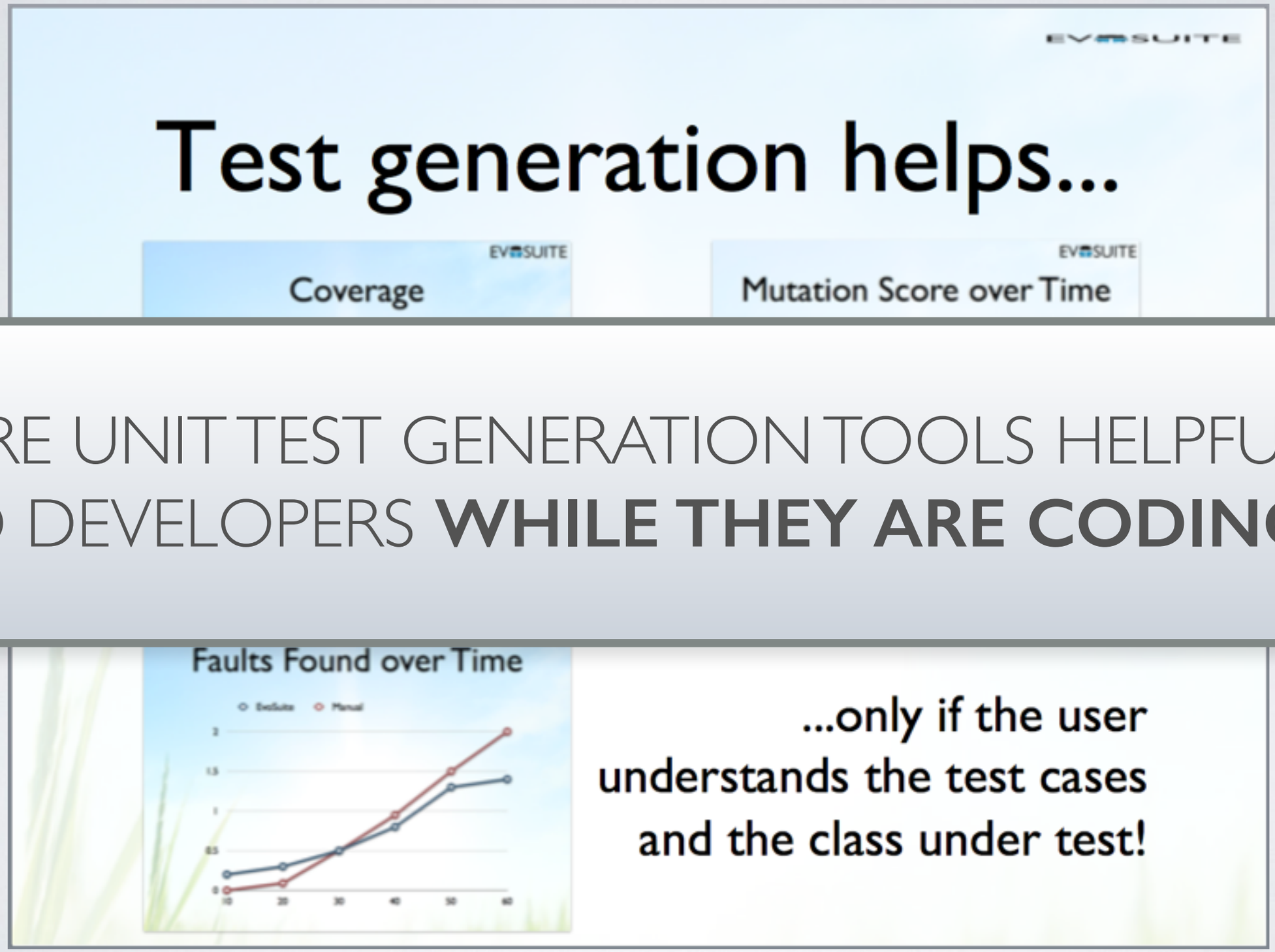
BACK IN ISSTA 2013…

Test generation helps...

...only if the user understands the test cases and the class under test!

"Does automated white-box test generation really help software testers?," G. Fraser, M. Staats, P. McMinn, A. Arcuri and F. Padberg

BACK IN ISSTA 2013…



ARE UNIT TEST GENERATION TOOLS HELPFUL TO DEVELOPERS **WHILE THEY ARE CODING**?

"Does automated white-box test generation really help software testers?," G. Fraser, M. Staats, P. McMinn, A. Arcuri and F. Padberg

CODE COVERAGE

CODE COVERAGE

TIME SPENT ON TESTING

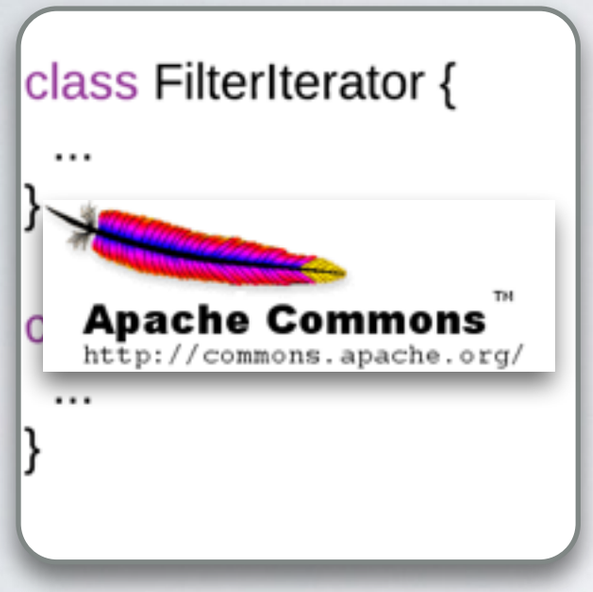
CODE COVERAGE

TIME SPENT ON TESTING

IMPLEMENTATION QUALITY

CONTROLLED EXPERIMENT

CONTROLLED EXPERIMENT



Golden Implementation
and Test Suite

CONTROLLED EXPERIMENT

Golden Implementation and Test Suite

Class Template

CONTROLLED EXPERIMENT

```
class FilterIterator {
  ...
}
```


```
  ...
}
```

Golden Implementation and Test Suite

```
class FilterIterator {
  /**
   *  Description
   */
  public void remove() {
    // TODO
  }
}
```

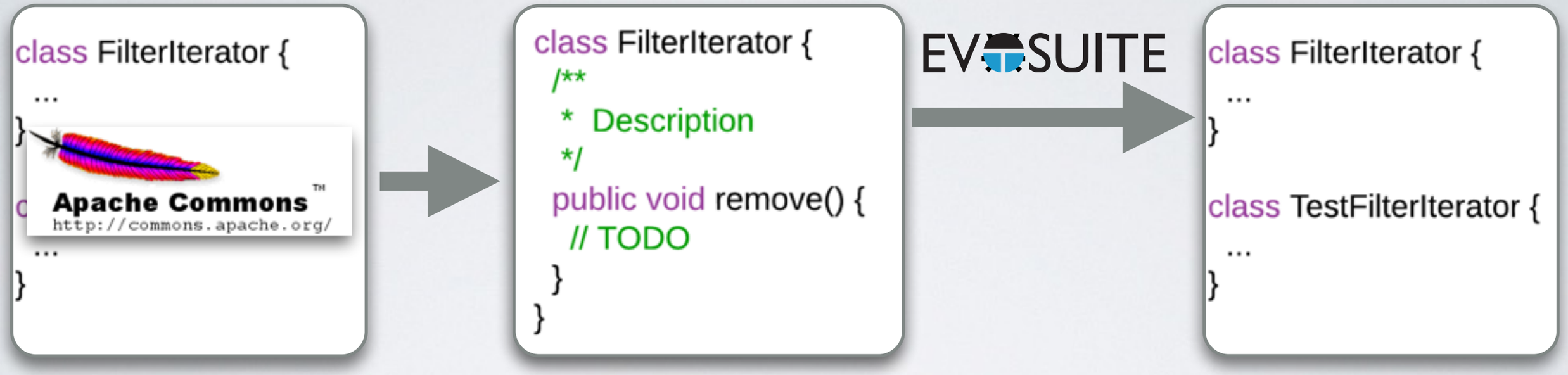
Class Template

```
class FilterIterator {
  ...
}

class TestFilterIterator {
  ...
}
```

Implementation and Test Suite

CONTROLLED EXPERIMENT

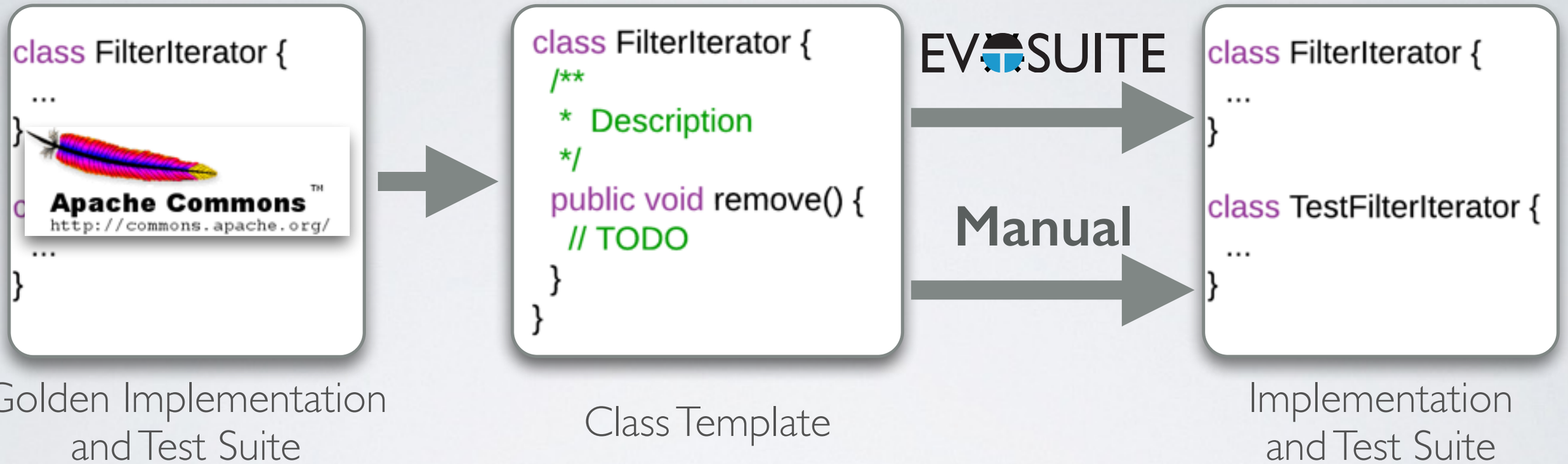


Golden Implementation and Test Suite

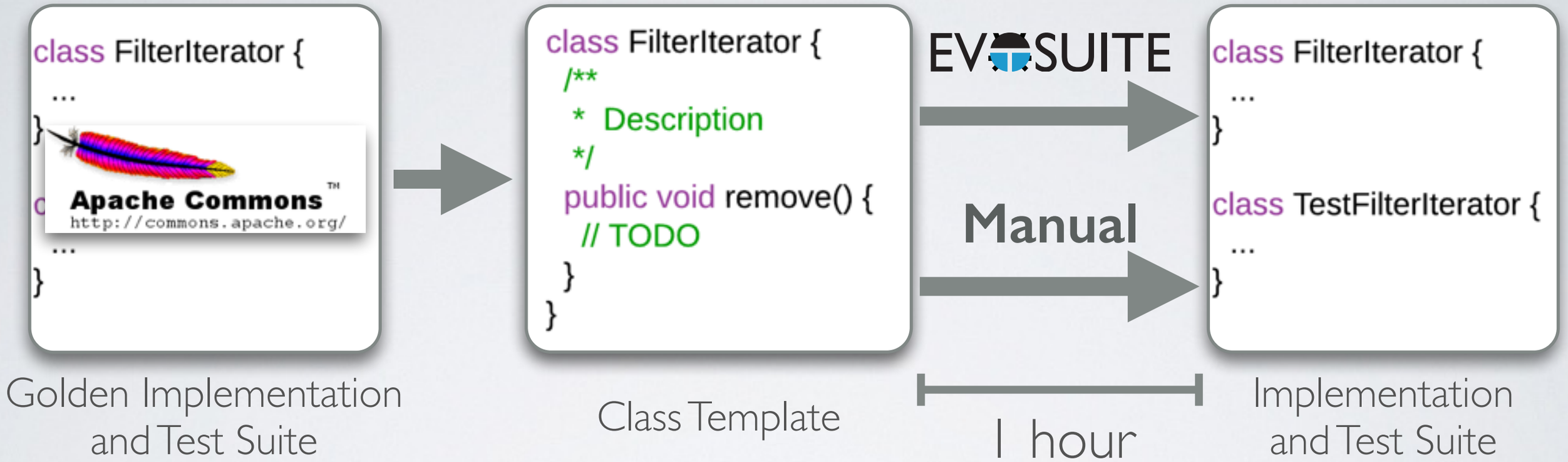
Class Template

Implementation and Test Suite

CONTROLLED EXPERIMENT



CONTROLLED EXPERIMENT

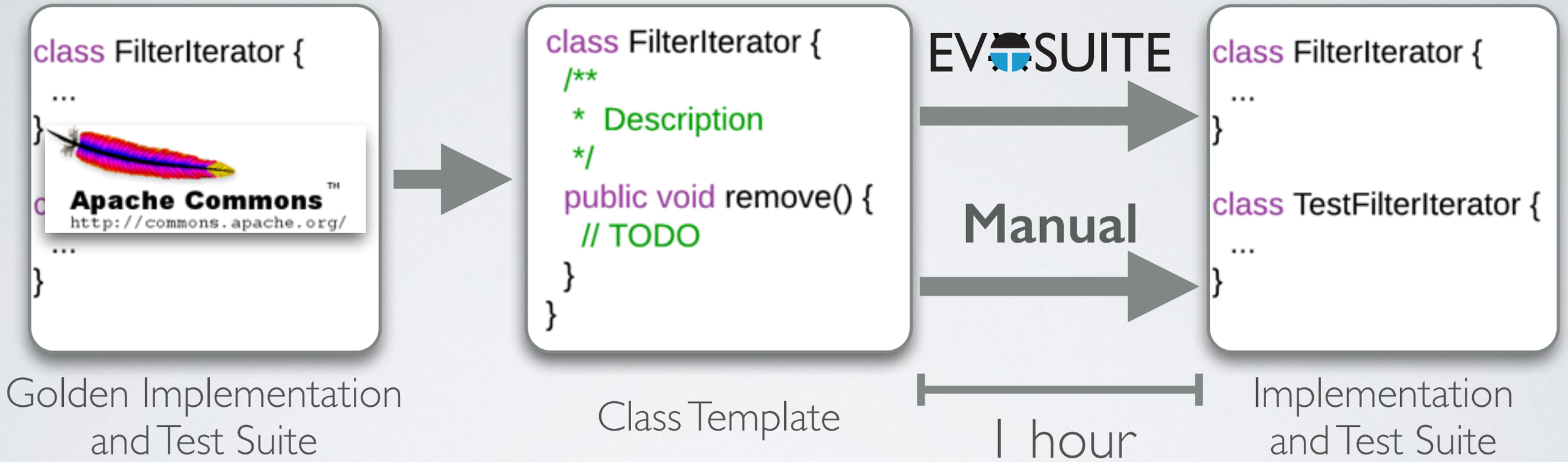


Golden Implementation and Test Suite

Class Template

Implementation and Test Suite

CONTROLLED EXPERIMENT



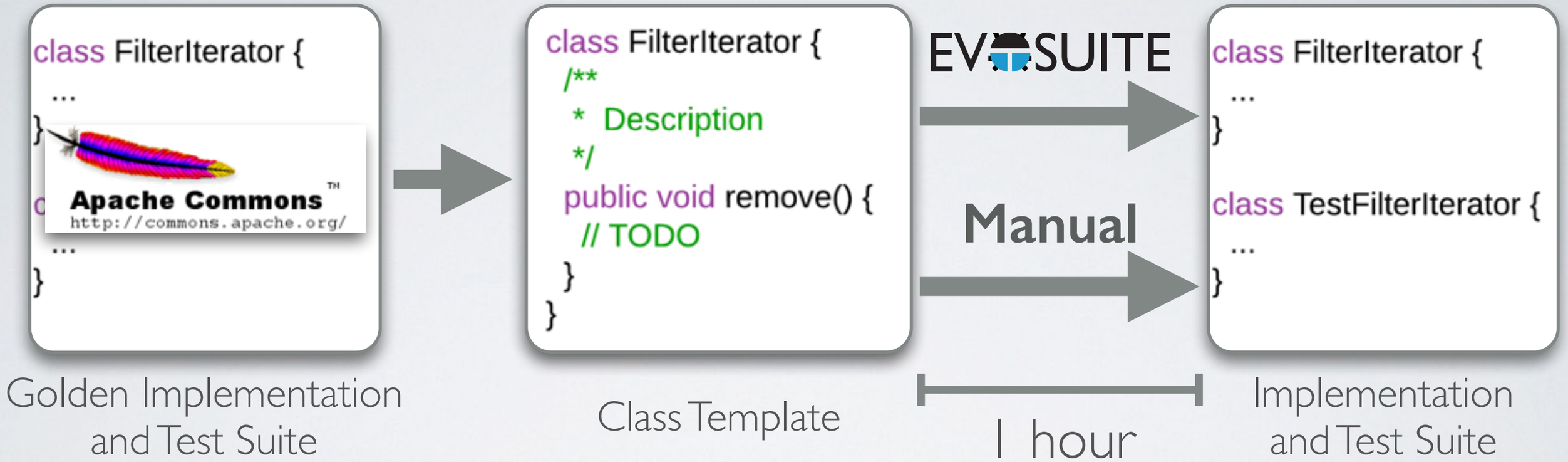
Golden Implementation and Test Suite

Class Template

Implementation and Test Suite

41

CONTROLLED EXPERIMENT



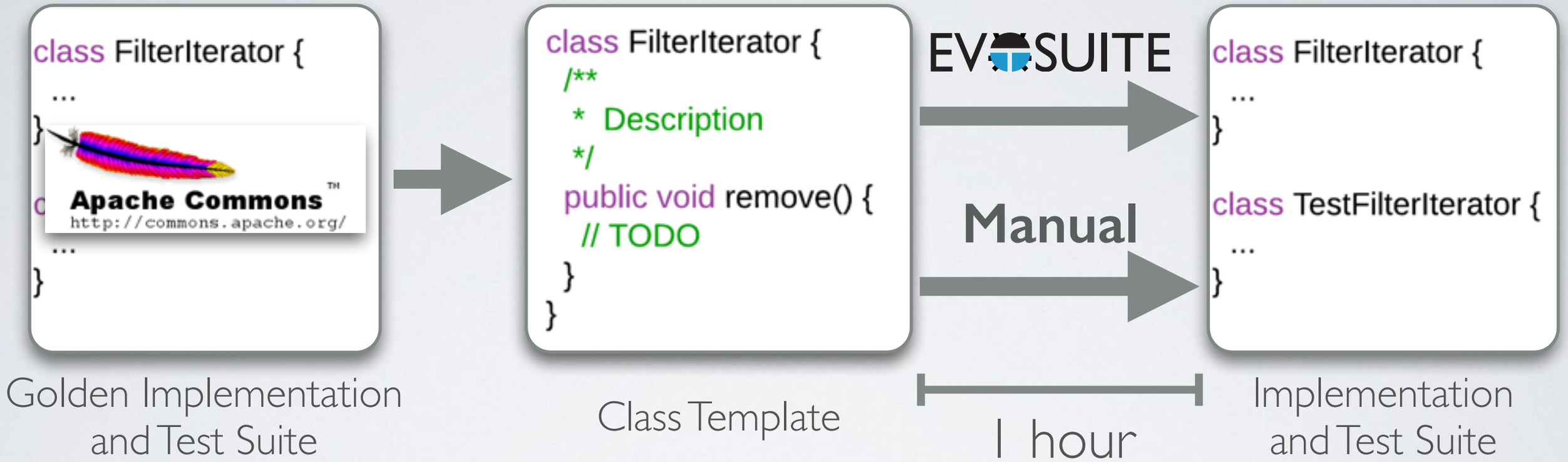
Golden Implementation and Test Suite

Class Template

Implementation and Test Suite

41

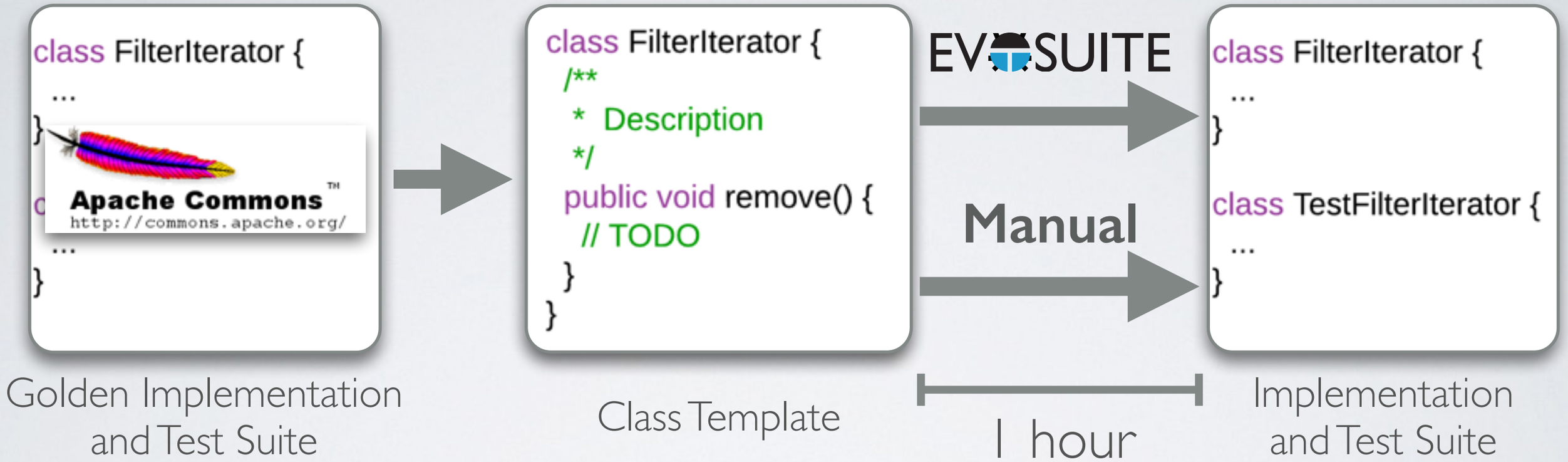
CONTROLLED EXPERIMENT



41

2

CONTROLLED EXPERIMENT



41

2

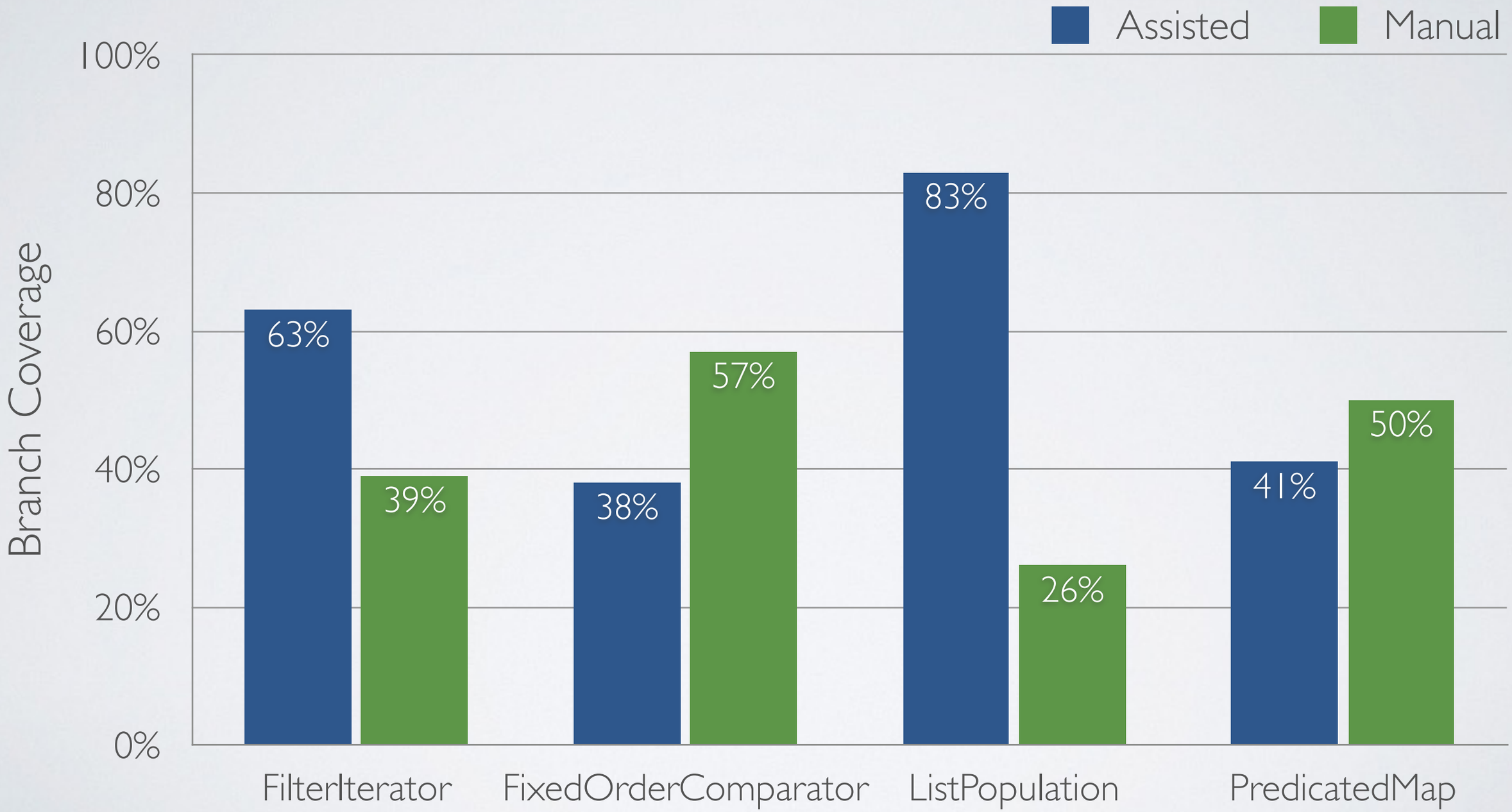
4

DOES USING **EVOSUITE** DURING SOFTWARE DEVELOPMENT LEAD TO TEST SUITES WITH HIGHER CODE COVERAGE?

RQ 1

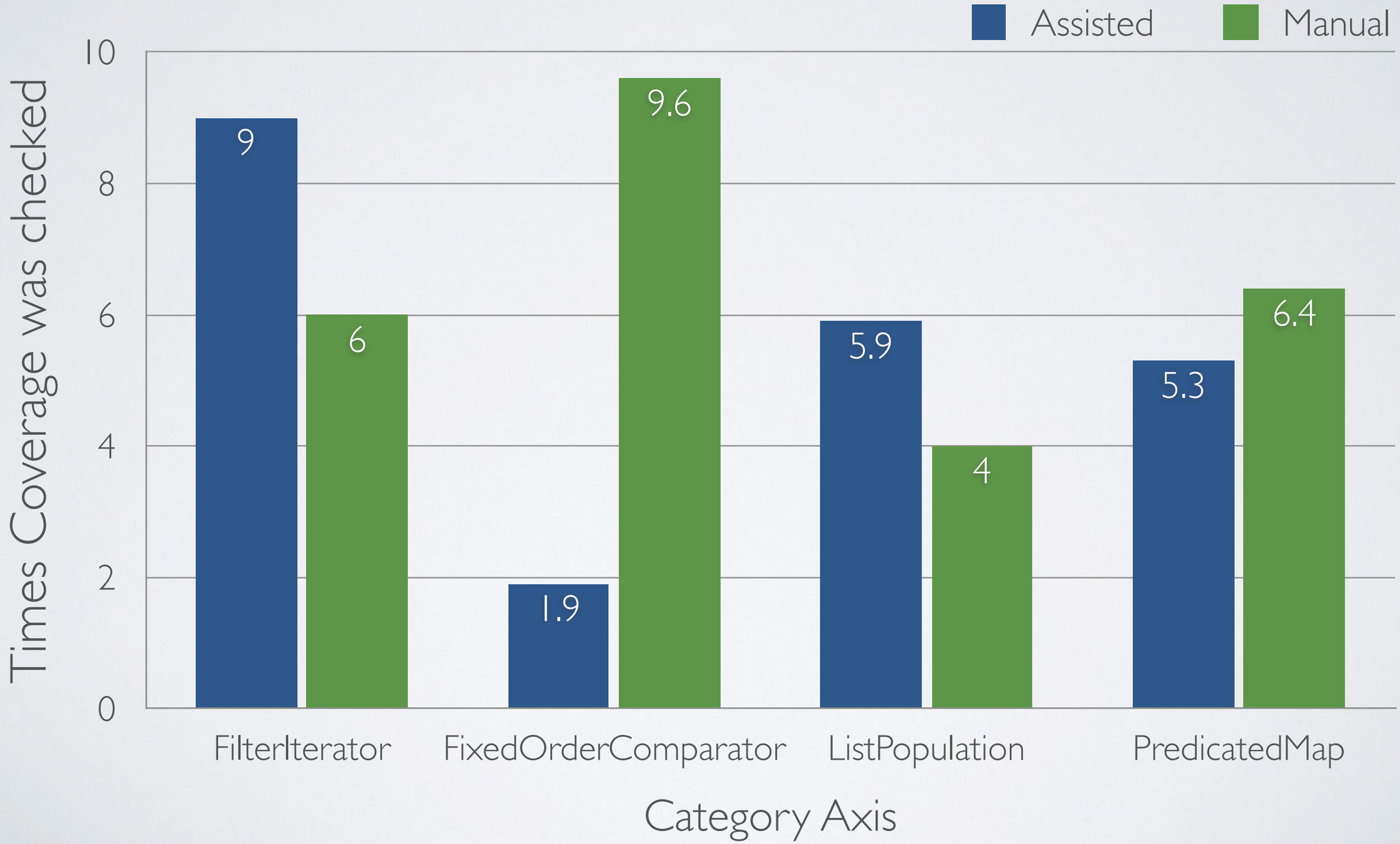
CODE COVERAGE

participants' test suites run on their own implementations



CODE COVERAGE

Times coverage was checked

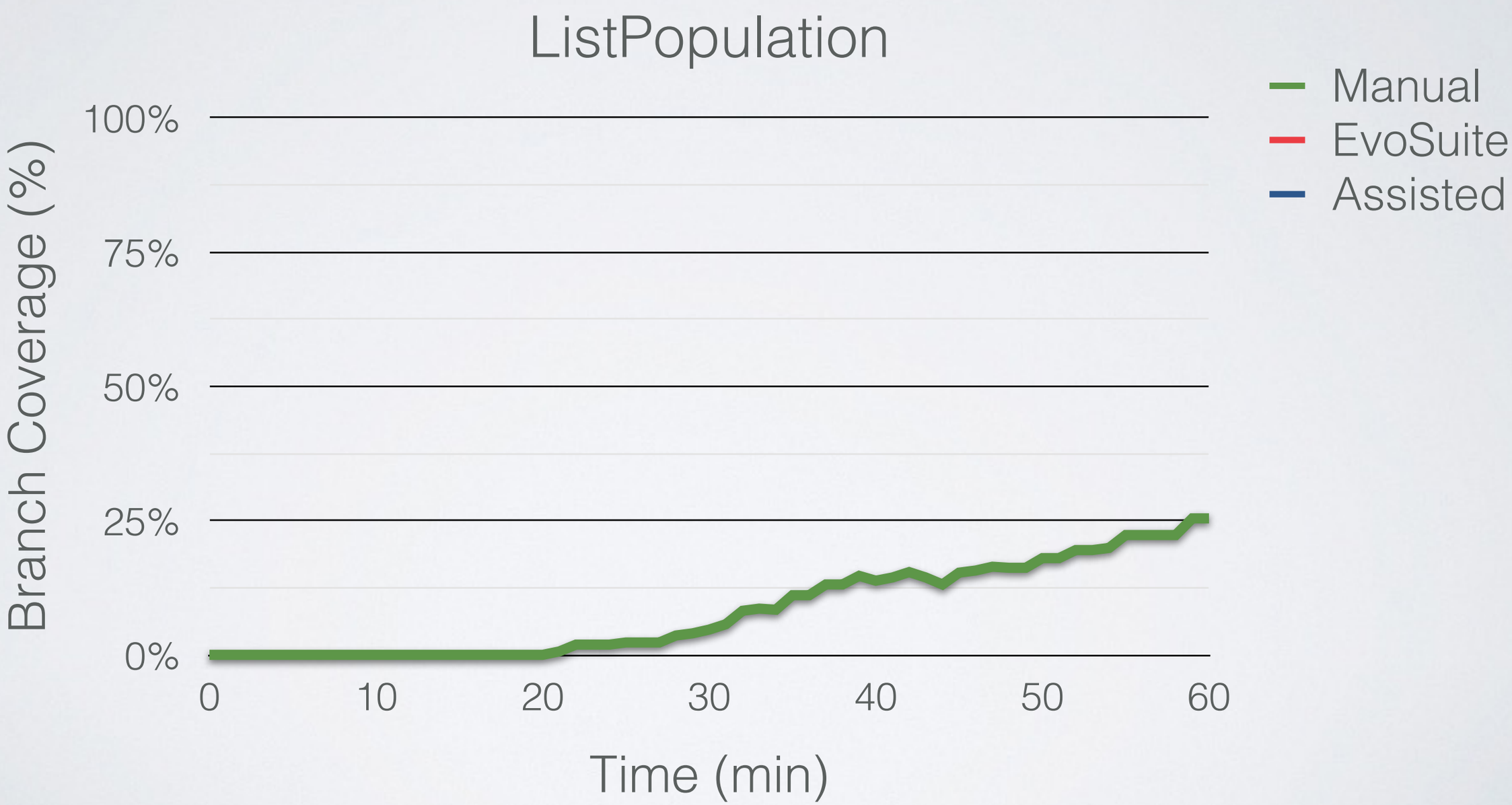


CODE COVERAGE

participant's test suites run on their own implementations

CODE COVERAGE

participant's test suites run on their own implementations



CODE COVERAGE

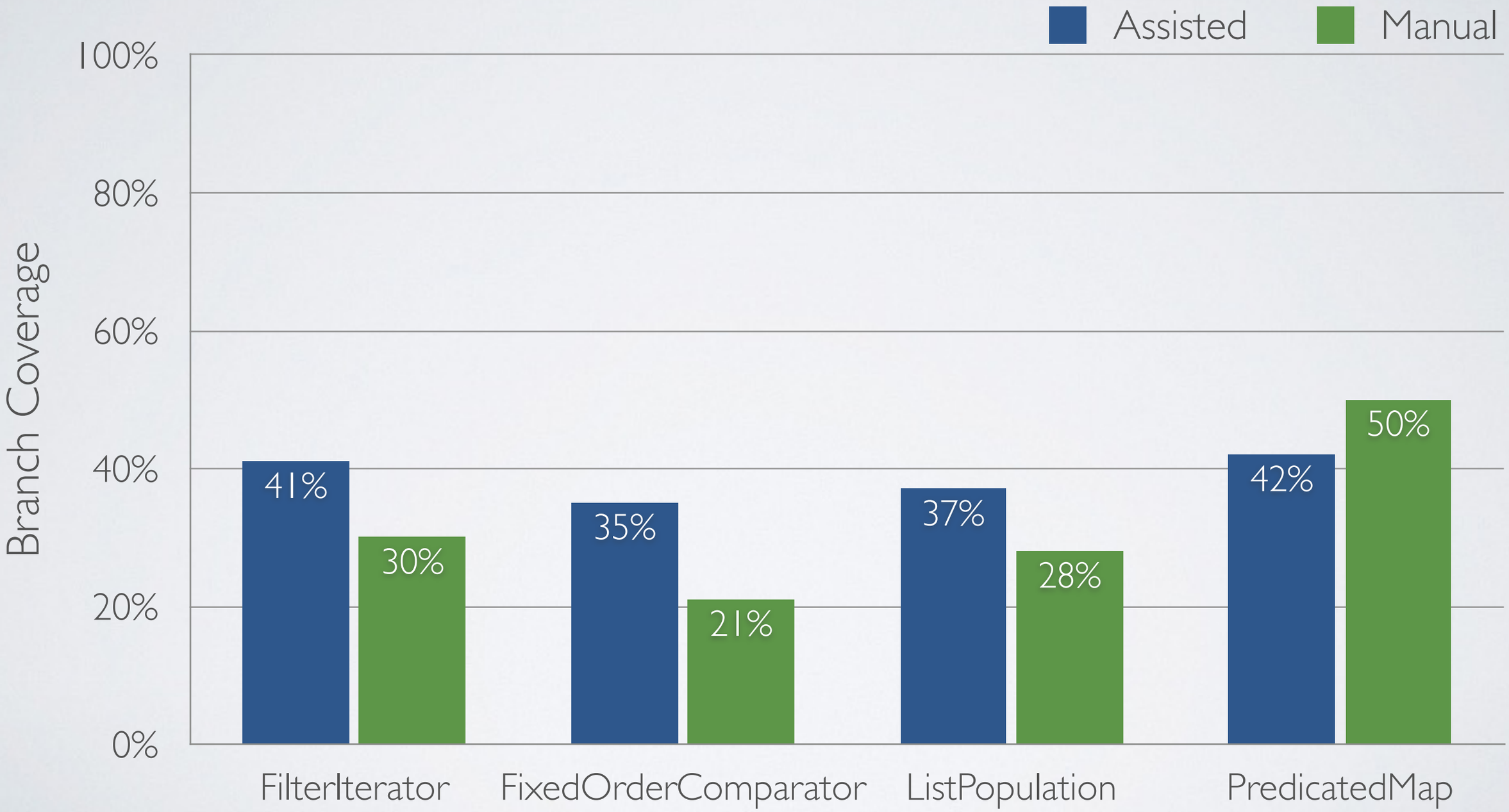
participant's test suites run on their own implementations

CODE COVERAGE

participant's test suites run on their own implementations

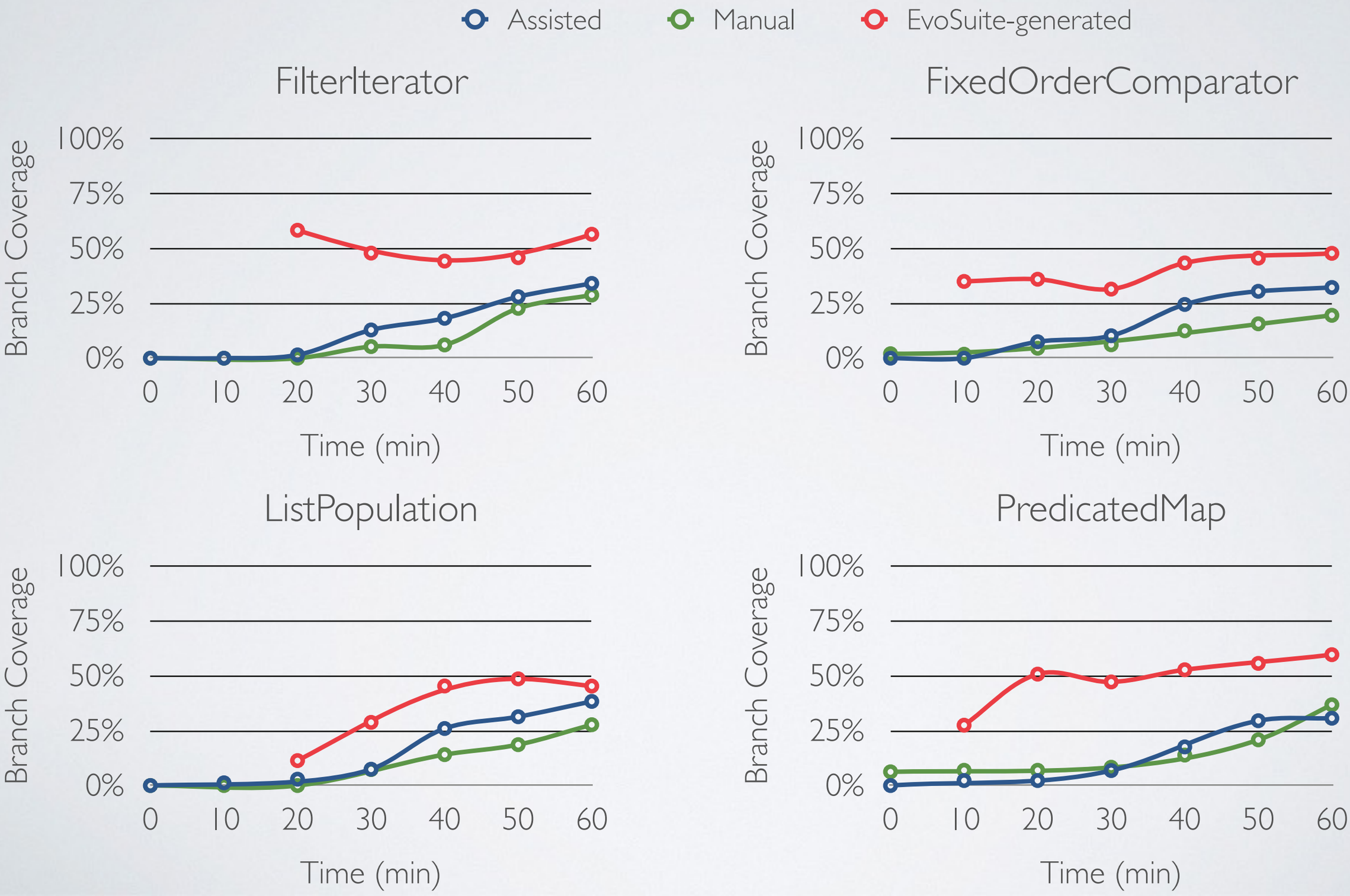
CODE COVERAGE

participants' test suites run **on golden implementations**



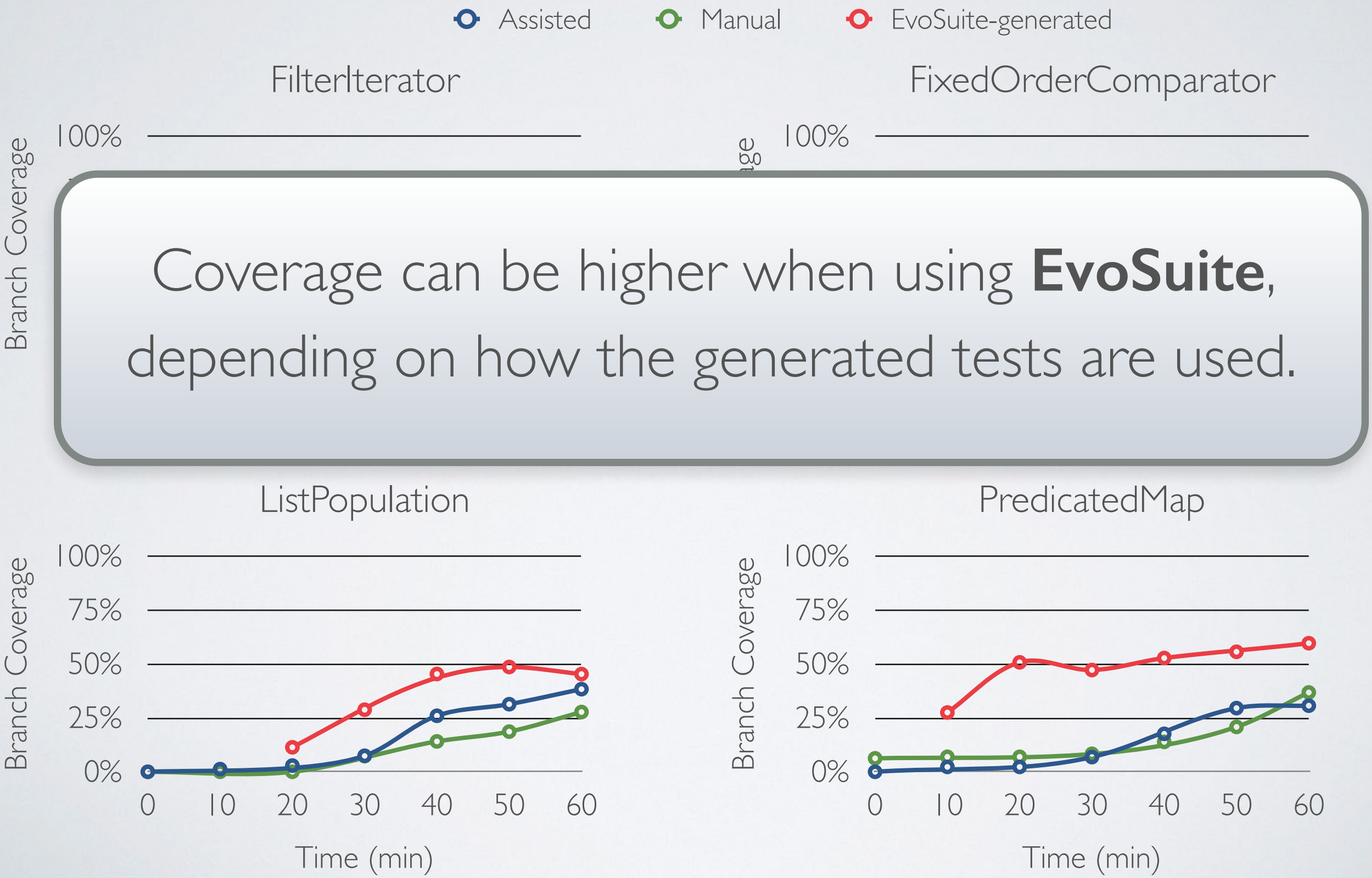
CODE COVERAGE

participant's test suites run on golden implementations, **over time**



CODE COVERAGE

participant's test suites run on golden implementations, **over time**



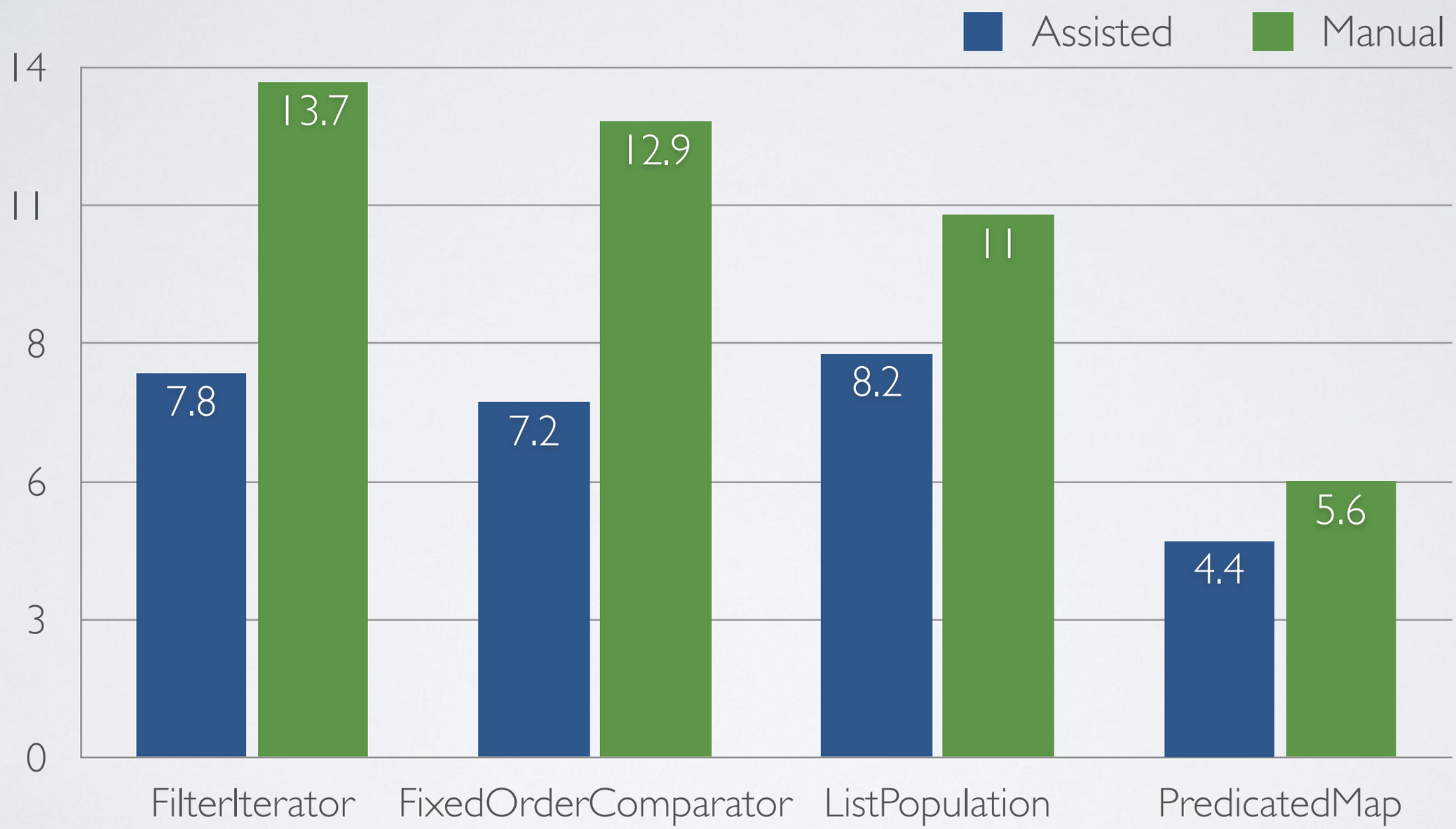
Coverage can be higher when using **EvoSuite**, depending on how the generated tests are used.

DOES USING **EVOSUITE** DURING SOFTWARE DEVELOPMENT LEAD TO DEVELOPERS SPENDING MORE OR LESS TIME ON TESTING?

RQ 2

TESTING EFFORT

Number of test runs

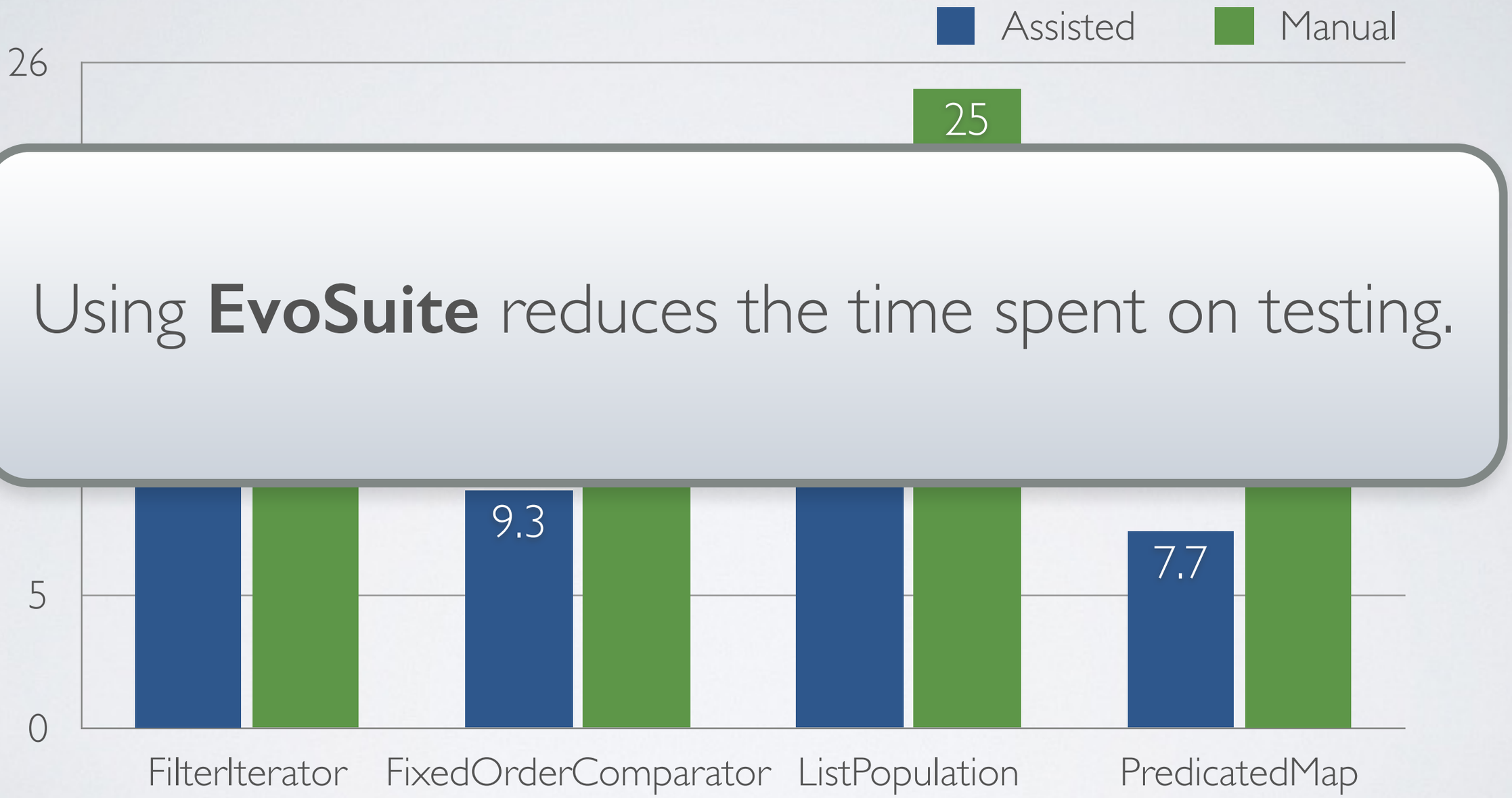


TESTING EFFORT

Minutes spent on testing

TESTING EFFORT

Minutes spent on testing



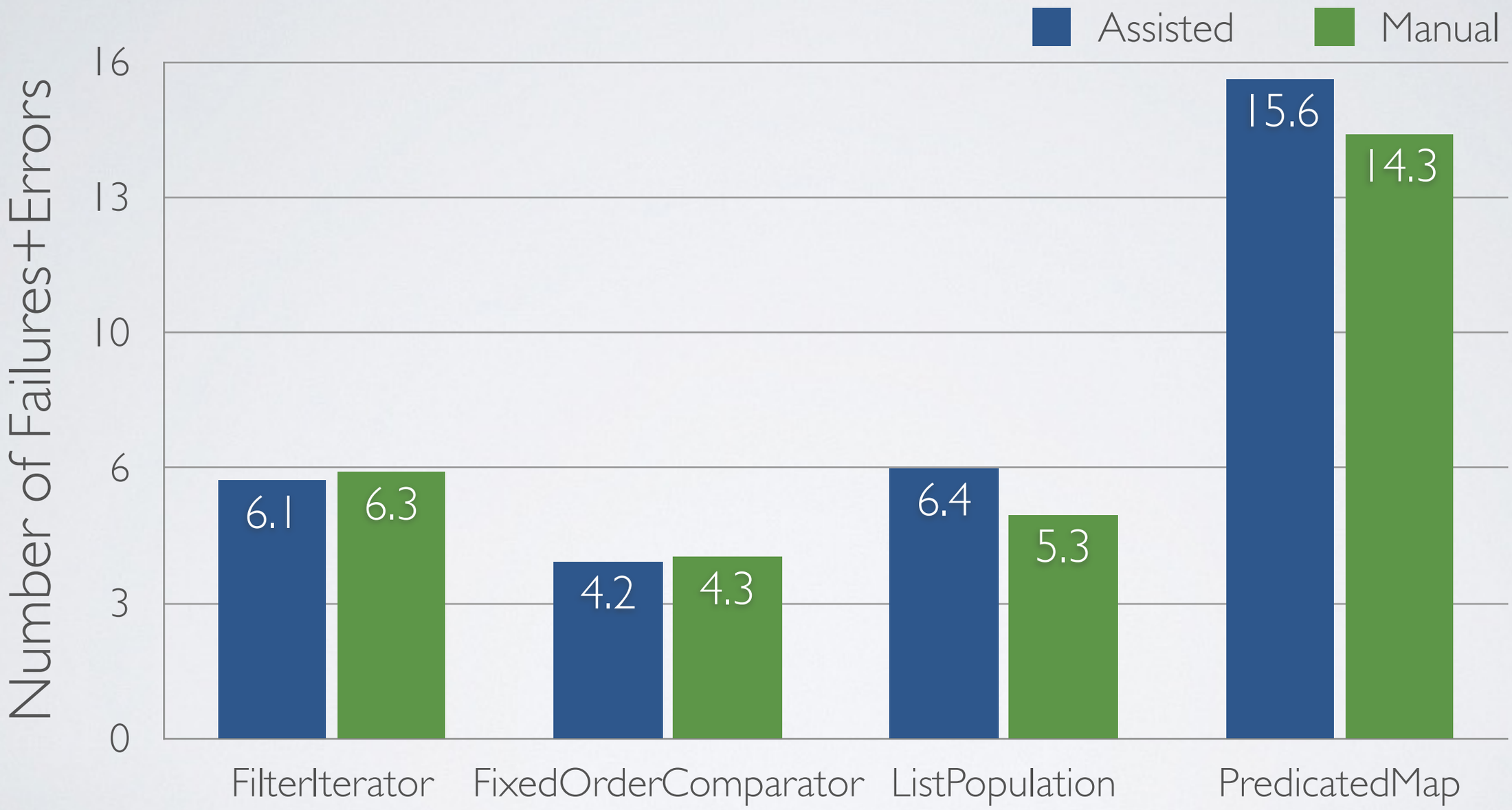
Using **EvoSuite** reduces the time spent on testing.

DOES USING **EVOSUITE** DURING SOFTWARE DEVELOPMENT LEAD TO SOFTWARE WITH FEWER BUGS?

RQ 3

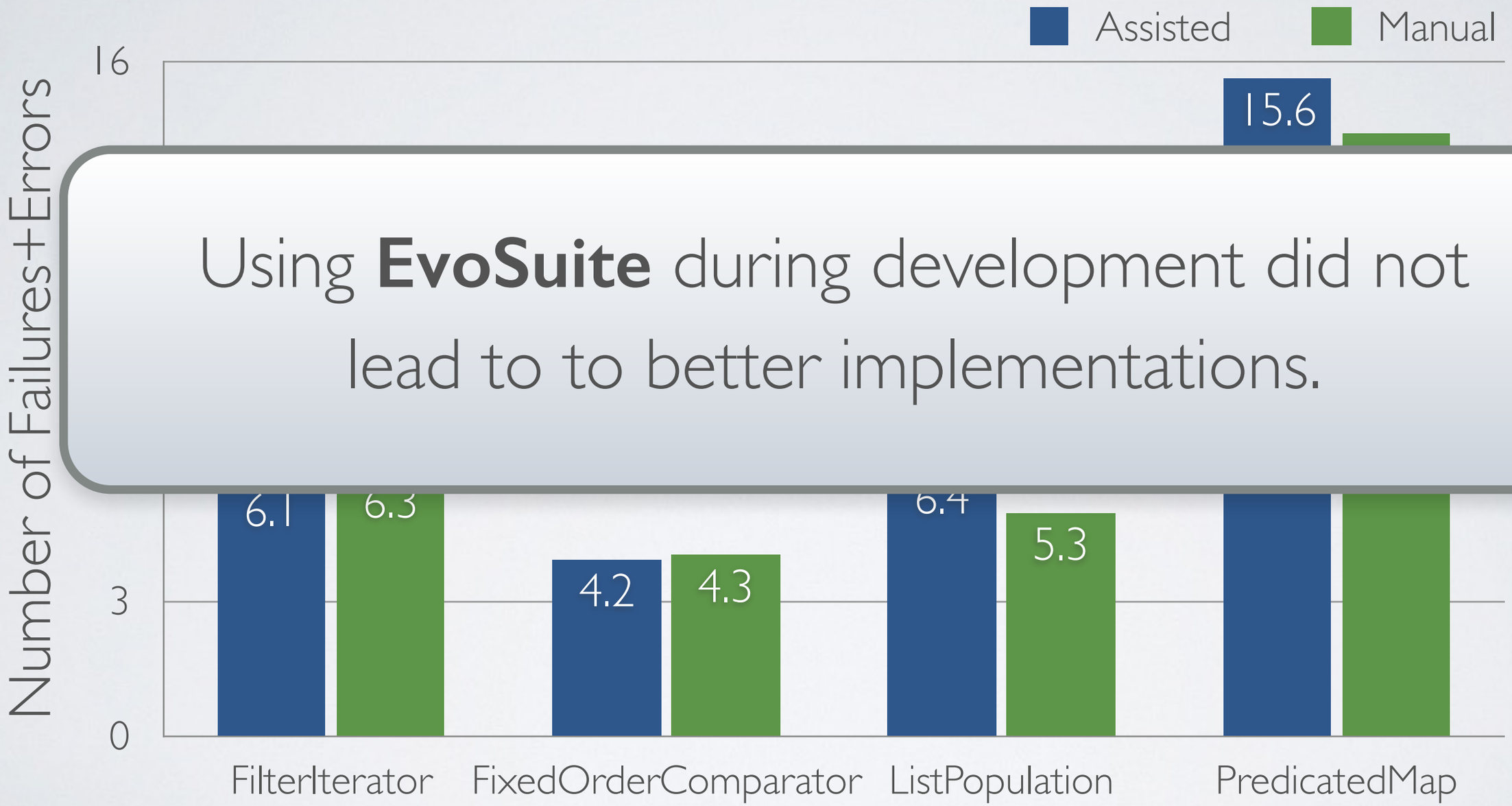
IMPLEMENTATION QUALITY

Golden test suites run on participants' implementations



IMPLEMENTATION QUALITY

Golden test suites run on participants' implementations



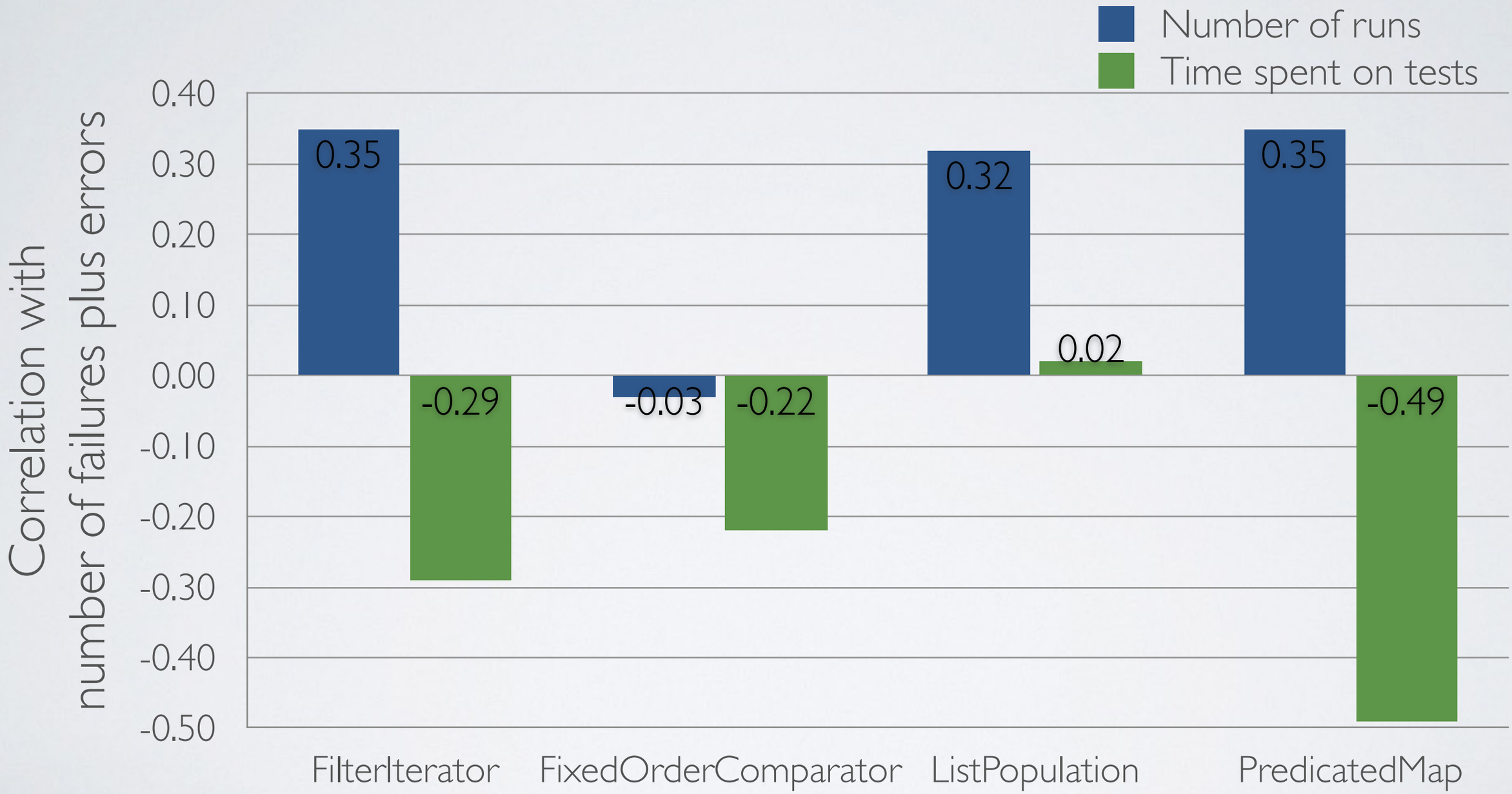
Using **EvoSuite** during development did not lead to to better implementations.

DOES SPENDING MORE TIME WITH **EVOSUITE** AND ITS TESTS LEAD TO BETTER IMPLEMENTATIONS?

RQ 4

PRODUCTIVITY

Time spent with **EvoSuite**



PRODUCTIVITY

Time spent with **EvoSuite**

Implementation quality improves the more time developers spend with **EvoSuite**-generated tests.

Using automated unit test generation does impact developers' productivity,

Using automated unit test generation does impact developers' productivity,

but…

Using automated unit test generation does impact developers' productivity,

but…

…how to make the most out of unit test generation tools?

THINK ALOUD OBSERVATIONS

K. A. Ericsson and H. A. Simon, *Protocol Analysis: Verbal Reports as Data (revised edition)*. MIT Press, 1993.

J. Hughes and S. Parkes, "Trends in the use of verbal protocol analysis in software engineering research," *Behaviour and Information Technology*, vol. 22, no. 2, pp. 127–140, 2003.

THINK ALOUD OBSERVATIONS

Subject

K. A. Ericsson and H. A. Simon, *Protocol Analysis: Verbal Reports as Data (revised edition)*. MIT Press, 1993.

J. Hughes and S. Parkes, "Trends in the use of verbal protocol analysis in software engineering research," *Behaviour and Information Technology*, vol. 22, no. 2, pp. 127–140, 2003.

THINK ALOUD OBSERVATIONS

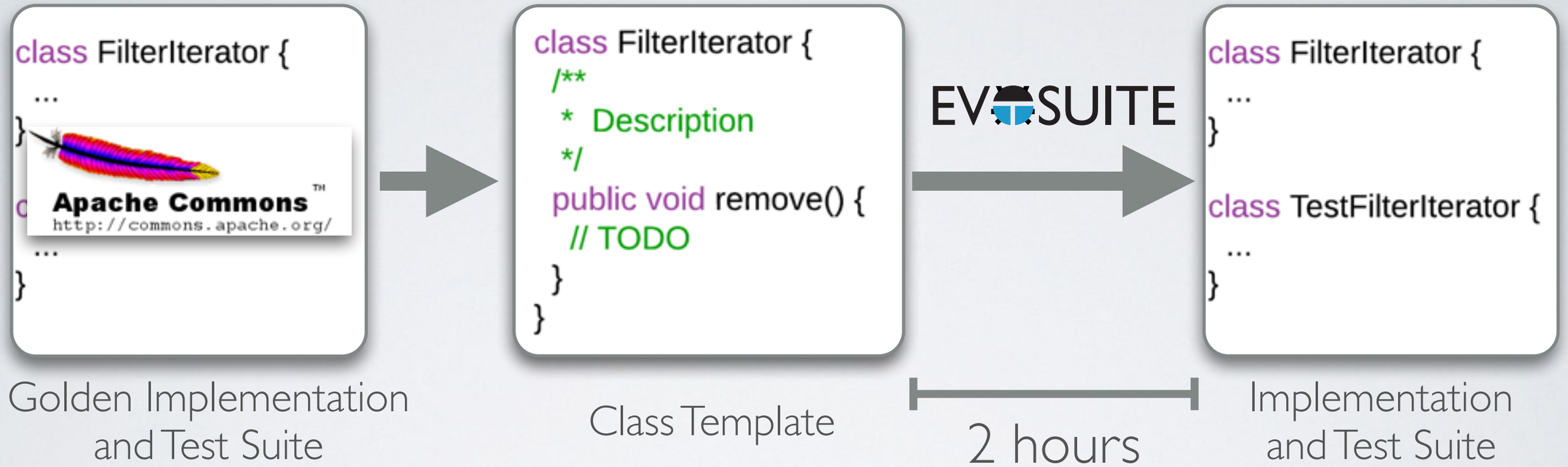
Observer

Subject

K. A. Ericsson and H. A. Simon, *Protocol Analysis: Verbal Reports as Data (revised edition)*. MIT Press, 1993.

J. Hughes and S. Parkes, "Trends in the use of verbal protocol analysis in software engineering research," *Behaviour and Information Technology*, vol. 22, no. 2, pp. 127–140, 2003.

THINK ALOUD OBSERVATIONS

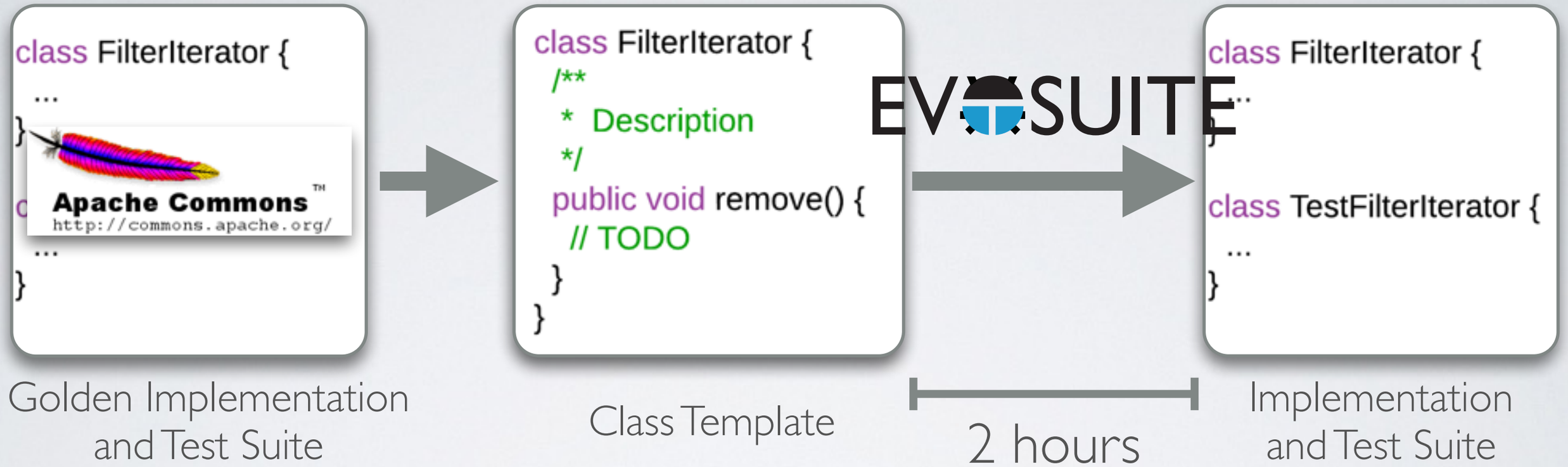


5

1

4

THINK ALOUD OBSERVATIONS

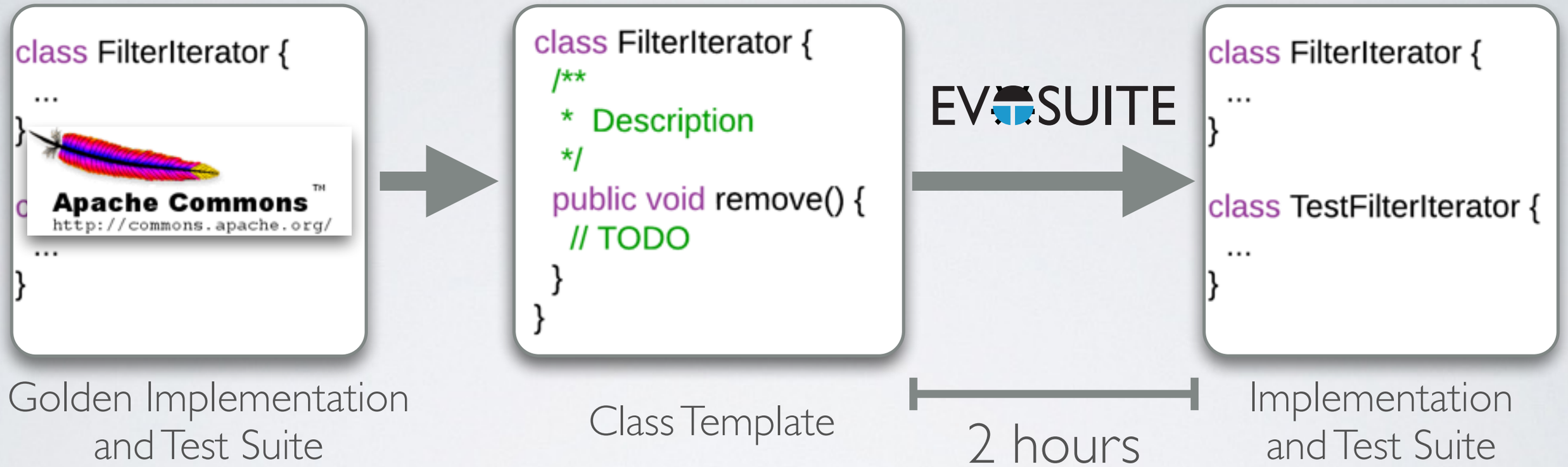


5

1

4

THINK ALOUD OBSERVATIONS

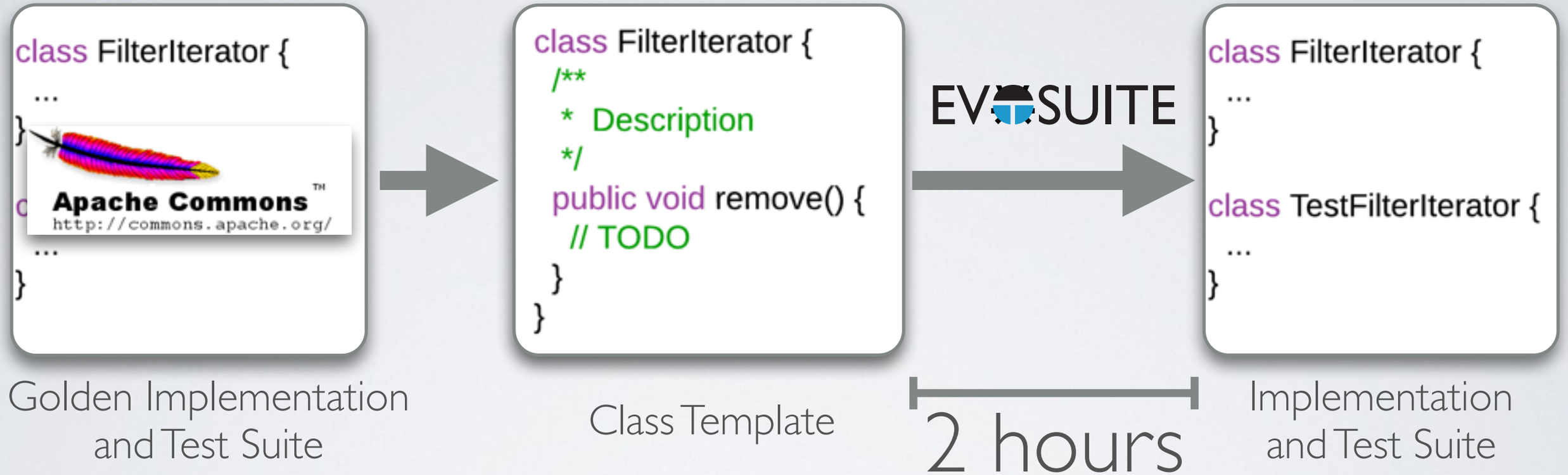


5

1

4

THINK ALOUD OBSERVATIONS

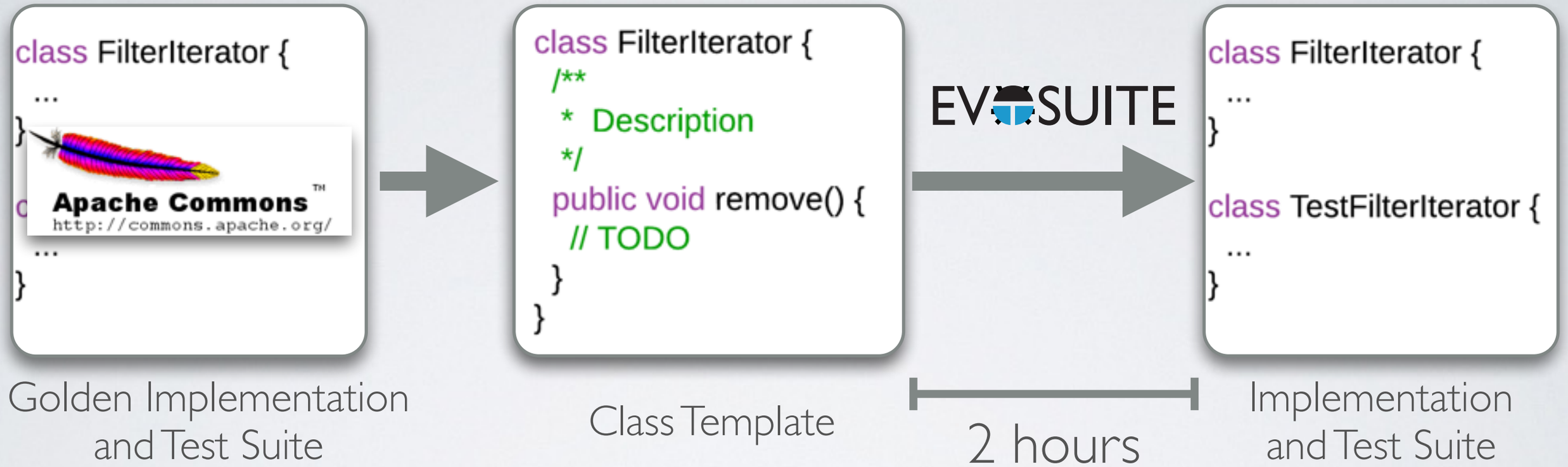


5

1

4

THINK ALOUD OBSERVATIONS

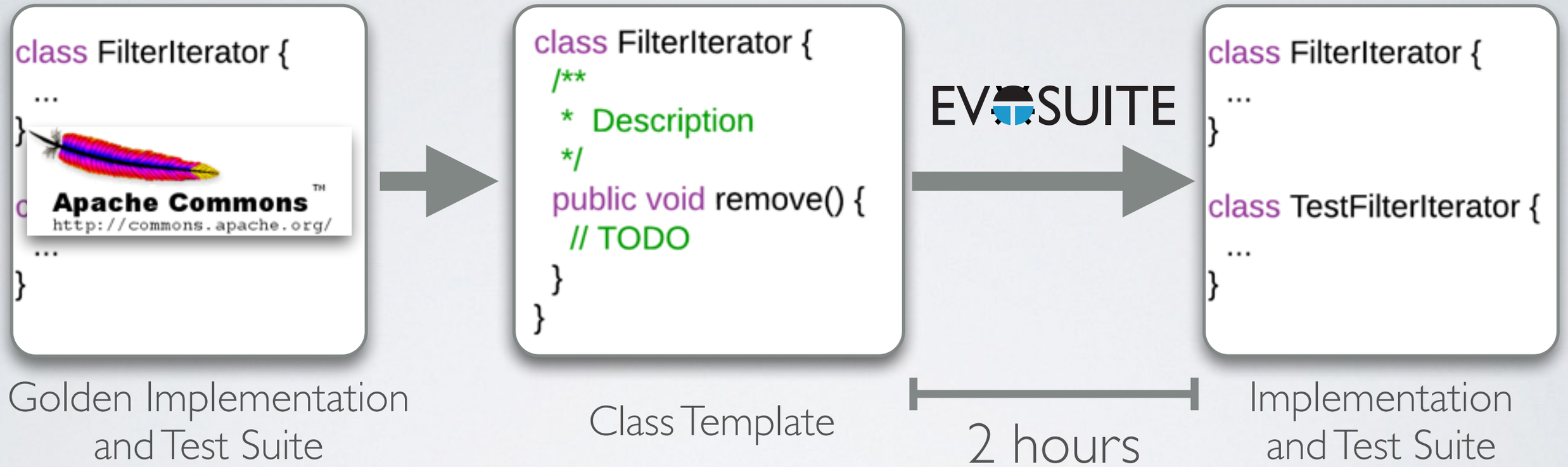


5

1

4

THINK ALOUD OBSERVATIONS

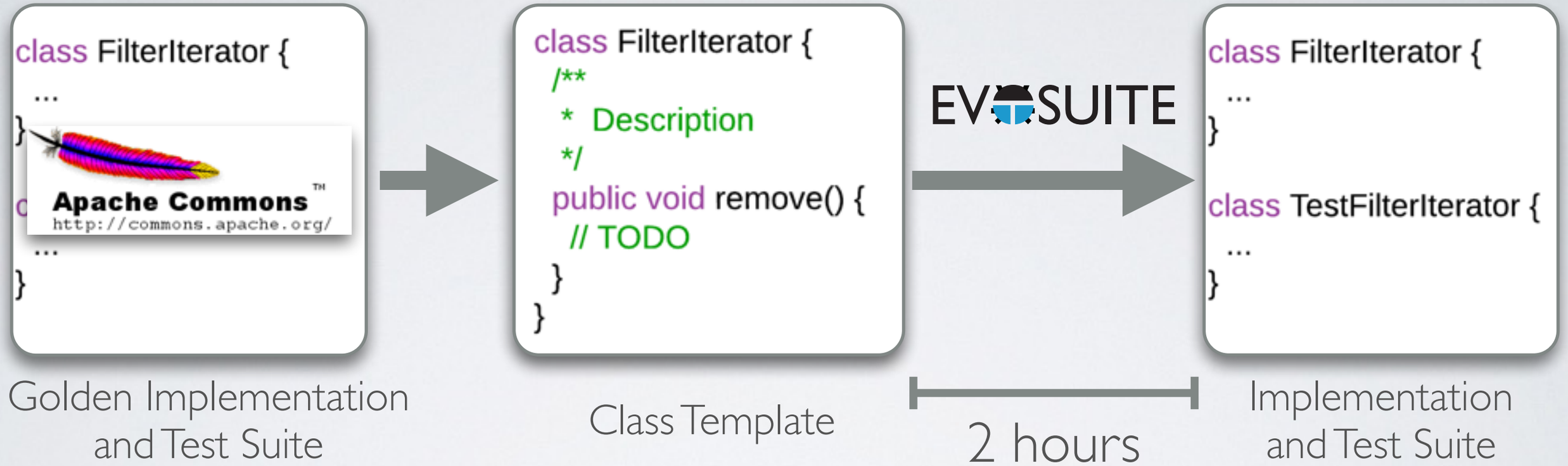


5

1

4

THINK ALOUD OBSERVATIONS

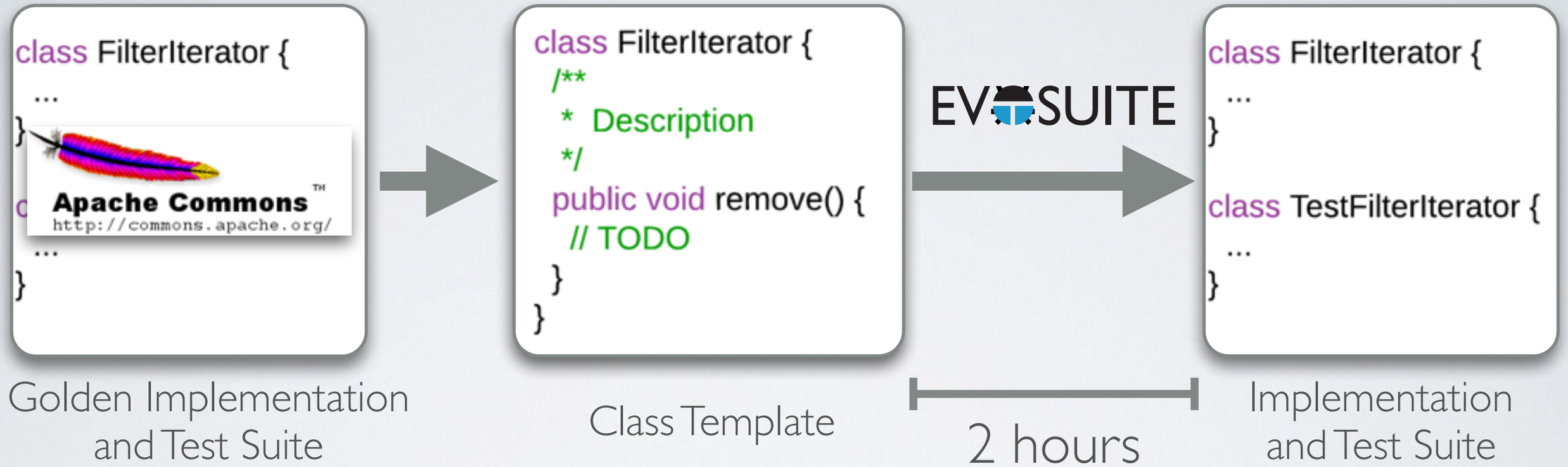


5

1

4

THINK ALOUD OBSERVATIONS



Golden Implementation and Test Suite

Class Template

2 hours

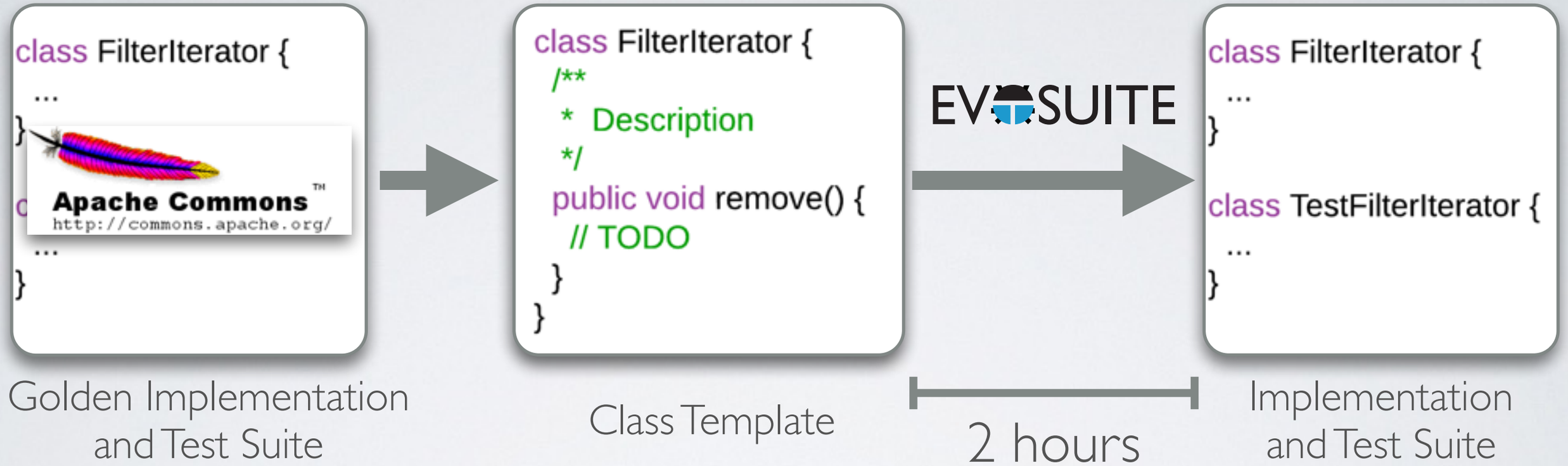
Implementation and Test Suite

5

1

4

THINK ALOUD OBSERVATIONS

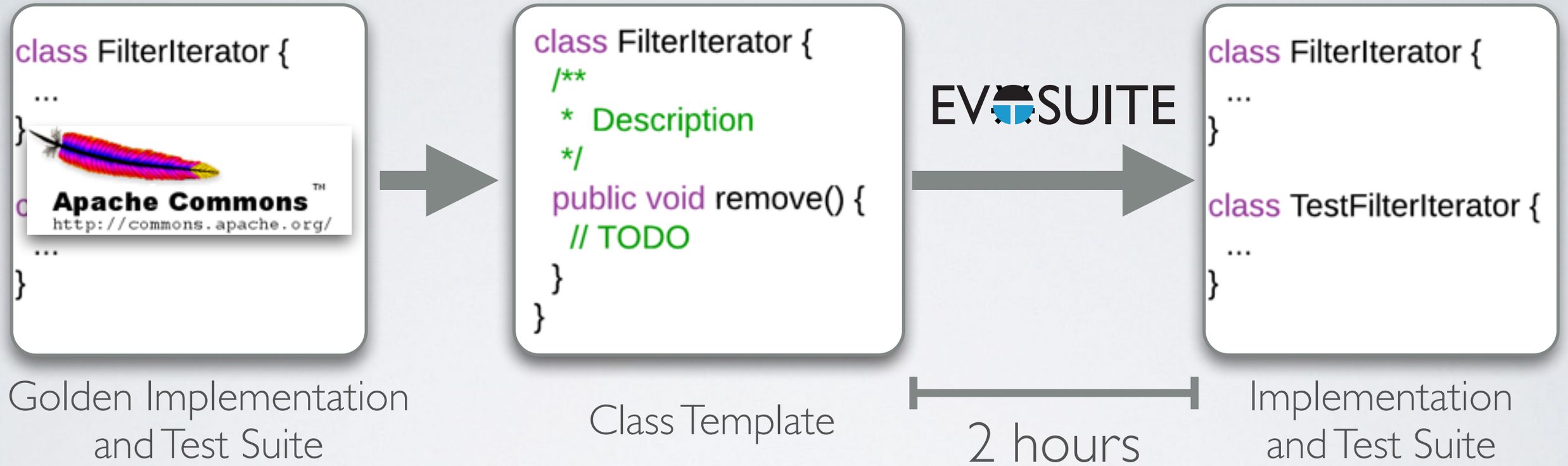


5

1

4

THINK ALOUD OBSERVATIONS



5

1

4

RESULTS

	FilterIterator	FixedOrderComparator	ListPopulation	PredicatedMap	PredicatedMap
EVOSUITE	3	6	2	2	5
JUnit	15	20	23	7	10
	94% | 92%	97% | 72%	94% | 95%	100% | 94%	100% | 83%

Images: http://jozef89.deviantart.com/

RESULTS

	FilterIterator	FixedOrderComparator	**ListPopulation**	PredicatedMap	PredicatedMap
EVOSUITE	3	6	2	2	5
JUnit	15	20	23	7	10
	94%	97%	94% | 95%	100%	100%

Images: http://jozef89.deviantart.com/

RESULTS

	FilterIterator	**FixedOrderComparator**	ListPopulation	PredicatedMap	PredicatedMap
EVOSUITE	3	6	2	2	5
JUnit	15	20	23	7	10
	94%	97% | 72%	94%	100%	100%

Images: http://jozef89.deviantart.com/

LESSONS LEARNED

LESSONS LEARNED

- There are different approaches to testing and test generation tools should be **adaptable** to them

LESSONS LEARNED

- There are different approaches to testing and test generation tools should be **adaptable** to them

- Developers' behaviour is often **not driven by code coverage**

LESSONS LEARNED

- There are different approaches to testing and test generation tools should be **adaptable** to them

- Developers' behaviour is often **not driven by code coverage**

- **Readability** of generated unit tests is paramount

LESSONS LEARNED

- There are different approaches to testing and test generation tools should be **adaptable** to them

- Developers' behaviour is often **not driven by code coverage**

- **Readability** of generated unit tests is paramount

- **Integration** into development environments **must** be improved

LESSONS LEARNED

- There are different approaches to testing and test generation tools should be **adaptable** to them

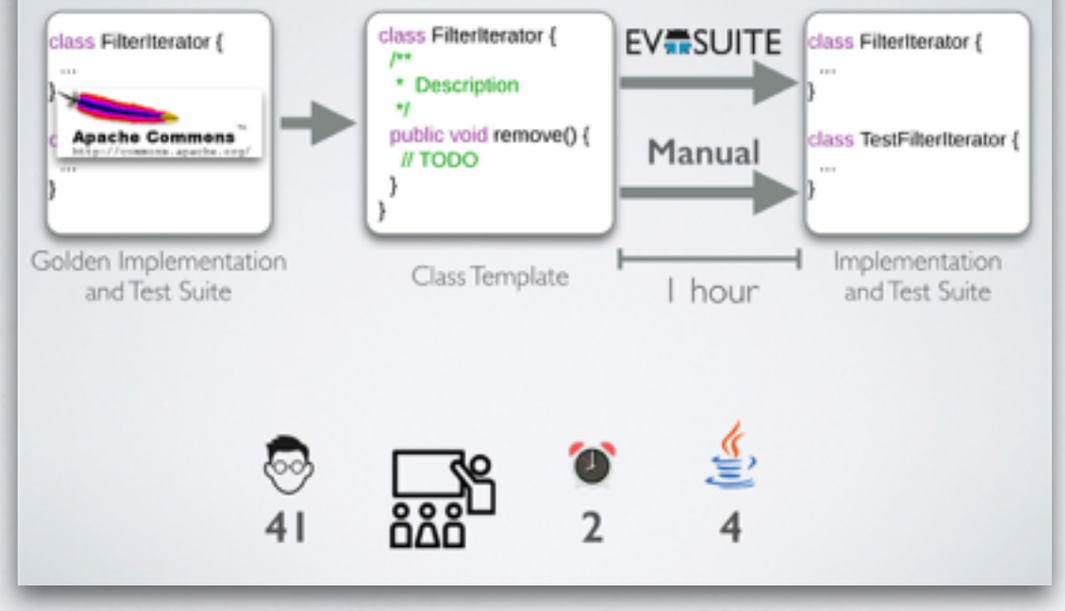
- Developers' behaviour is often **not driven by code coverage**

- **Readability** of generated unit tests is paramount

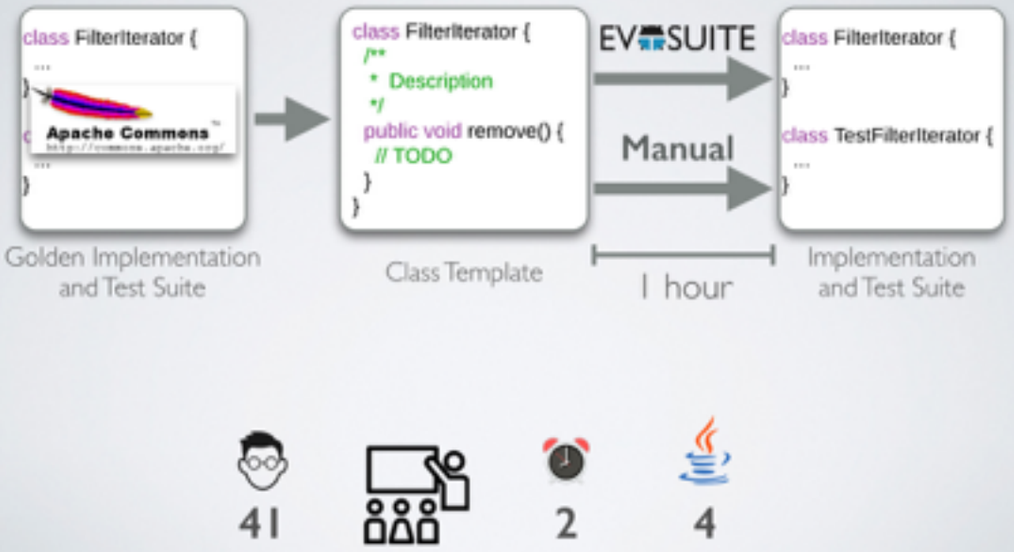
- **Integration** into development environments **must** be improved

- **Education/Best practices**: Developers **do not** know how to best use automated test generation tools!

CONTROLLED EXPERIMENT



CONTROLLED EXPERIMENT

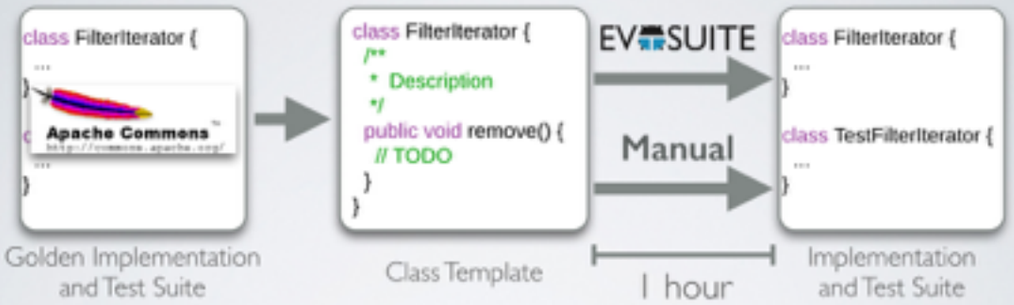


THINK ALOUD OBSERVATIONS

K. A. Ericsson and H. A. Simon, *Protocol Analysis: Verbal Reports as Data (revised edition)*. MIT Press, 1993.

J. Hughes and S. Parkes, "Trends in the use of verbal protocol analysis in software engineering research," *Behaviour and Information Technology*, vol. 22, no. 2, pp. 127–140, 2003.

CONTROLLED EXPERIMENT



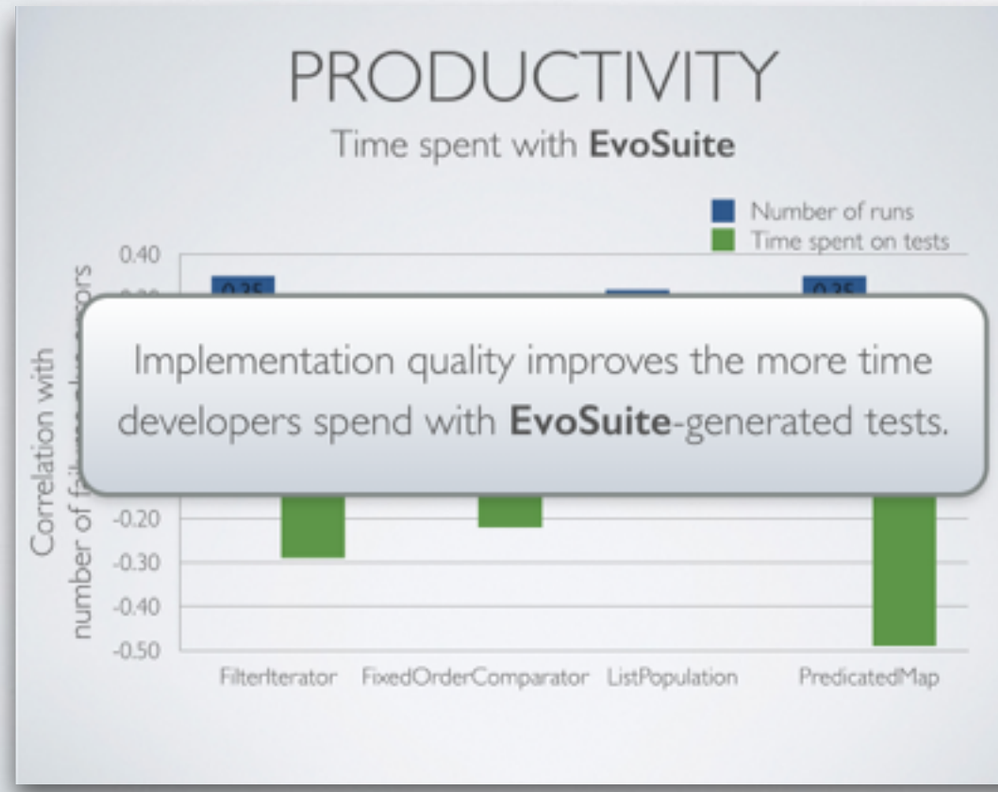
THINK ALOUD OBSERVATIONS

K. A. Ericsson and H. A. Simon, *Protocol Analysis: Verbal Reports as Data (revised edition)*. MIT Press, 1993.

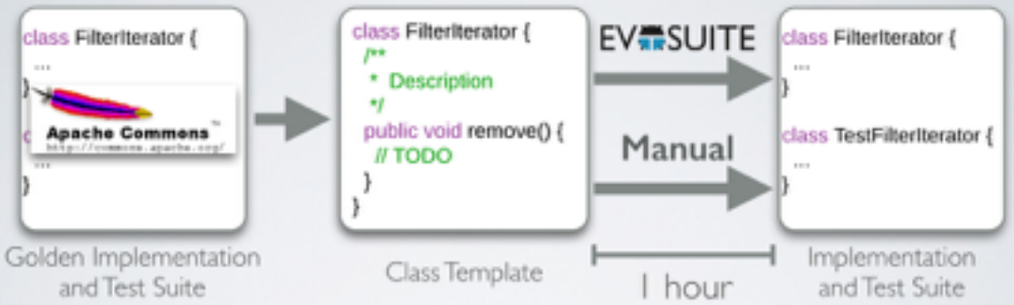
J. Hughes and S. Parkes, "Trends in the use of verbal protocol analysis in software engineering research," *Behaviour and Information Technology*, vol. 22, no. 2, pp. 127–140, 2003.

PRODUCTIVITY

Time spent with **EvoSuite**



CONTROLLED EXPERIMENT



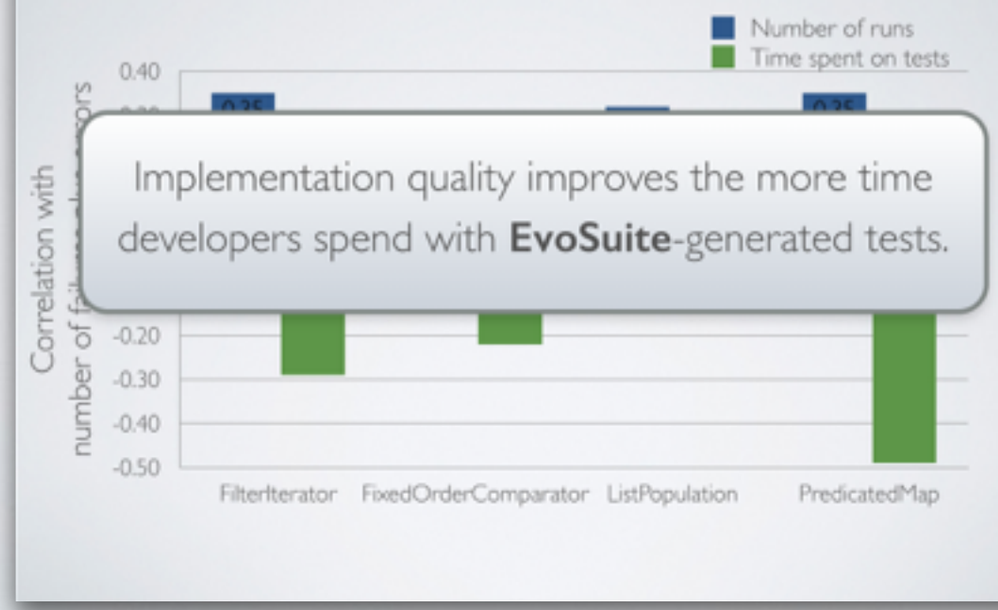
THINK ALOUD OBSERVATIONS

K. A. Ericsson and H. A. Simon, *Protocol Analysis: Verbal Reports as Data (revised edition)*. MIT Press, 1993.

J. Hughes and S. Parkes, "Trends in the use of verbal protocol analysis in software engineering research," *Behaviour and Information Technology*, vol. 22, no. 2, pp. 127–140, 2003.

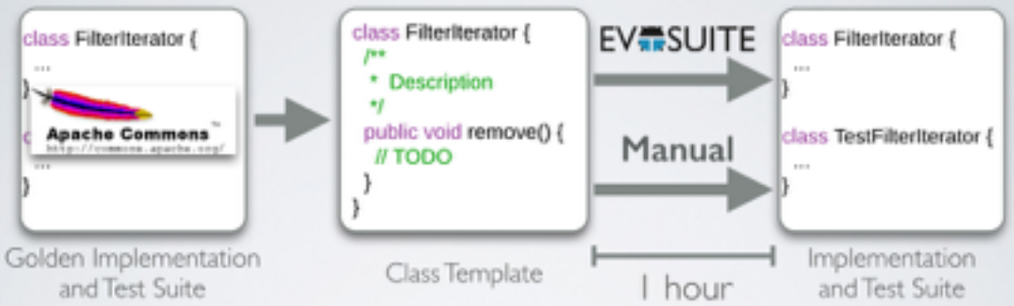
PRODUCTIVITY

Time spent with **EvoSuite**



"Coverage is easy to assess because it is a number, while readability is a very non-tangible property…

CONTROLLED EXPERIMENT



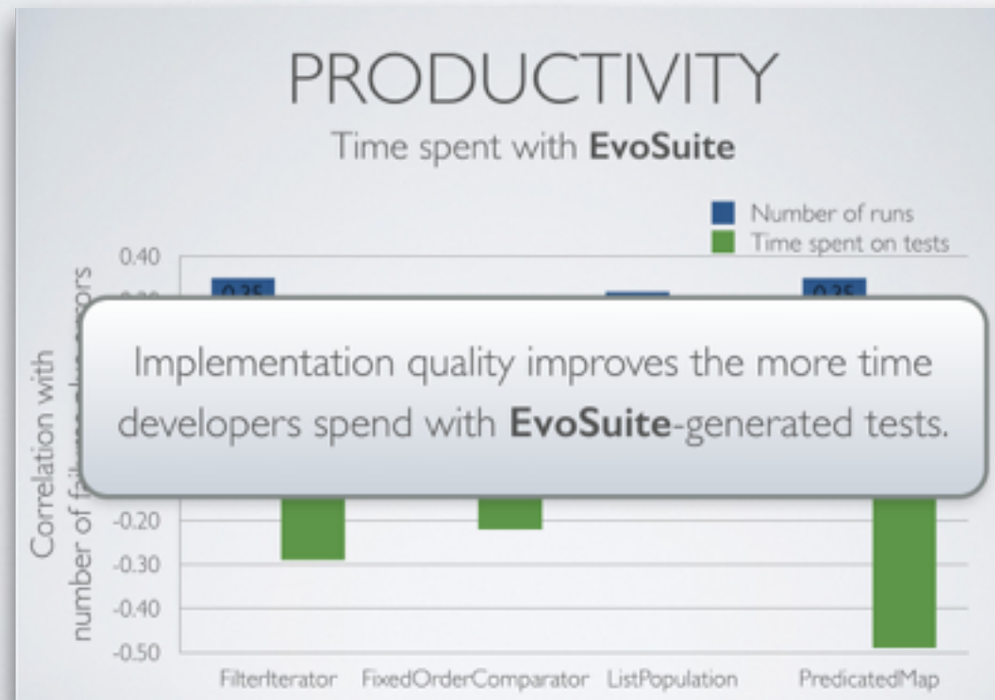
THINK ALOUD OBSERVATIONS

K. A. Ericsson and H. A. Simon, *Protocol Analysis: Verbal Reports as Data (revised edition)*. MIT Press, 1993.

J. Hughes and S. Parkes, "Trends in the use of verbal protocol analysis in software engineering research," *Behaviour and Information Technology*, vol. 22, no. 2, pp. 127–140, 2003.

PRODUCTIVITY

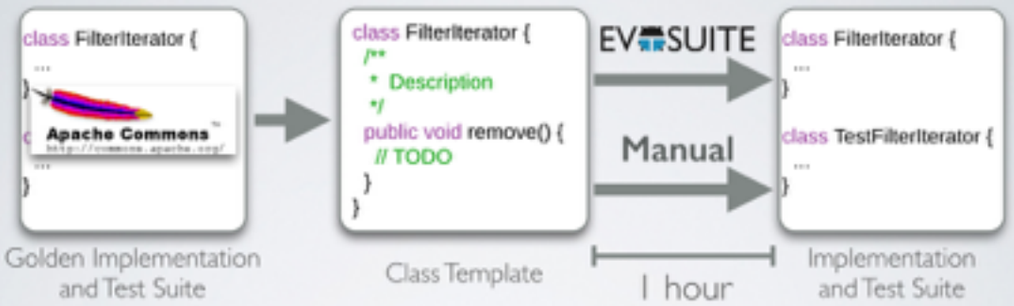
Time spent with **EvoSuite**



"Coverage is easy to assess because it is a number, while readability is a very non-tangible property…

… What is readable to me may not be readable to you. It is readable to me just because I spent the last hour and a half doing this."

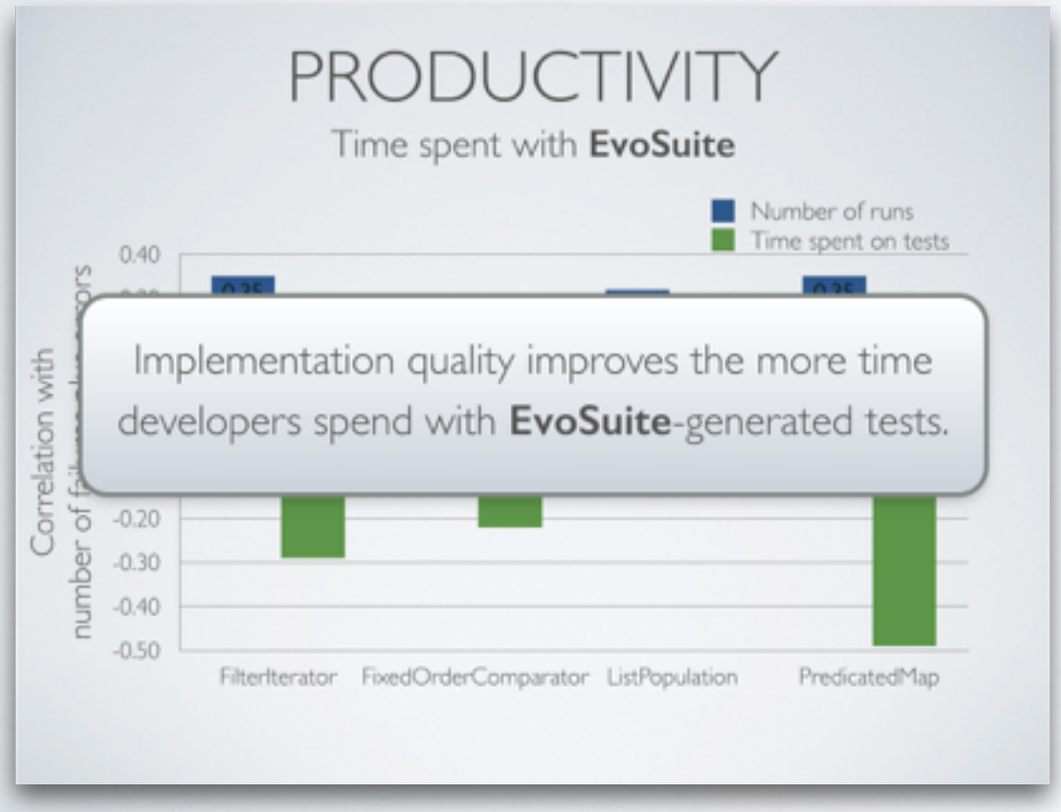
CONTROLLED EXPERIMENT



THINK ALOUD OBSERVATIONS

K. A. Ericsson and H. A. Simon, *Protocol Analysis: Verbal Reports as Data (revised edition)*. MIT Press, 1993.

J. Hughes and S. Parkes, "Trends in the use of verbal protocol analysis in software engineering research," *Behaviour and Information Technology*, vol. 22, no. 2, pp. 127–140, 2003.



"Coverage is easy to assess because it is a number, while readability is a very non-tangible property…

… What is readable to me may not be readable to you. It is readable to me just because I spent the last hour and a half doing this."

—Participant 5

j.rojas@sheffield.ac.uk

www.evosuite.org/study-2014/

j.rojas@sheffield.ac.uk