A Comparative Case Study on the Impact of Test-Driven Development on Program Design and Test Coverage

Maria Siniaalto and Pekka Abrahamsson

ESEM, 2007

Irena Pletikosa Cvijikj
Softvare Engineering Seminar
ETH Zurich, 30.03.2010

CONTENT

- Introduction
- Empirical Body of Evidence
- Empirical Results from a Comparative Case Study
- Conclusions

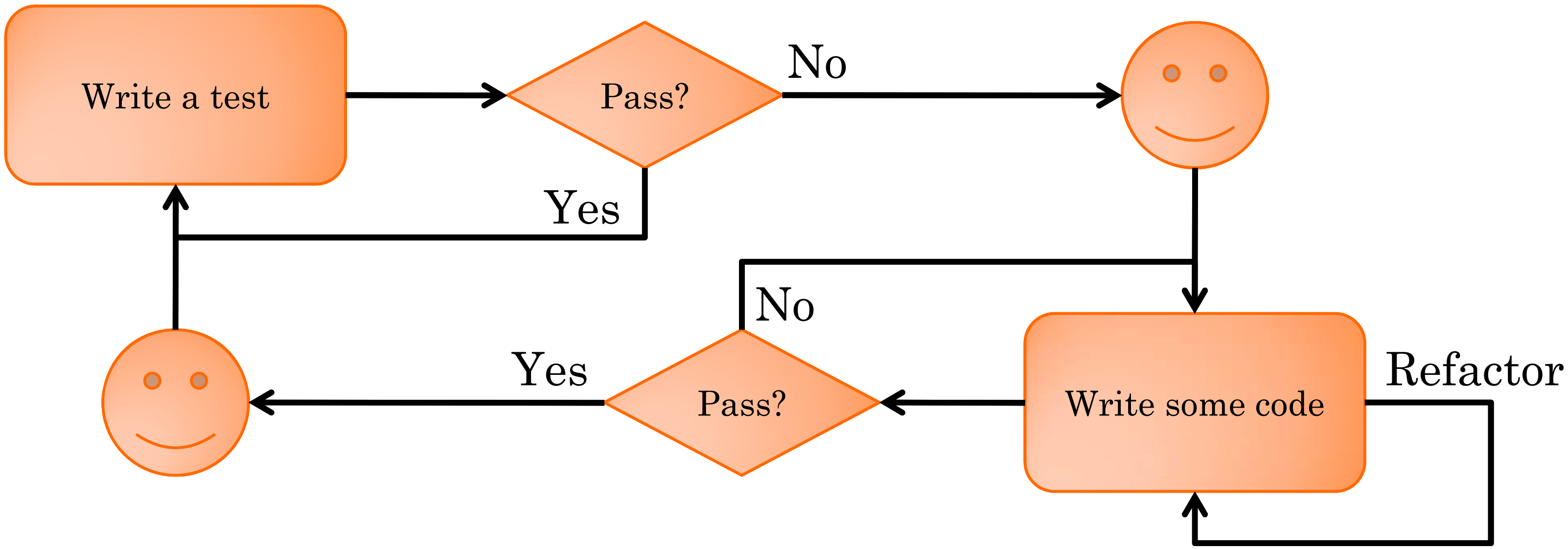
INTRODUCTION

"…TDD is one of the most fundamental practices enabling the development of software in an agile and iterative manner…"

What is Test Driven Development?

TDD is a software development technique based on the repetition of a short development cycle:

- Write a test
- Write the code to pass the test
- Refactor the code for better quality.



WHY TEST DRIVEN DEVELOPMENT?

- Improves test coverage.
- Leads to modularized, flexible, extensible code.
- Enhances developers job satisfaction/confidence.
- Enables simultaneous work on same code.
- Limits number of defects.
- Increases productivity.

MOTIVATION FOR THIS PAPER

Confirm existing claims by:

- Organizing existing body of evidence
- Conducting comparative case study

H1: TDD leads to increased test coverage

H2: TDD improves design quality

EMPIRICAL BODY OF EVIDENCE

"…there are a few studies addressing the impact of TDD on program design and are currently very scarce…"

Clasification of TDD Studies

Type	Subjects	Context	#Studies
Industry	Industrial developers	Real project	4
Semi–industry	Industrial developers	Experimental task	4
	Student developers	Real project	1
Academic	Student developers	Experimental task	7
		Total	16

Summary of Results

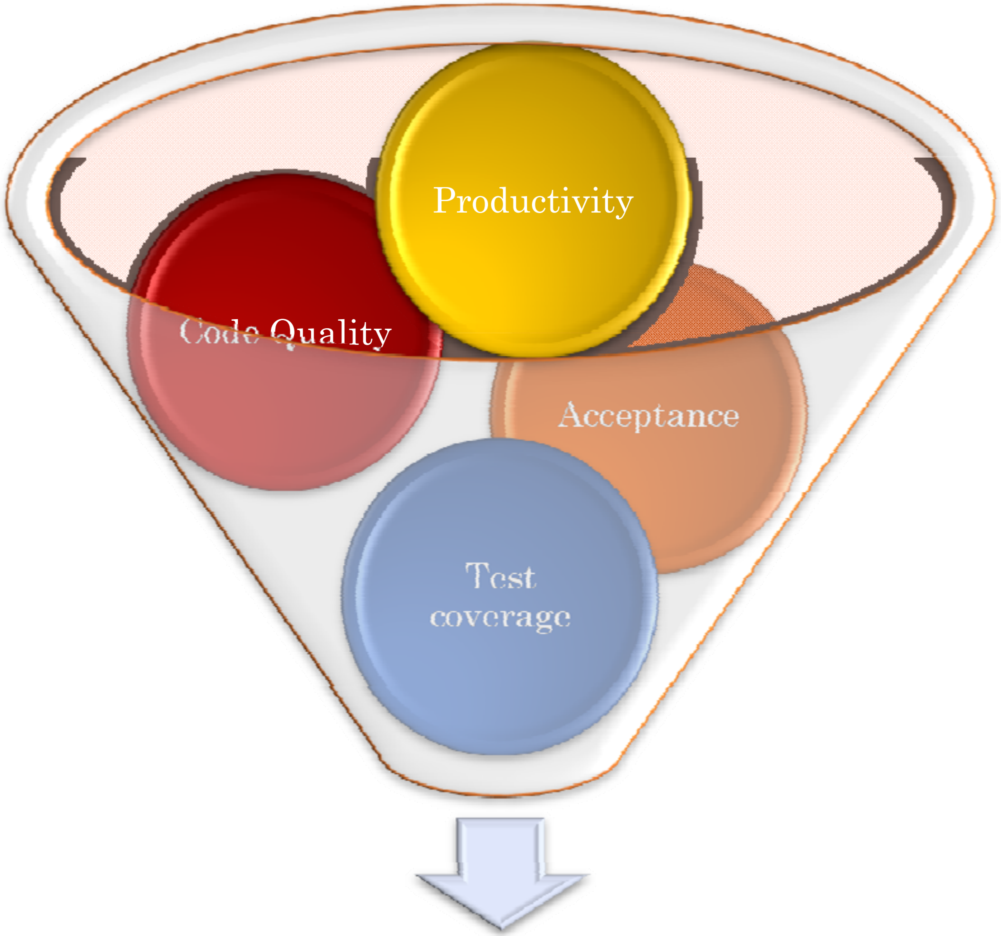
Positive Results

- Increased code quality
- High test coverage
- Increased productivity
- Good acceptance
- Reduced late integration problems

Negative Results

- Reduced code quality
- Low test coverage
- Reduced productivity
- Strong reluctance to adopt
- Greater incidence of failures at the acceptance level

CONCLUSION?



No meaningful conclusion!

Empirical Results from a Comparative Case Study

"...the main goal of this comparative empirical evaluation of TDD is to explore the impact of TDD on program design and test coverage..."

STUDY DESIGN

- Controlled case study
- 3 projects - real products for real customer
- Project duration: 9 weeks (not simultaneous)
- Participants: senior undergraduate students
- Programming language: Java
- Agile development method: Mobile-D

[Ihme, T. and Abrahamsson, P. 2004]

Case Projects Summary

	Project 1	Project 2	Project 3
# of developers	4*	5*	4**
Dev. technique	Test-last	Test-last	TDD
Application type	Web	Mobile	Web
Product concept	Research data management	Stock market browser	PM tool
Product size (LOC)	7700	7000	5800

* All participant had some industrial experience

** Only one participant worked in industry before

RESULTS

- Coupling Between Object Classes
- Lack of Cohesion in Methods
- Test Coverage

OO Metrics Definitions

Coupling Between Objects is…

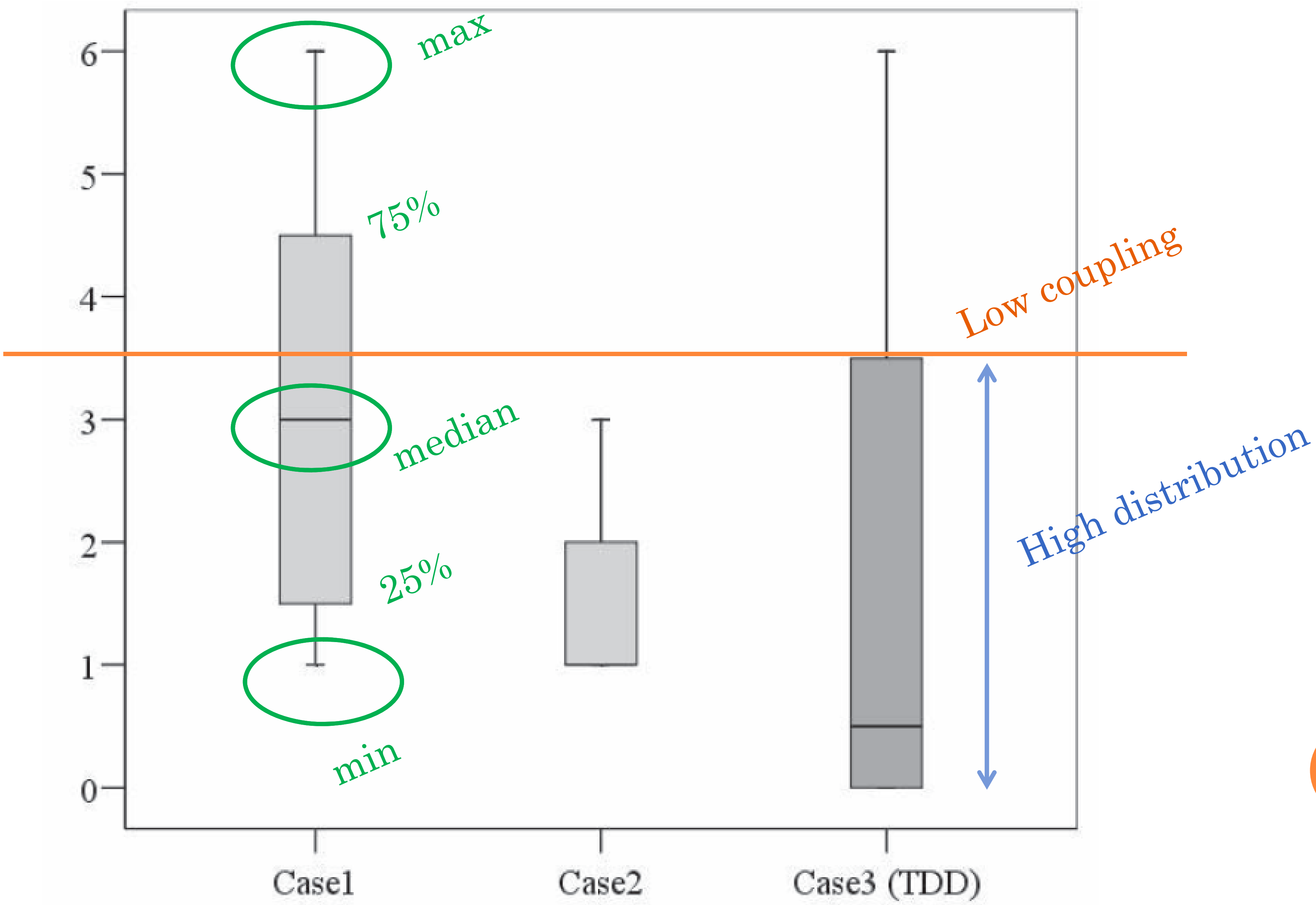
- …the count of the classes to which this class is coupled. Two classes are coupled when methods declared in one class use methods or instance variables of the other class

Lack of Cohesion in Methods is…

- …the number of different methods within a class that reference a given instance variable.

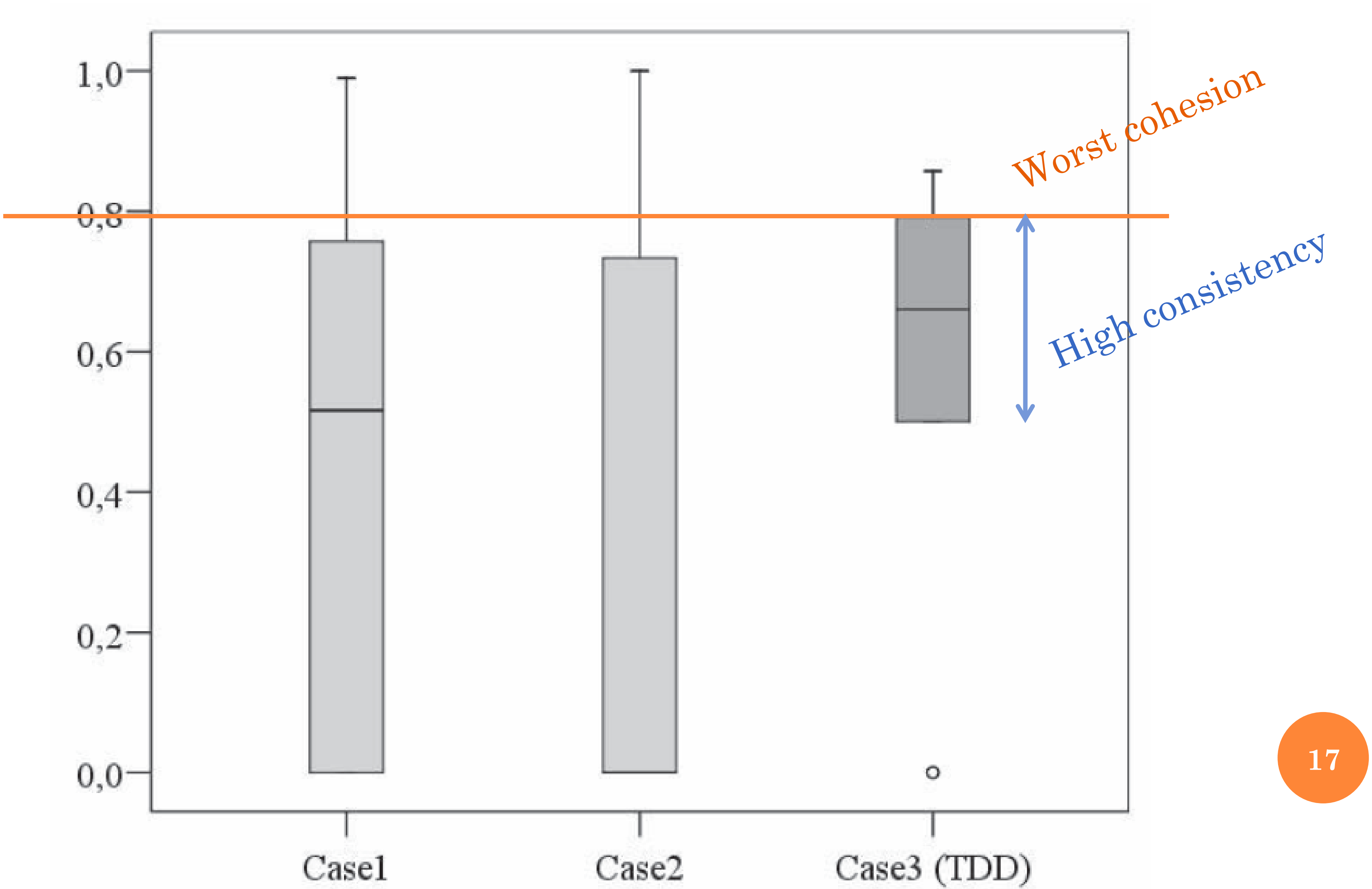
[Chidamber and Kemerer 1994, Henderson-Sellers 1996]

CBO Results

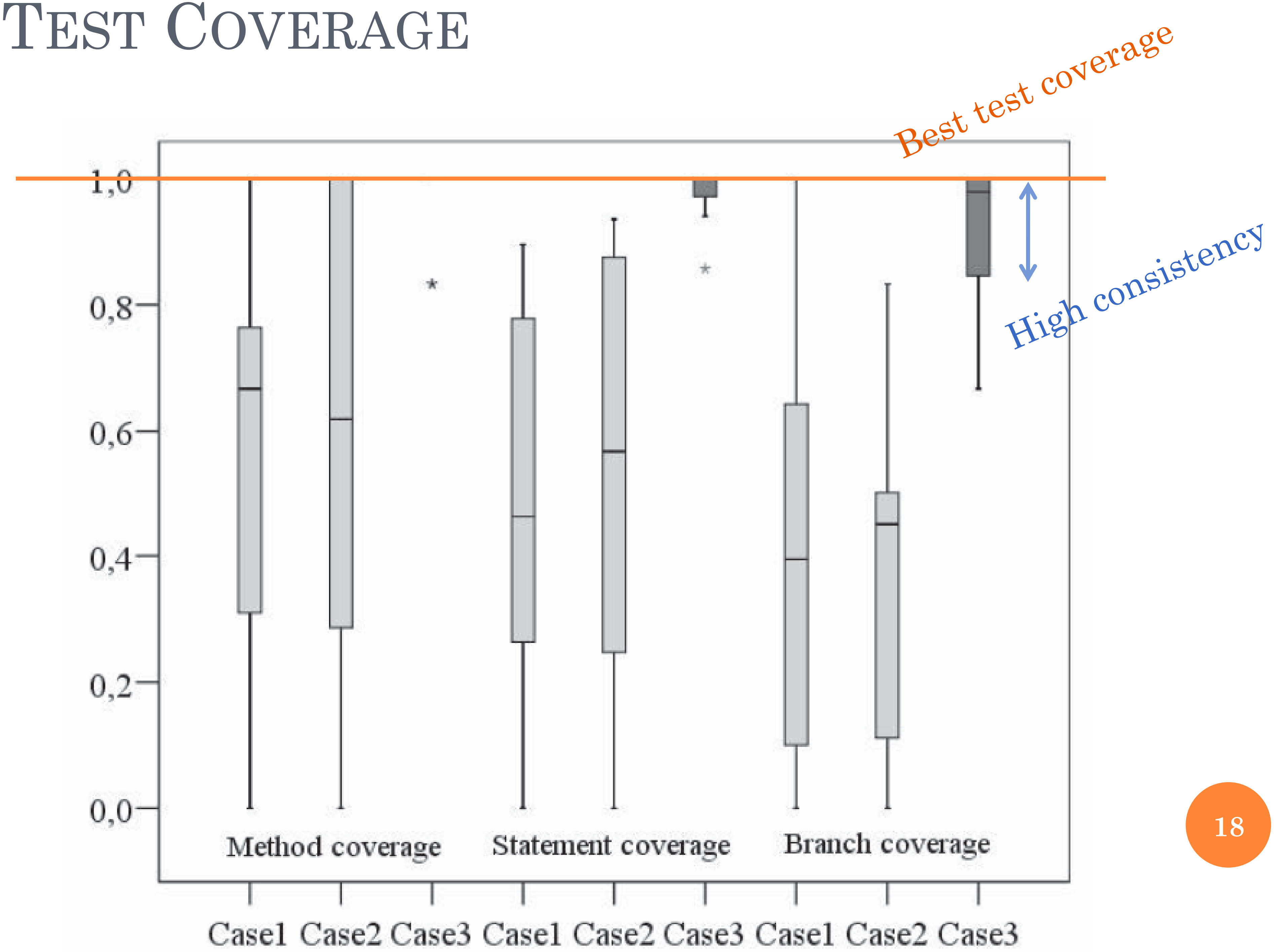


6
5
4
3
2
1
0
max
75%
median
25%
min
Low coupling
High distribution
Case1
Case2
Case3 (TDD)

LCOM Results

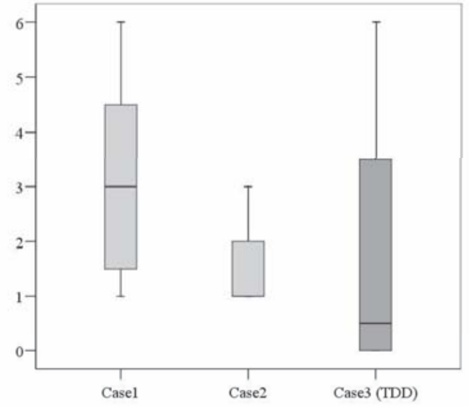


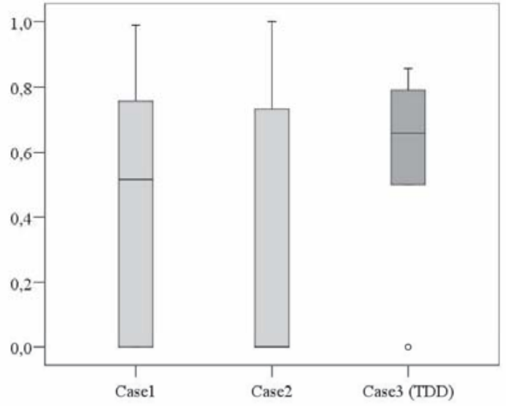
Test Coverage

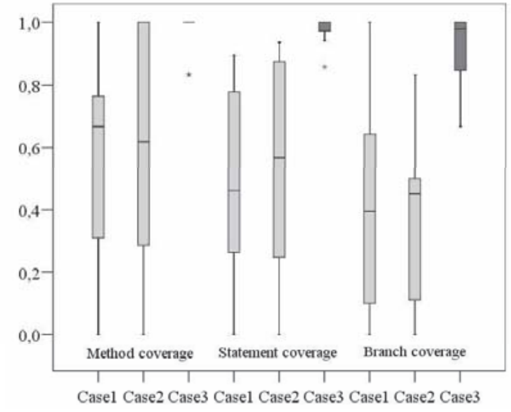


Summary of Results

- WMC, DIT, NOC, RFC: no significant difference
- CBO: no conclusion can be made
- LCOM: experience in TDD usage is required
- Test coverage: significant increase
- Productivity: no effect







THREATS TO VALIDITY

Differences in:

- programming experience,
- level of complexity for projects,
- size of the project, and
- distribution of the work.

Non-TDD developers were…

- encouraged to write tests, but…
- …not informed about the test coverage measurement.

Use of students as study subjects.

Conclusions

"…the results presented in this paper are important as they contribute to the gradual build up of empirical evidence on software engineering innovations…"

CONCLUSIONS

- ❗ Claims that TDD leads to more loosely coupled objects can not be confirmed.
- ❗ TDD does not automatically result in highly cohesive code.
- ❗ TDD leads to significantly increased test coverage.

APPENDIX

Criticisms for TDD

- Does not compensate the lack of up-front design.
- Not suitable for security and multithreaded applications.
- Rapid changes cause expensive breakage in tests.
- Lack of skills produce inadequate test coverage.
- Tests become part of the maintenance overhead.
- May bring a false sense of security.

Existing Body of Knowledge

Details on Previous Studies

TDD in Industrial Settings

Reference	# of subjects	Study focus/Comparison
Bhat and Nagappan	2 projects, 11-14 developers	Non-TDT
Lui and Chan	2 teams	Traditional test-last
Williams et al and Maximillien and Williams	9 developers	Ad-hoc unit testing
Damm et al.	2 projects	Specialized testing tool

Results in Industrial Settings

Positive Results

- Increased code quality
- Increased test coverage
- Tests used as auto documentation
- Improved task estimation
- Improved process tracking
- Defect rate reduction
- Decreased defect repair time
- Decreased project lead time
- Daily integration reduces late integration problems

Negative Results

- Increased development time/reduced productivity

TDD IN SEMI-INDUSTRIAL SETTINGS

Reference	# of subjects	Study focus/Comparison
Canfora et al.	28 developers	Traditional test-last
Müller	5 TDD and 3 conventional projects	Traditional projects
George and Williams	24 developers	Traditional test-last
Geras at al.	14 developers	Traditional test-last
Abrahamsson et al.	4 developers	Exploratory data

Results in Semi-Industrial Settings

Positive Results

- Better performance predictability
- Improved code quality
- Increased test coverage
- Increased number of tests
- More frequent test running

Negative Results

- Increased development time/reduced productivity
- Greater incidence of failures at the acceptance level
- Strong reluctance to adopt TDD
- Developers didn’t see the benefits

TDD in Academic Settings

Reference	# of subjects	Study focus/Comparison
Janzen and Saiedian	3 teams	Iterative test-last, no tests
Kaufmann and Janzen	8 developers	Test-last
Müller and Hagner	19 developers	Traditional test-last
Pancur et al.	38 developers	Iterative test-last
Erdogmus et al.	24 developers	Iterative test-last
Steinberg	/	Exploratory data
Edwards	59 developers	Automated tool, no tests

Results in Academic Settings

Positive Results

- Increased productivity
- TDD accepted after trying
- Improved code quality
- Increased confidence
- Better program/requirements understanding
- Fast and correct use of methods
- Reduced debugging and refactoring effort
- Less faults and easier correction
- Increased cohesion / looser coupling

Negative Results

- Low test coverage
- Concerns regarding complexity and coupling
- Lower final reliability on acceptance test
- Reduced code quality
- Adopting TDD was difficult and it was found as not very effective

Object Oriented Metrics

Object-Oriented Metrics

- WMC – Weighted Methods per Class
- DIT – Depth of Inheritance Tree
- NOC – Number Of Children
- CBO – Coupling Between Objects
- RFC – Response for a Class
- LCOM – Lack of Cohesion in Methods

Weighted Methods per Class

- WMC is defined as the sum of the complexities of all methods of a class.
  - The number of methods and the complexity of methods involved is a predictor of how much time and effort is required to develop and maintain the class.
  - The larger the number of methods in a class the greater the potential impact on children, since children will inherit all the methods defined in the class.
  - Classes with large numbers of methods are likely to be more application specific, limiting the possibility of reuse.

[Chidamber and Kemerer 1994]

Depth of Inheritance Tree

- DIT is defined as the maximum length from the node to the root of the tree.
  - The deeper a class is in the hierarchy, the greater the number of methods it is likely to inherit, making it more complex to predict its behavior.
  - Deeper trees constitute greater design complexity, since more methods and classes are involved.
  - The deeper a particular class is in the hierarchy, the greater the potential reuse of inherited methods.

[Chidamber and Kemerer 1994]

Number Of Children

- NOC is defined as the number of immediate subclasses.
  - The greater the number of children, the greater the reuse, since inheritance is a form of reuse.
  - The greater the number of children, the greater the likelihood of improper abstraction of the parent class. If a class has a large number of children, it may be a case of misuse of subclassing.
  - The number of children gives an idea of the potential influence a class has on the design. If a class has a large number of children, it may require more testing of the methods in that class.

[Chidamber and Kemerer 1994]

Coupling Between Objects

- CBO is defined as the count of the classes to which this class is coupled. Two classes are coupled when methods declared in one class use methods or instance variables of the other class.
  - Excessive coupling between object classes is detrimental to modular design and prevents reuse. The more independent a class is, the easier it is to reuse it in another application.
  - In order to improve modularity and promote encapsulation, inter-object class couples should be kept to a minimum. The larger the number of couples, the higher the sensitivity to changes in other parts of the design, and therefore maintenance is more difficult.
  - A measure of coupling is useful to determine how complex the testing of various parts of a design are likely to be. The higher the inter-object class coupling, the more rigorous the testing needs to be.

[Chidamber and Kemerer 1994]

Response for a Class

- RFC is defined as number of methods in the set of all methods that can be invoked in response to a message sent to an object of a class.
    - If a large number of methods can be invoked in response to a message, the testing and debugging of the class becomes more complicated since it requires a greater level of understanding on the part of the tester.
    - The larger the number of methods that can be invoked from a class, the greater the complexity of the class.
    - A worst case value for possible responses will assist in appropriate allocation of testing time.

[Chidamber and Kemerer 1994]

Lack of Cohesion in Methods (1)

- LCOM is defined as the number of different methods within a class that reference a given instance variable.
  - Cohesiveness of methods within a class is desirable, since it promotes encapsulation.
  - Lack of cohesion implies classes should probably be split into two or more subclasses.
  - Any measure of disparateness of methods helps identify flaws in the design of classes.
  - Low cohesion increases complexity, thereby increasing the likelihood of errors during the development process.

[Chidamber and Kemerer 1994]

Lack of Cohesion in Methods (2)

- LCOM is defined as the number of different methods within a class that reference a given instance variable.

- Henderson-Sellers algorithm produces answers in the range 0 to 1, with the value zero representing perfect cohesion and with value one presenting extreme lack of cohesion.

- LCOM = (m - sum(mA)/a) / (m-1)
  - m: number of methods in a class
  - a: number of attributes in a class.
  - mA: number of methods that access the attribute a.
  - sum(mA): sum of all mA over all the attributes in the class.

[Henderson -Sellers 1996]