An Empirical Evaluation of the Impact of Test-Driven Development on Software Quality

David Janzen (djanzen@ku.edu)
University of Kansas

Acknowledgements

- Committee Members
  - Dr. Hossein Saiedian, Advisor
  - Dr. Arvin Agah
  - Dr. Perry Alexander
  - Dr. John Gauch
  - Dr. Carey Johnson
- EECS Department

Organization

- Problem Statement
- Background
- Research Methodology
- Evaluation and Results
- Conclusions and Future Work

Organization

- Problem Statement
  - Introduction
  - Context
  - Research Proposal
  - Results Overview
- Background
- Research Methodology
- Evaluation and Results
- Conclusions and Future Work

Introduction

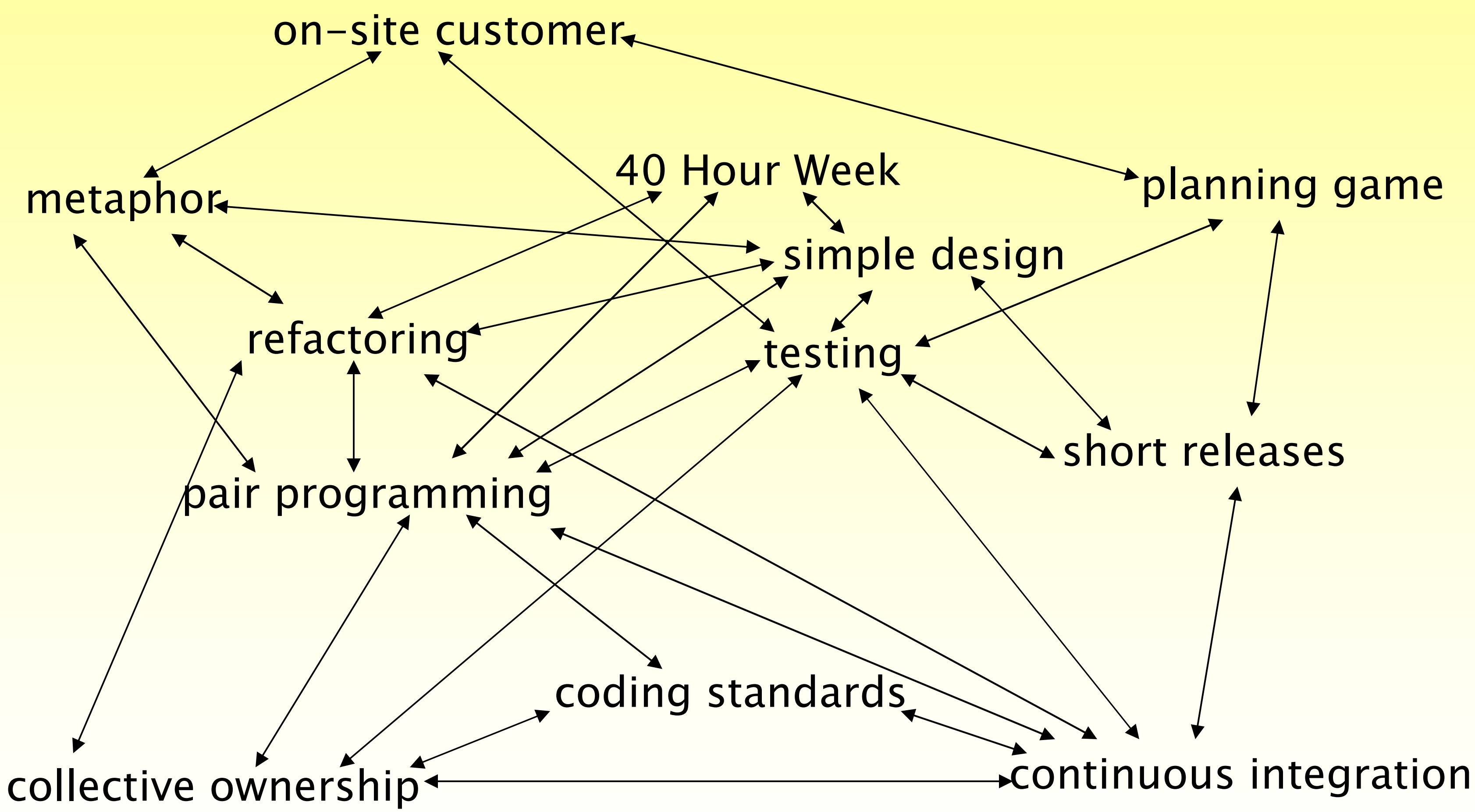
- Observation
  - Test-driven development is a popular new method for designing and testing software
- Problem
  - No empirical evidence of TDD efficacy as a design methodology
- Opportunity
  - Poor testing is a significant contributor to software crisis
  - Can TDD improve both design and testing, resulting in better software?

Mainstream Software Development Milestones

Era	Language	Process
1960's	Assembly	
1970's	Structured	Waterfall
1980's	Object-Oriented	OOA&D
1990's	Object-Oriented	UML/CMM/RUP
2000's	Object-Oriented	Agile (XP)[1]

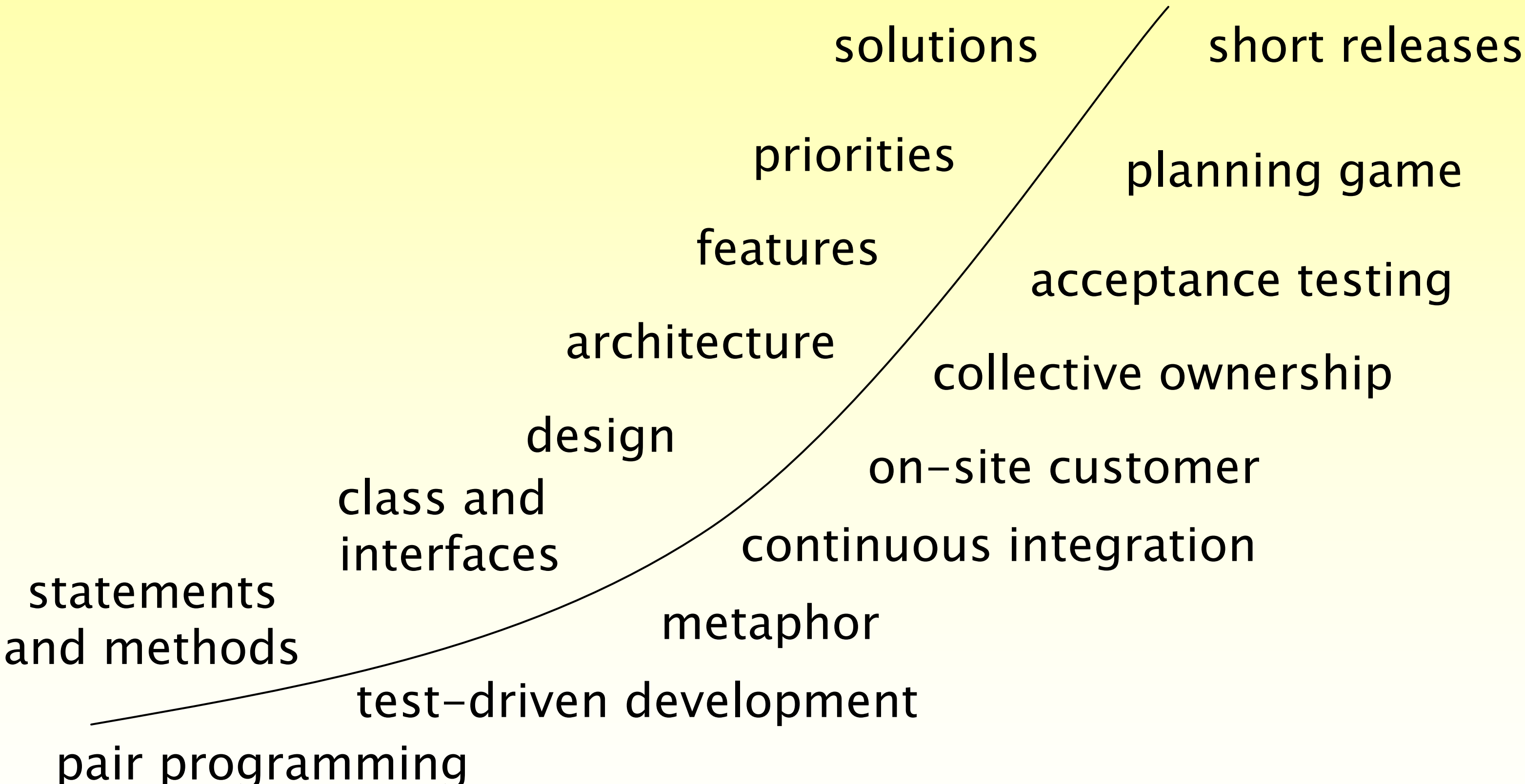
1. Rajlich, "Changing the Paradigm of Software Engineering", *Communications of the ACM,* 2006

XP Practice Coupling[1]



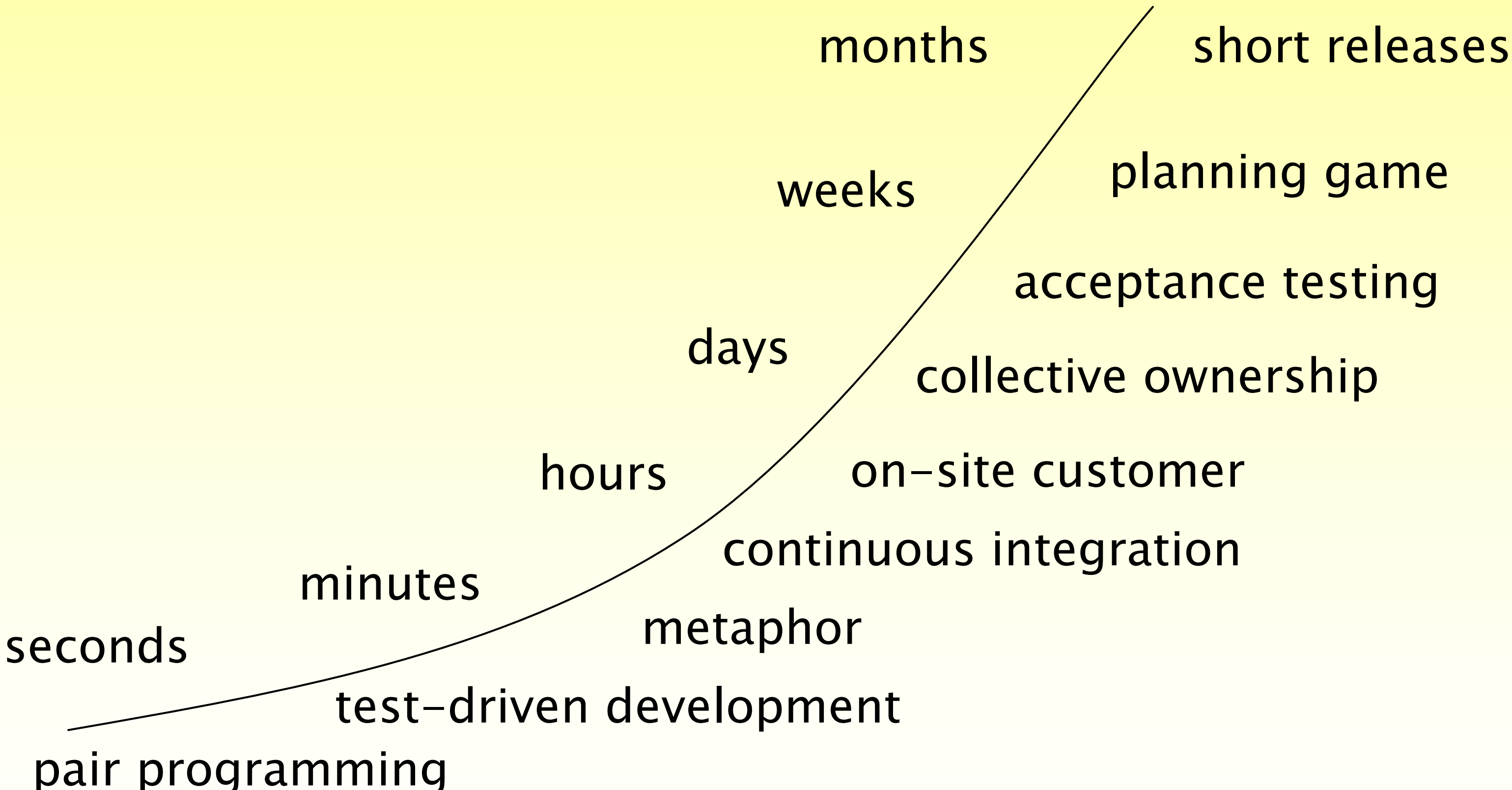
1. Beck, Extreme Programming Explained: Embrace Change, 2000

XP Scale-Defined Practices[1]



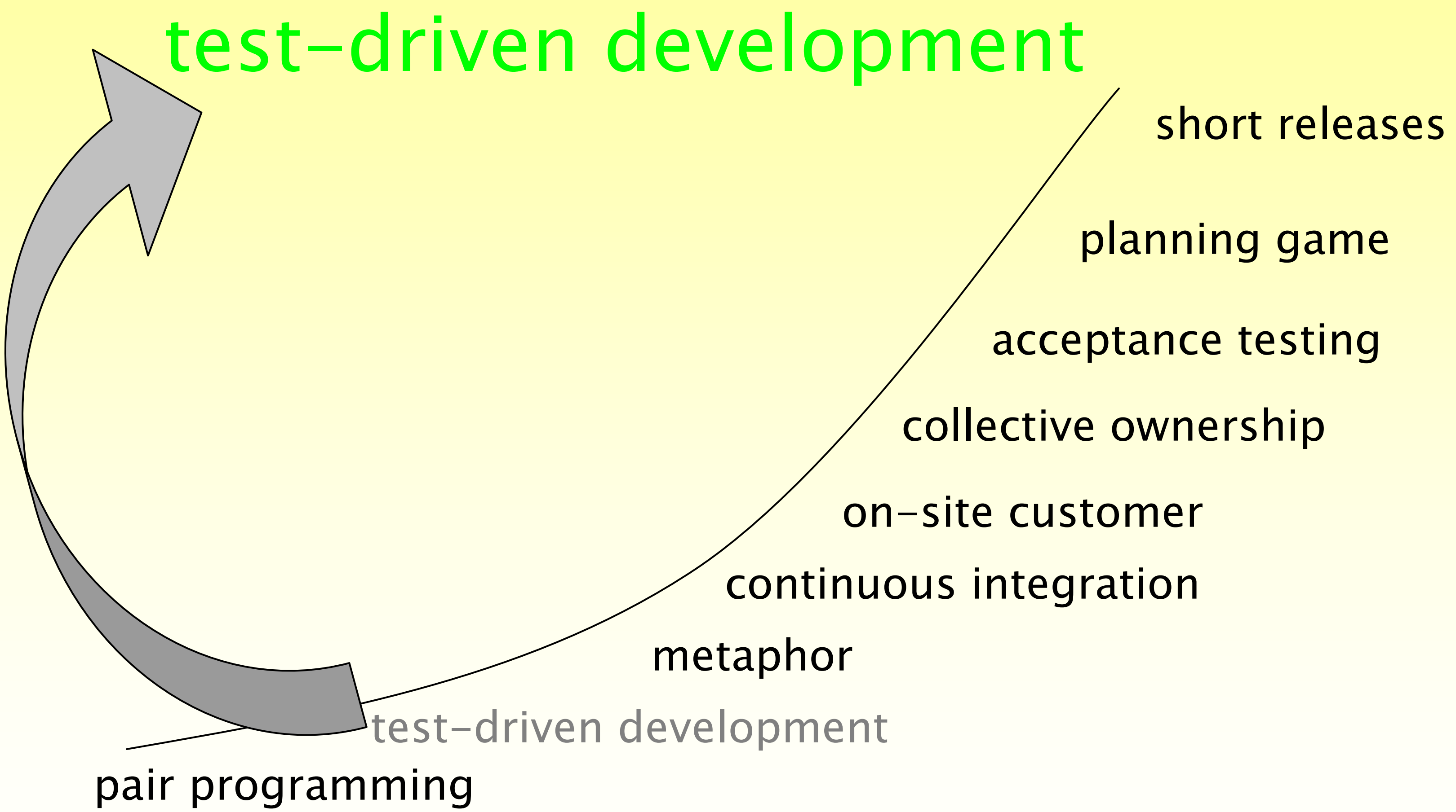
1. Vanderburg, "A Simple Model of Agile Software Processes", *OOPSLA,* 2005

XP Practices and Time Scales[1]



1. Vanderburg, "A Simple Model of Agile Software Processes", *OOPSLA,* 2005

Extracting TDD from XP



Research Objective

- Conduct empirical studies examining how TDD affects *testing* and *internal design quality*
- Controlled experiments in academic courses
  - At all levels to gauge optimal introduction point
- Semi-controlled experiments and case study in Fortune 500 companies
  - Conduct small experiment in training course
  - Compare same team in transition to TDD
  - Compare different teams/projects
- Longitudinal studies examine voluntary TDD adoption in subsequent projects

Summary of Results

- TDD improves internal quality aspects
  - Software is smaller
  - Software is less complex and more elegant
- TDD improves testing
  - Increased coverage, more test cases
  - Fewer defects
- Programmer opinions
  - Mature programmers prefer TDD after trying both approaches

Additional Research Results

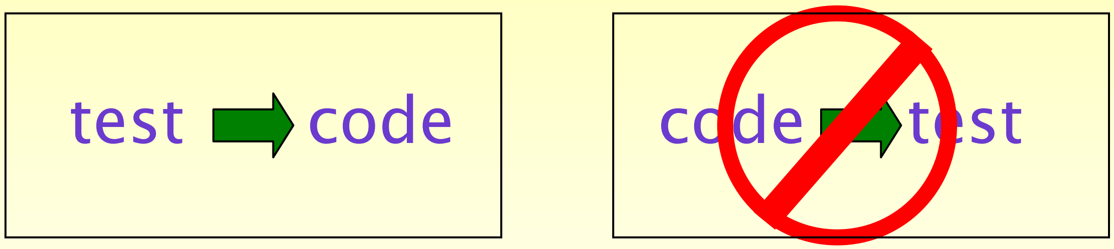
- Test-Driven Learning
  - A pedagogical approach that integrates TDD instruction at all levels with minimal cost
- Framework for future studies
  - Results establish benchmark
  - Methods, tools, and artifacts provided for replicated studies

Organization

- Problem Statement
- Background
  - TDD Overview
  - Related Research
- Research Methodology
- Evaluation and Results
- Conclusions and Future Work

Test-Driven Development (TDD)

- Disciplined development approach
- Emerged from agile methods (XP)
- Reverses traditional micro workflow

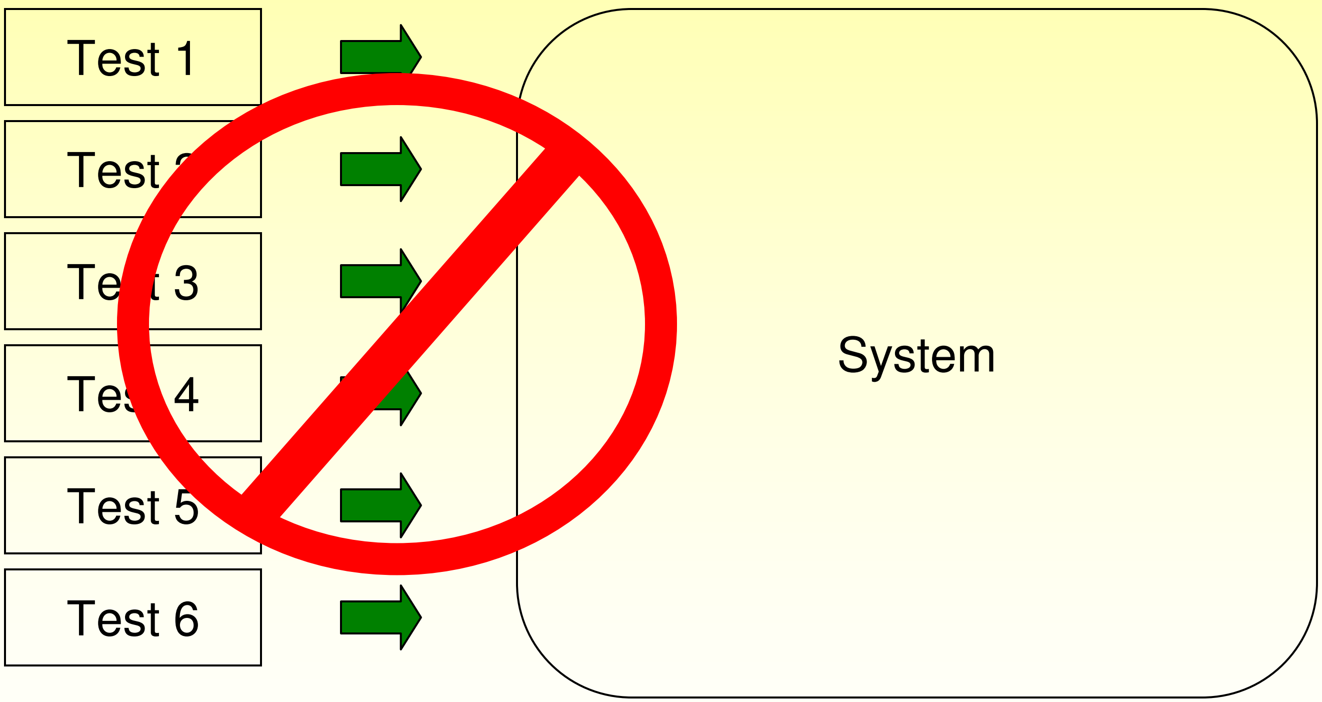


- More about design than testing[1]
- Primarily focuses on unit tests
- Supported by automated testing frameworks such as JUnit

1. Beck, “Aim, Fire”, *IEEE Software,* 2001

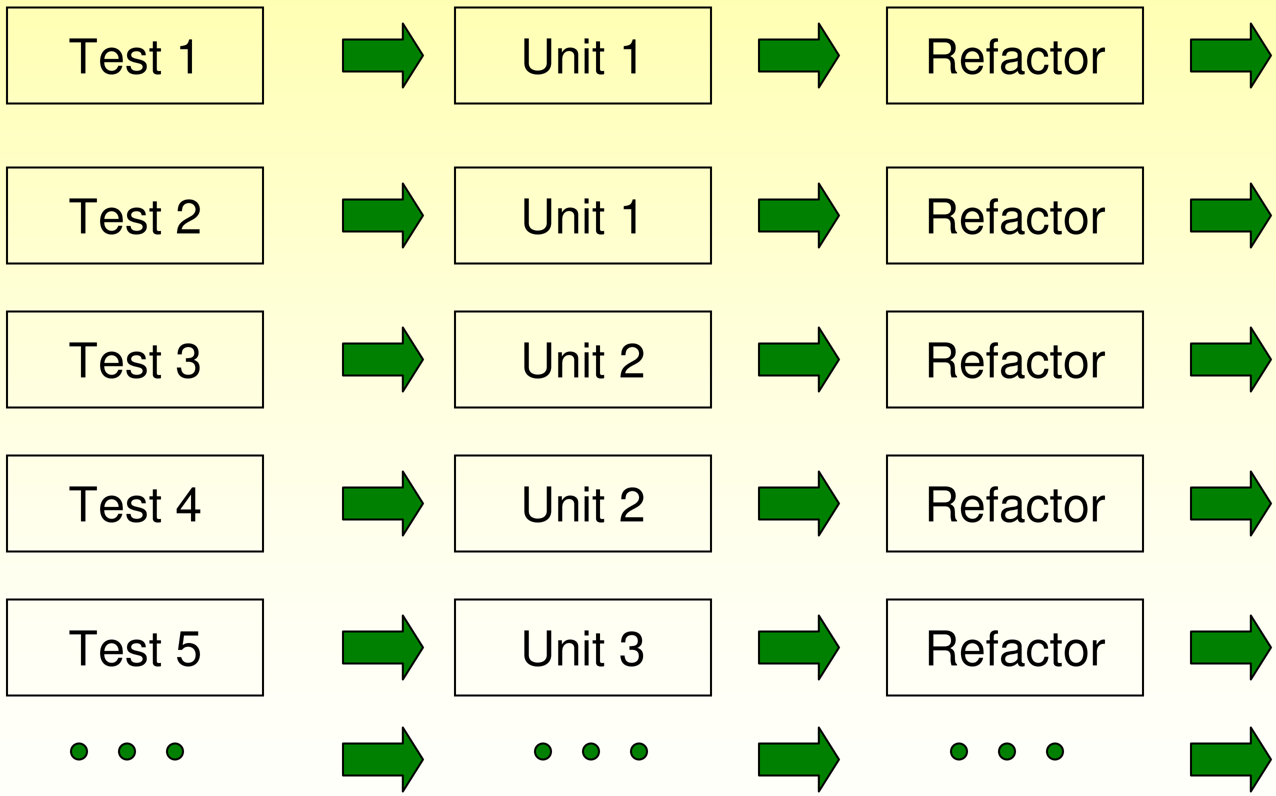
TDD Misconception

- TDD does not mean “write all the tests, then build a system that passes the tests”



TDD Clarified

- TDD means "write one test, write code to pass that test, refactor, and repeat"



Related TDD Studies in Industry

Study[a]	Type	Number of companies	Number of programmers	Quality effects	Productivity effects
George[1] (NCSU 2004)	CE	3	24	TDD passed 18% more tests	TDD took 16% longer[b]
Maximillien[2] (NCSU 2003)	CS	1	9	50% reduction in defect density	Minimal impact
Williams[3] (NCSU 2003)	CS	1	9	40% reduction in defect density	No change

[a] Studies reported less time spent debugging with TDD

[b] TDD group wrote many more tests than control group

1. George and Williams, "A Structured Experiment of Test-Driven Development", *Info & Sw Tech*, 2004
2. Maximilien and Williams, "Assessing Test-Driven Development at IBM", *ICSE*, 2003
3. Williams et. al., "Test-driven development as a defect-reduction practice", *Sw Rel. Eng*, 2003

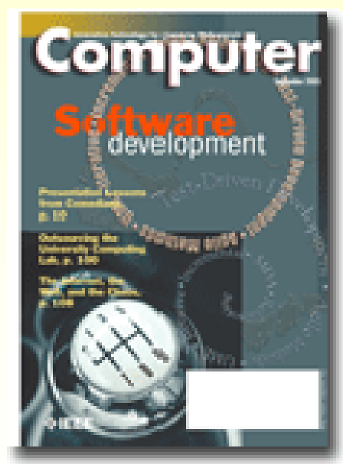
Related TDD Studies in Academia

Study	Type	# programmers	Quality effects	Productivity effects
Edwards[1] (Virginia Tech 2003)	CE	59	54% fewer defects	n/a
Kaufmann[2] (Bethel 2003)	CE	8	improved information flow	50% improvement
Müller[3] (Karlsruhe 2002)	CE	19	no change, but better reuse	no change
Pančur[4] (Ljubljana 2003)	CE	38	no change	no change
Erdogmus[5] (Torino 2005)	CE	35	no change	28% improvement

1. Edwards, “Rethinking Computer Science Education from a Test-first Perspective”, *OOPSLA*, 2003
2. Kaufmann and Janzen, “Implications of test-driven development: a pilot study”, *OOPSLA*, 2003
3. Muller and Hagner, “Experiment About Test-First Programming”, *IEEE Software*, 2002
4. Pancur et. al., “Towards Empirical Evaluation of Test-Driven Development in a University Environment”, *Eurocon*, 2003
5. **Erdogmus, “On the Effectiveness of Test-first Approach to Programming”, *IEEE Trans on SE*, 2005**

Background and Related Work Published in IEEE Computer

- D. Janzen and H. Saiedian, Test-Driven Development: Concepts, Taxonomy and Future Directions, *IEEE Computer*, 38(9), 2005
- Background study, challenges, clarifying TDD as design approach, need for the research
- Cover feature

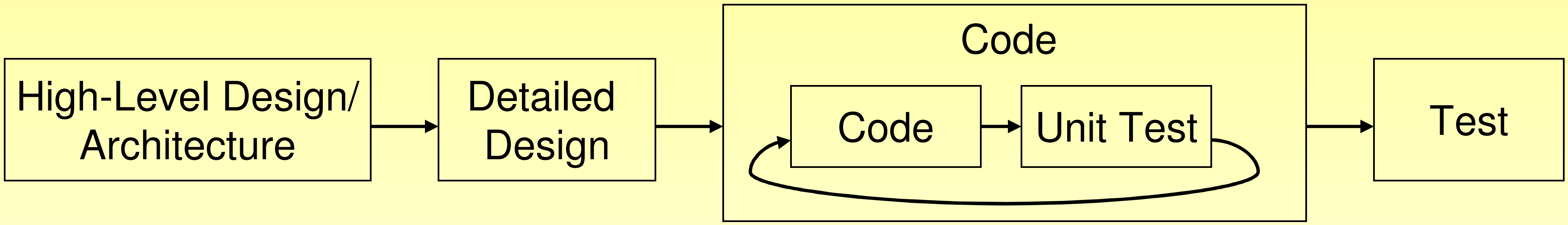


Organization

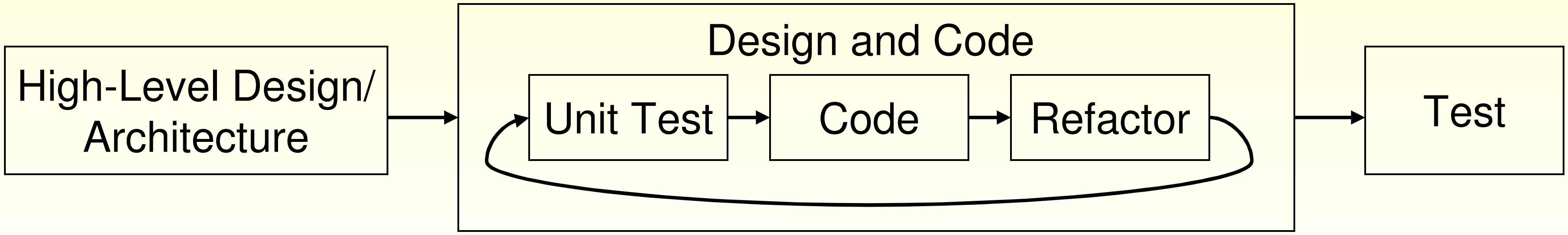
- Problem Statement
- Background
- Research Methodology
  - TDD and Design
  - Hypotheses
  - Experiment Design
  - Metrics
- Evaluation and Results
- Conclusions and Future Work

TDD Process Flow

- Traditional test-last process



- TDD process



TDD is about Design

```
public class TestBank extends TestCase {
    public void testCreateBankEmpty() {
      Bank b = new Bank();
      assertEquals(b.getNumAccounts(), 0);
    }
}
```

Design decisions

- TDD gives early focus to a unit's
  - Interface: How will I use it?
  - Behavior: What does it do?
  - Reuse: Multiple clients (test and source)
  - Coupling: Units need to be tested in isolation
  - Cohesion: Testable units have one purpose

Hypothesis

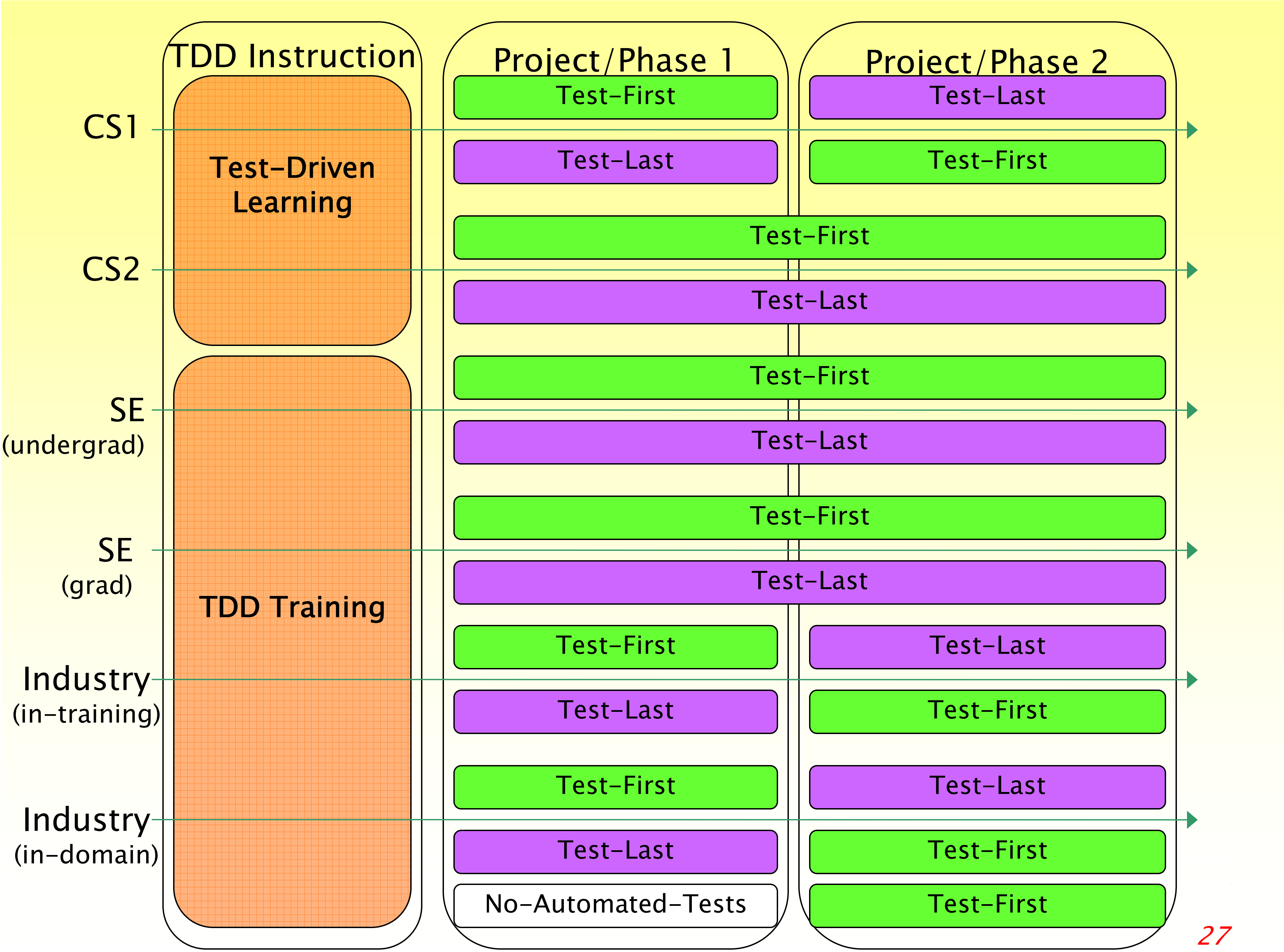
- Null hypothesis
  - Software constructed using the test-driven development approach will have similar quality at higher cost to develop when compared to software constructed with a traditional test-last approach
- Independent variable
  - Use of test-driven (test-first) versus test-last development
- Dependent variables
  - Software quality
  - Degree of testing
  - Software cost (programmer productivity)
- Additional dependent variables observed
  - Student performance on related assessments
  - Subsequent voluntary usage of TDD

Formal Hypotheses: Internal Quality and Testing

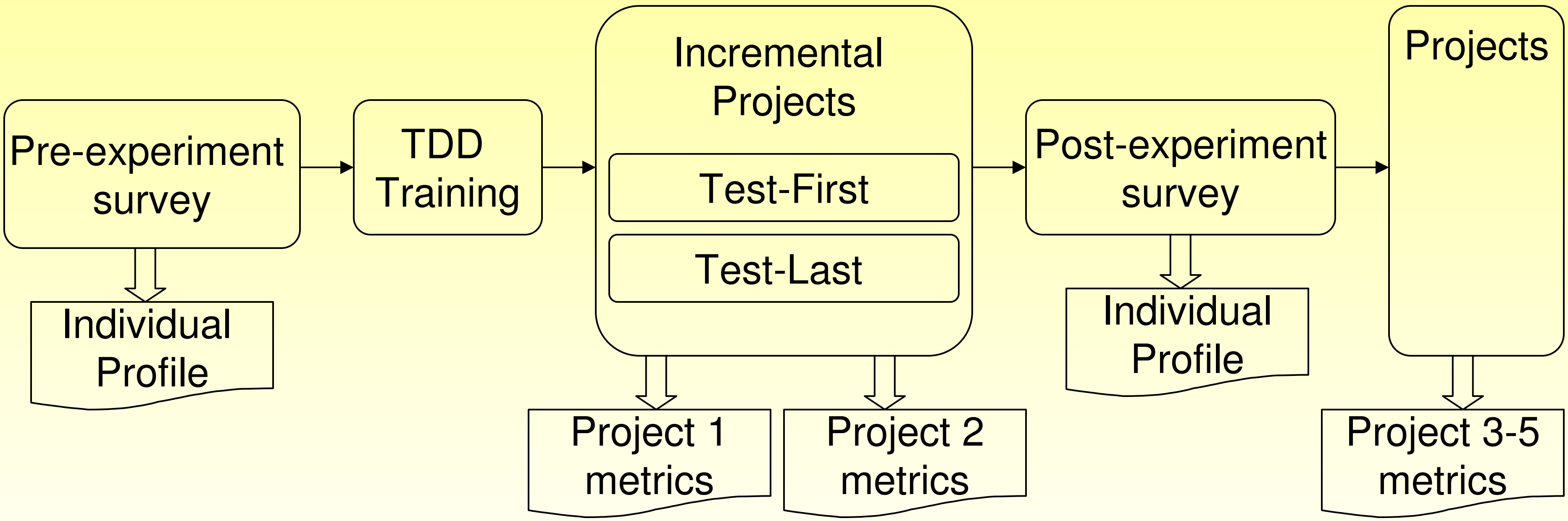
Name	Null Hypothesis	Alternative Hypothesis
Q1	$IntQlty_{TF} = IntQlty_{TL}$	$IntQlty_{TF} > IntQlty_{TL}$ Test-first code has higher internal quality
Q2	$IntQlty\|Tested_{TF} = IntQlty\|Not\text{-}Tested_{TF}$	$IntQlty\|Tested_{TF} > IntQlty\|Not\text{-}Tested_{TF}$ Test-first code covered by tests has higher internal quality
T1	$\#Tests_{TF} = \#Tests_{TL}$	$\#Tests_{TF} > \#Tests_{TL}$ Test-first programmers write more tests
T2	$TestCov_{TF} = TestCov_{TL}$	$TestCov_{TF} > TestCov_{TL}$ Test-first programmers write tests with better code coverage

Formal Hypotheses: Productivity and Programmer Opinions

Name	Null Hypothesis	Alternative Hypothesis
P1	$Prod_{TF} = Prod_{TL}$	$Prod_{TF} > Prod_{TL}$ Test-first programmers are more productive
O1	$Op_{TF} = Op_{TL}$	$Op_{TF} > Op_{TL}$ Programmers perceive test-first as better approach
O2	$Op\|TF_{TF} = Op\|TF_{TL}$	$Op\|TF_{TF} > Op\|TF_{TL}$ Programmers who have attempted test-first prefer test-first


TDD Instruction
Project/Phase 1
Project/Phase 2
CS1
CS2
SE
(undergrad)
SE
(grad)
Industry
(in-training)
Industry
(in-domain)
Test-Driven Learning
TDD Training
Test-First
Test-Last
Test-Last
Test-First
Test-First
Test-Last
Test-First
Test-Last
Test-First
Test-Last
Test-First
Test-Last
Test-Last
Test-First
Test-First
Test-Last
Test-Last
Test-First
No-Automated-Tests
Test-First

Sample Experiment Design (CS2)



Test-Driven Learning[1] in CS1/CS2

- Teach testing simply by example

Traditional Approach

```
int sum(int min, int max) {
  int sum = 0;
  for(int i=min;i<=max;i++)
     sum += i;
  return sum;
}
int main() {
  cout << sum(3,7); //should print 25
  cout << sum(-2,2); //should print 0
  cout << sum(-4,-2); //should print -9
}
```

TDL Approach

```
int sum(int min, int max) {
  int sum = 0;
  for(int i=min;i<=max;i++)
     sum += i;
  return sum;
}
void runTests() {
  assert(sum(3,7)==25);
  assert(sum(-2,2)==0);
  assert(sum(-4,-2)==-9);
}
int main() {
  runTests();
}
```

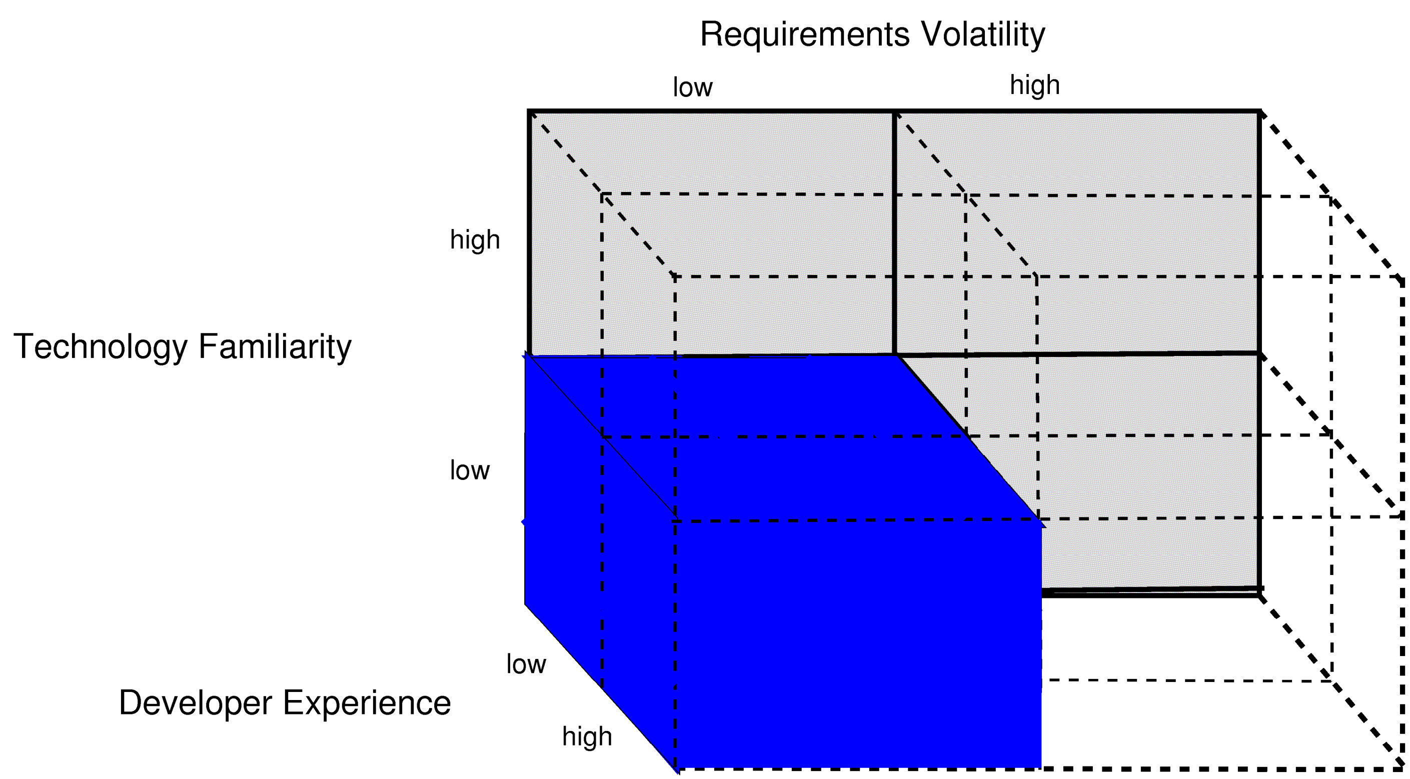
1. Janzen and Saiedian, "Test-Driven Learning: Intrinsic Integration of Testing into the CS/SE Curriculum," *Technical Symposium on Computer Science Education (SIGCSE'06)*, 2006

TDD Training in Industry

- Company agreed to participate in study if author developed and delivered training
  - Six-day course
    - One-day on TDD
    - Remainder on Spring and Hibernate
      - Spring is a lightweight dependency-injection framework that developed based on TDD
      - Hibernate is an object-relational database mapping framework
  - About 500 presentation slides
  - Hands-on lab exercises
  - Delivered on-site in October 2005

Context

- Small Projects (typically less than 3000 LOC)
- C++ and Java
- Mix of text UI, graphical UI, web applications, libraries



Internal Design Quality Measures

- Product Metrics
  - i.e. only look at code (and tests)
- Desirable Attributes
  - Understandability
    - Low complexity, high cohesion, simple
  - Maintainability
    - Low complexity, high cohesion, low coupling
  - Reusability
    - Low complexity, high cohesion, low coupling, inheritance
  - Testability
    - High cohesion, low coupling, high test coverage
- Complexity, coupling, and cohesion are cross-cutting measures

Metrics Collection and Analysis

- Calculated nearly 100 metrics for each project
  - Many calculated at multiple levels
    - project, package, class, interface, method
- Acquired and evaluated twelve metrics tools
  - Selected CCCC, Eclipse Metrics, JavaNCSS, JStyle, Krakatau, Clover, Cobertura
- Custom-built Ant scripts and Java programs
  - Invoke metrics tools
  - Extract metrics
    - Count asserts in code
    - Parse xml files produced by metrics tools
- Extensive spreadsheet and statistical analysis
- Web-based and paper survey collection

Metrics

Complexity	• McCabe's Cyclomatic Complexity • Halstead Complexity • LOC/method • Weighted Methods per Class (WMC) • Number of Parameters • Depth of Inheritance Tree	• #Children • Specialization Index • #Overridden Methods • Nested Block Depth • Response for Class
Coupling	• Coupling between Objects • Fan-in, Fan-out (Afferent/Efferent Coupling) • Information Flow	• Instability • #Interfaces
Cohesion	• Lack of Cohesion of Methods • Weighted Methods per Class • LOC/Method	

Metrics

Size	• LOC (source and test) • #Modules • #Classes • #Methods • #Interfaces • Weighted Methods per Class	• LOC/Module • LOC/Method • LOC/Class • #Attributes • #Static Attributes • #Packages
Reusability	• Depth of Inheritance Tree • #Children (bigger is good) • Fan-in • Specialization Index • Distance from Main	• Abstractness • Instability • #Overridden Methods • #Interfaces
Testability	• #Asserts • Line Coverage • Branch Coverage • Method Coverage • Total Coverage	• Response for Class • Depth of Inheritance Tree • #Children • #Overridden Methods

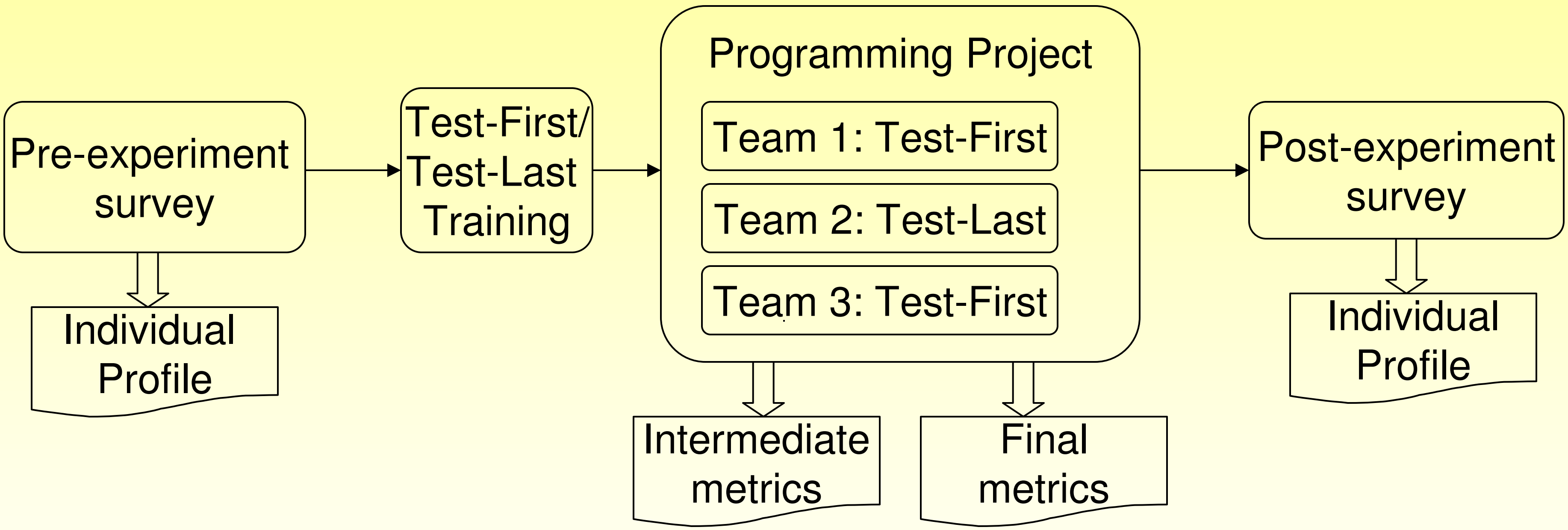
Subjective Metrics

- CS1 and CS2
  - Correctness score (lack of defects)
  - Style (design quality, standards conformance)
  - Source: TA Reviewers
- Industry Projects
  - Design Review Scorecard
    - Understandability: simplicity, architectural clarity/consistency
    - Maintainability: low coupling, high cohesion
    - Reusability/Extensibility: use of design patterns
    - Testability: use of dependency inversion, small cohesive modules
    - Overall Design Quality
  - Source: Peer Reviewers

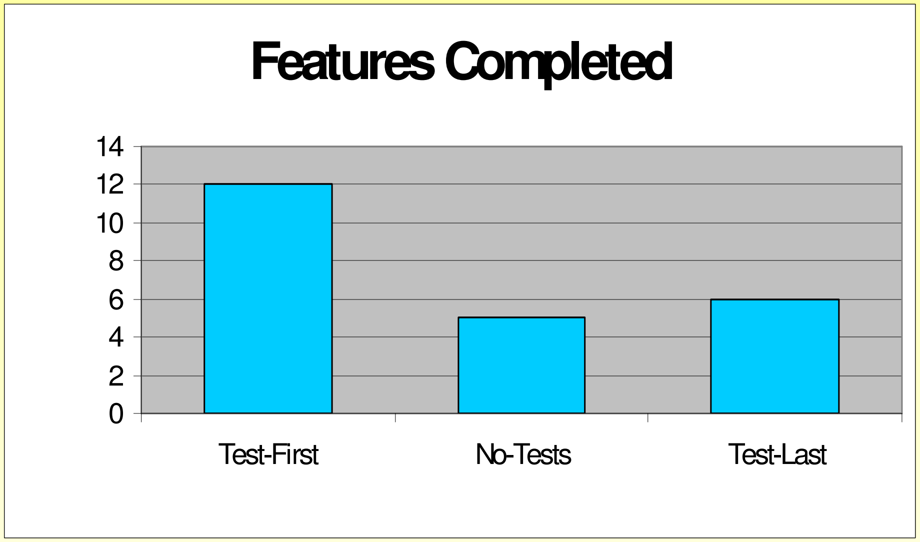
Organization

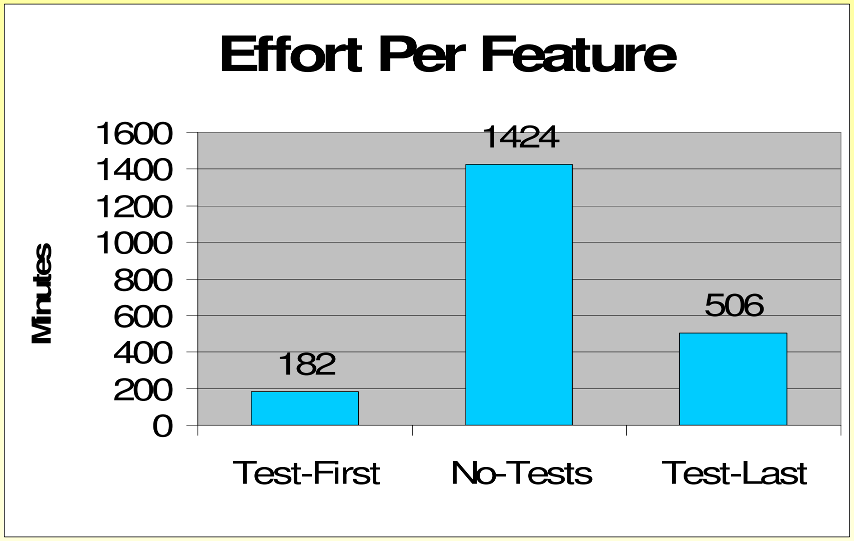
- Problem Statement
- Background
- Research Methodology
- **Evaluation and Results**
  - Sample Detailed Results
  - Summary Results
- Conclusions and Future Work

Undergrad SE Experiment Design



Productivity Results[1]





- Test-First spent 88% less effort/feature than No-Tests
- Test-First spent 57% less effort/feature than Test-Last
- Only Test-First completed both phases

1. Janzen and Saiedian, "On the Influence of Test-Driven Development on Software Design," *Conference on Software Engineering Education and Training (CSEE&T'06)*, 2006

Code Size and Test Density

- Code size (Source only)

	# of classes	LOC	#methods	methods/class	LOC/class	LOC/method	LOC/feature
Test-First	13	1053	87	6.69	81.00	12.10	87.75
No-Tests	7	995	36	5.14	142.14	27.64	199.00
Test-Last	4	259	35	8.75	64.75	7.40	43.17

- Code size (Test only) and Test Coverage

	Test LOC	% Classes Tested	Assertions/SLOC	Test Coverage (lines)	Test Coverage (branches)
Test-First	168	38.46%	0.077	19.00%	39.00%
No-Tests	0	0.00%	0.000	0.00%	0.00%
Test-Last	38	25.00%	0.045	29.00%	23.00%

Test-First wrote more tests per LOC

but, coverage was mixed

Code Size and Test Density (No GUI)

- Test-first project included an extensive GUI
- GUI's are traditionally difficult to test
- Code size (source only without GUI)

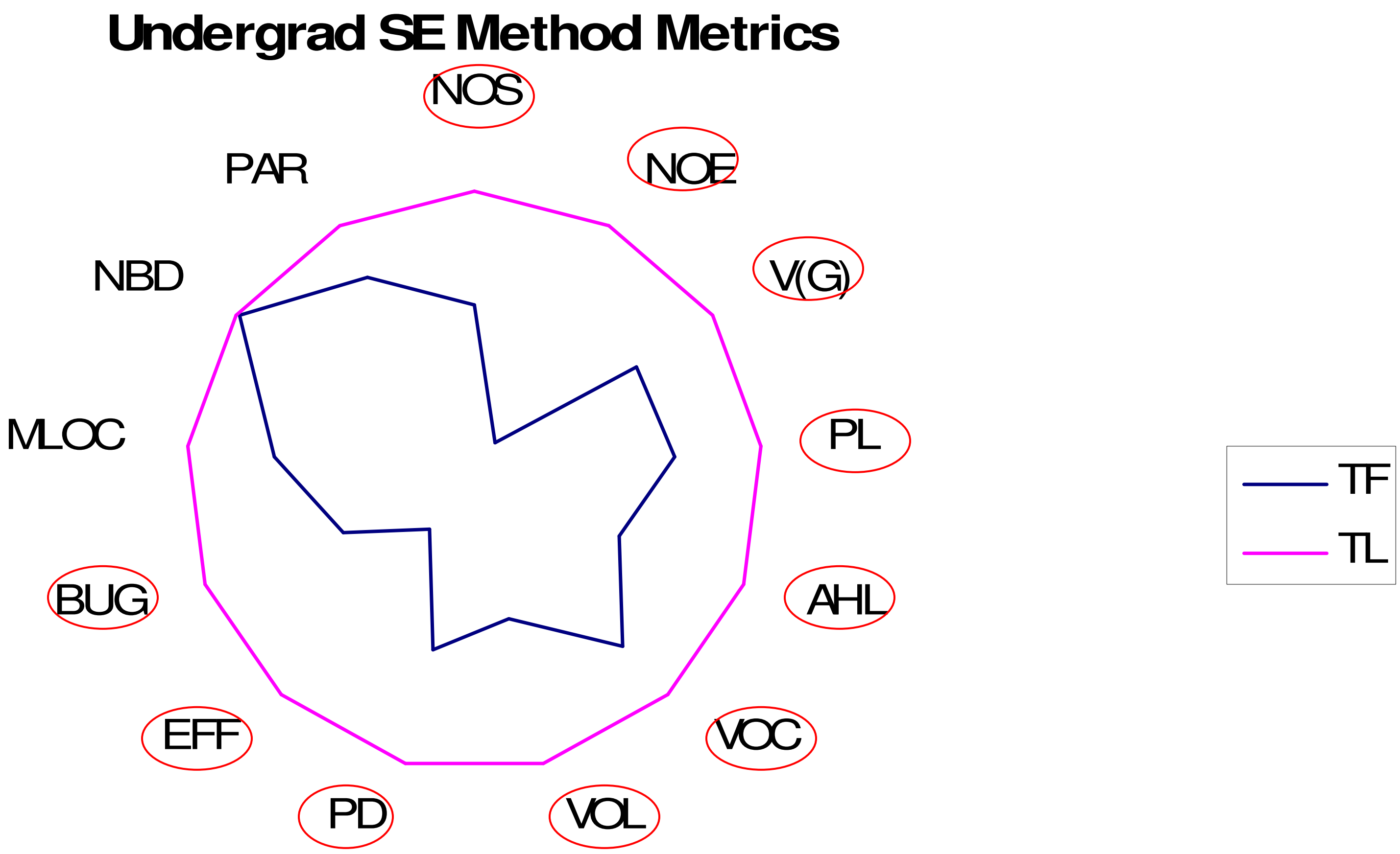
	# of classes	LOC	#methods	methods/class	LOC/class	LOC/method	LOC/feature
Test-First	11	670	57	5.18	60.91	11.75	55.83
No-Tests	7	995	36	5.14	142.14	27.64	199.00
Test-Last	4	259	35	8.75	64.75	7.40	43.17

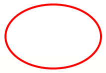
- Code size (test only) and test coverage

	Test LOC	% Classes Tested	Assertions/SLOC	Test Coverage (lines)	Test Coverage (branches)
Test-First	168	38.46%	0.086	31.00%	43.00%
No-Tests	0	0.00%	0.000	0.00%	0.00%
Test-Last	38	25.00%	0.045	29.00%	23.00%

Test-First tests covered more source code

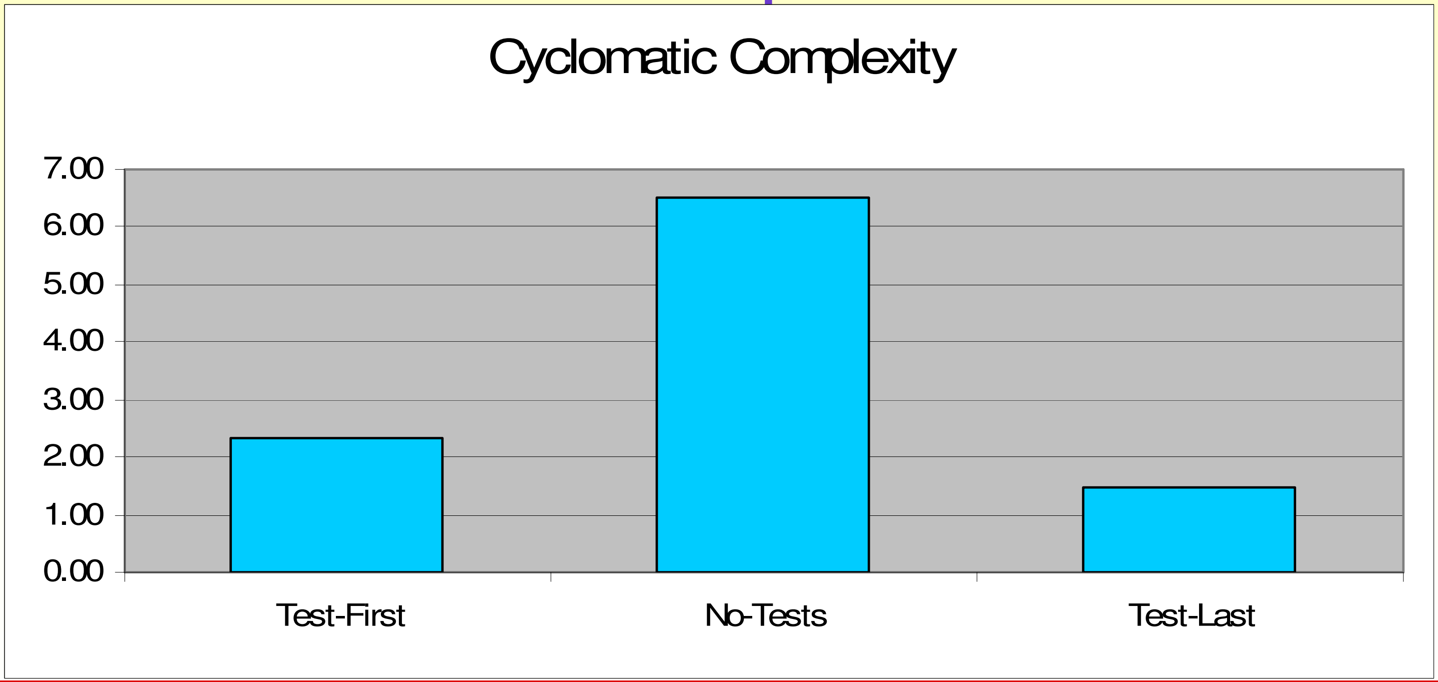
Design Quality: Method-level Metrics



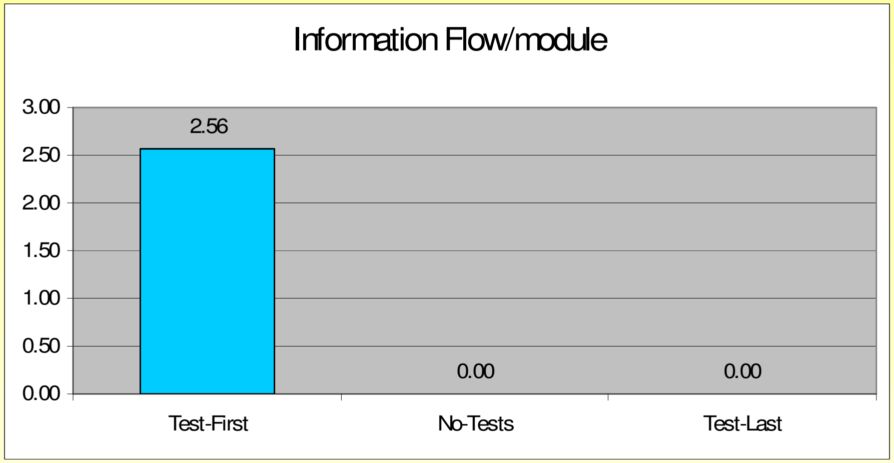
 indicates statistically significant difference with p<.05

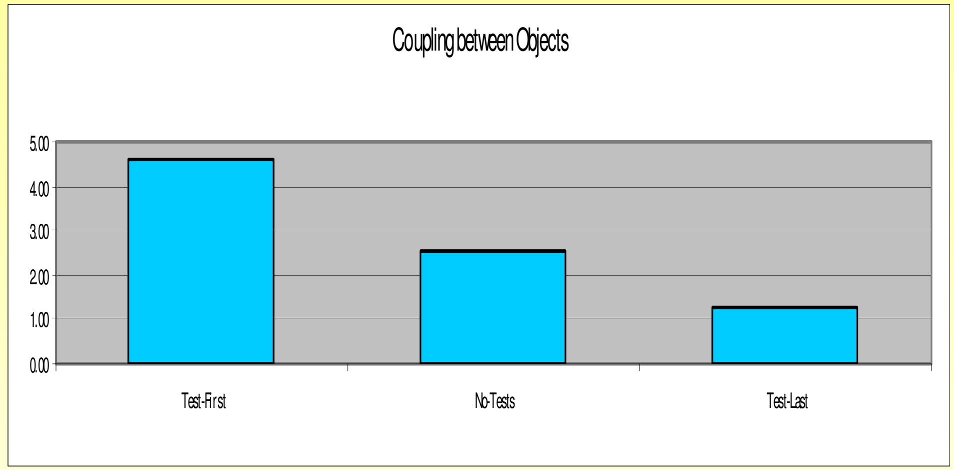
Design Quality: Class-level Metrics

- Comparable/acceptable levels for most metrics: DIT, NOC, LCOM, …
- NII only metric w/ statistically significant diff
- Tested code was simpler



Design Quality: Class-level Metrics



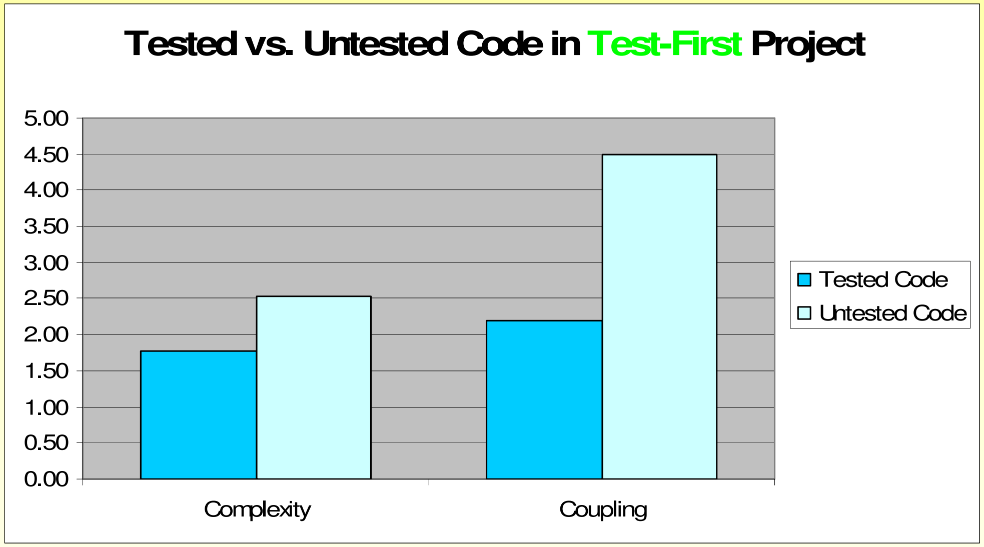


0 Information Flow indicates procedural/flat design in No-Tests and Test-Last teams

Higher coupling in Test-First

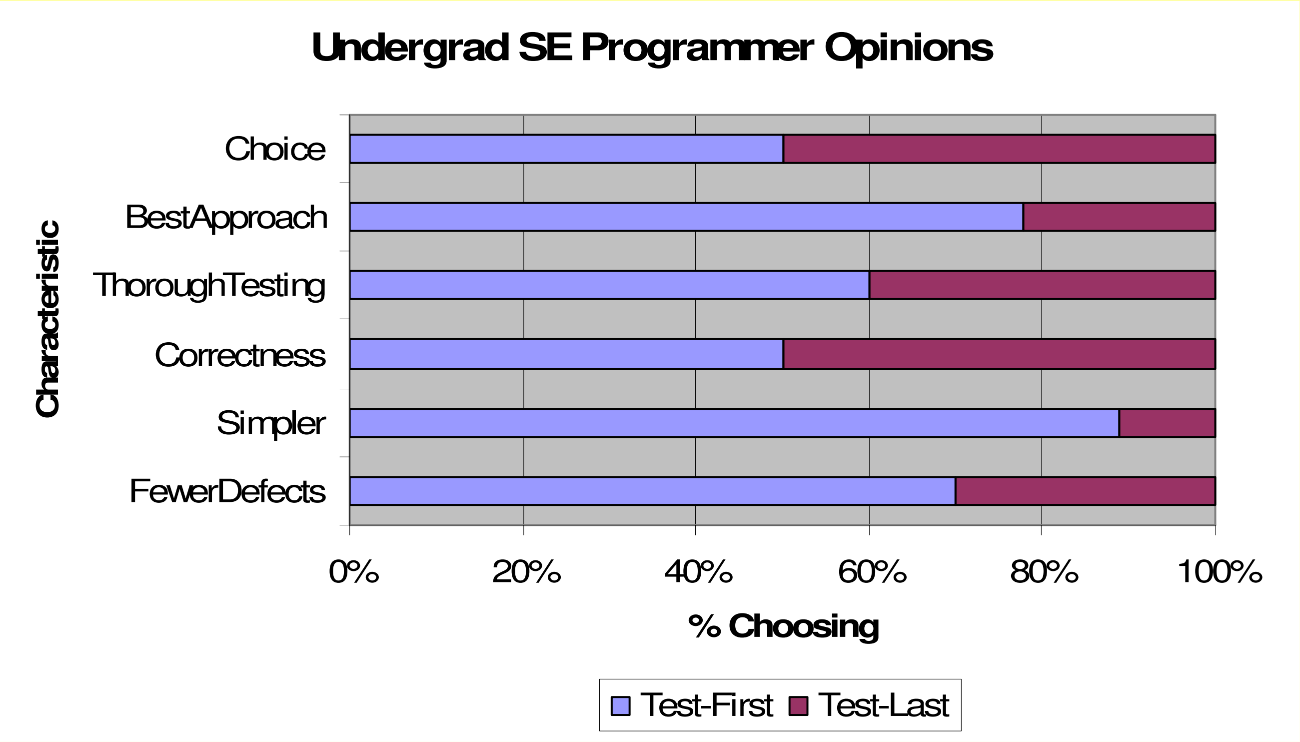
Test-First Team Micro-evaluation

- Evaluated differences in methods tested versus those without tests



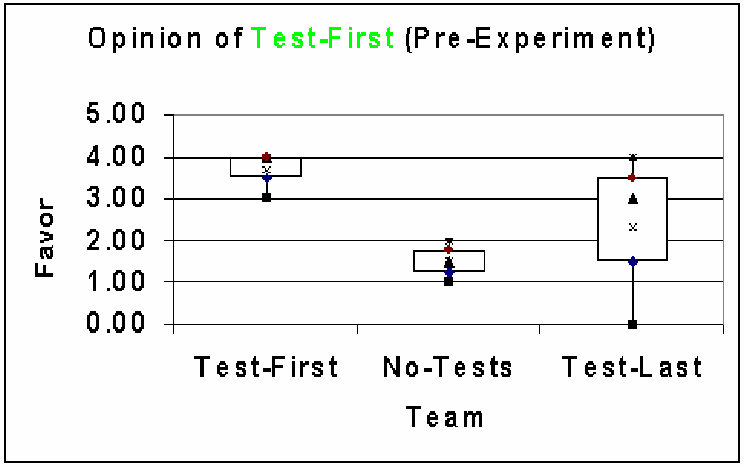
- About 28% of the methods were tested directly
  - These methods had ~43% lower complexity average
  - Not statistically significant at p=.08
- Classes that had some methods tested directly had an average coupling that was ~104% lower

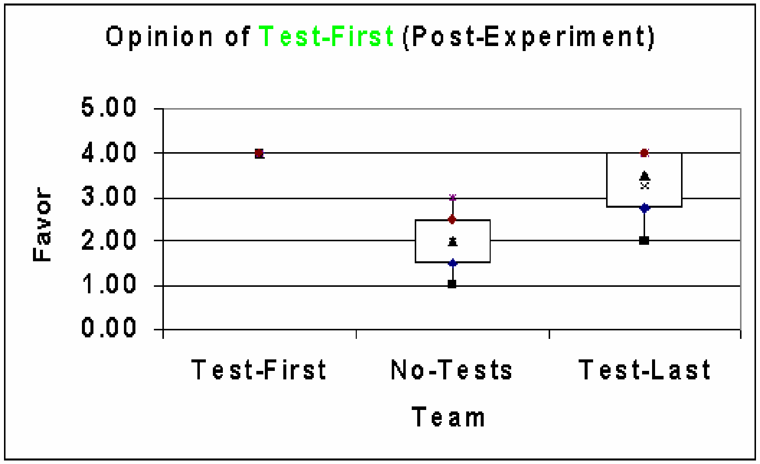
Programmer Opinions



Student Perceptions[1]

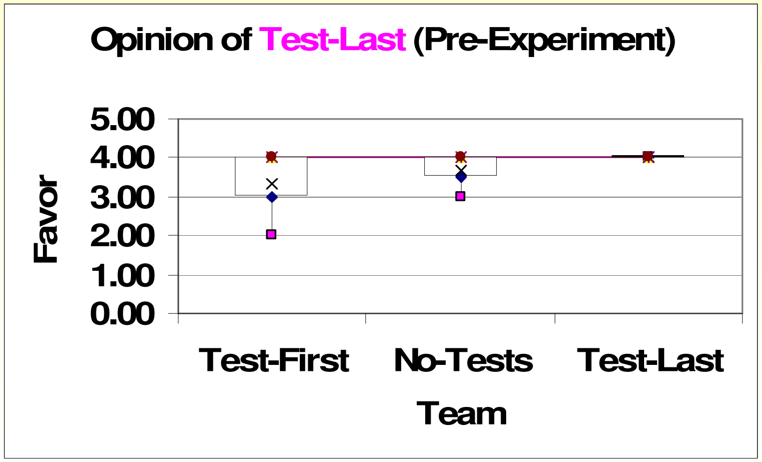
Test-First

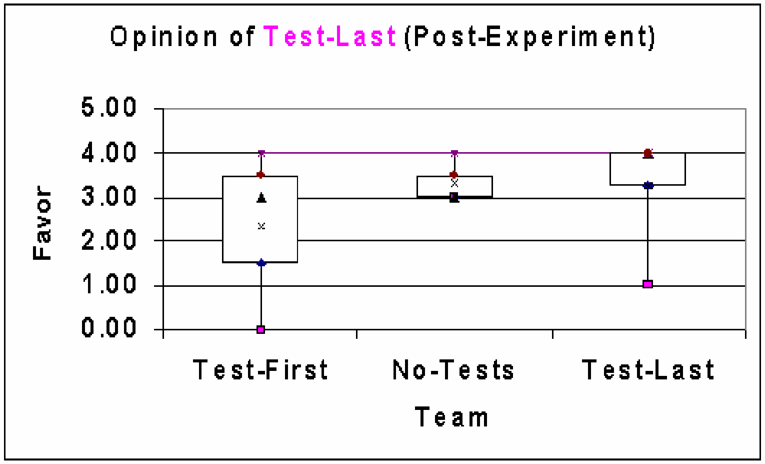




Opinions of TF improved 27% – paired t-test was statistically significant

Test-Last



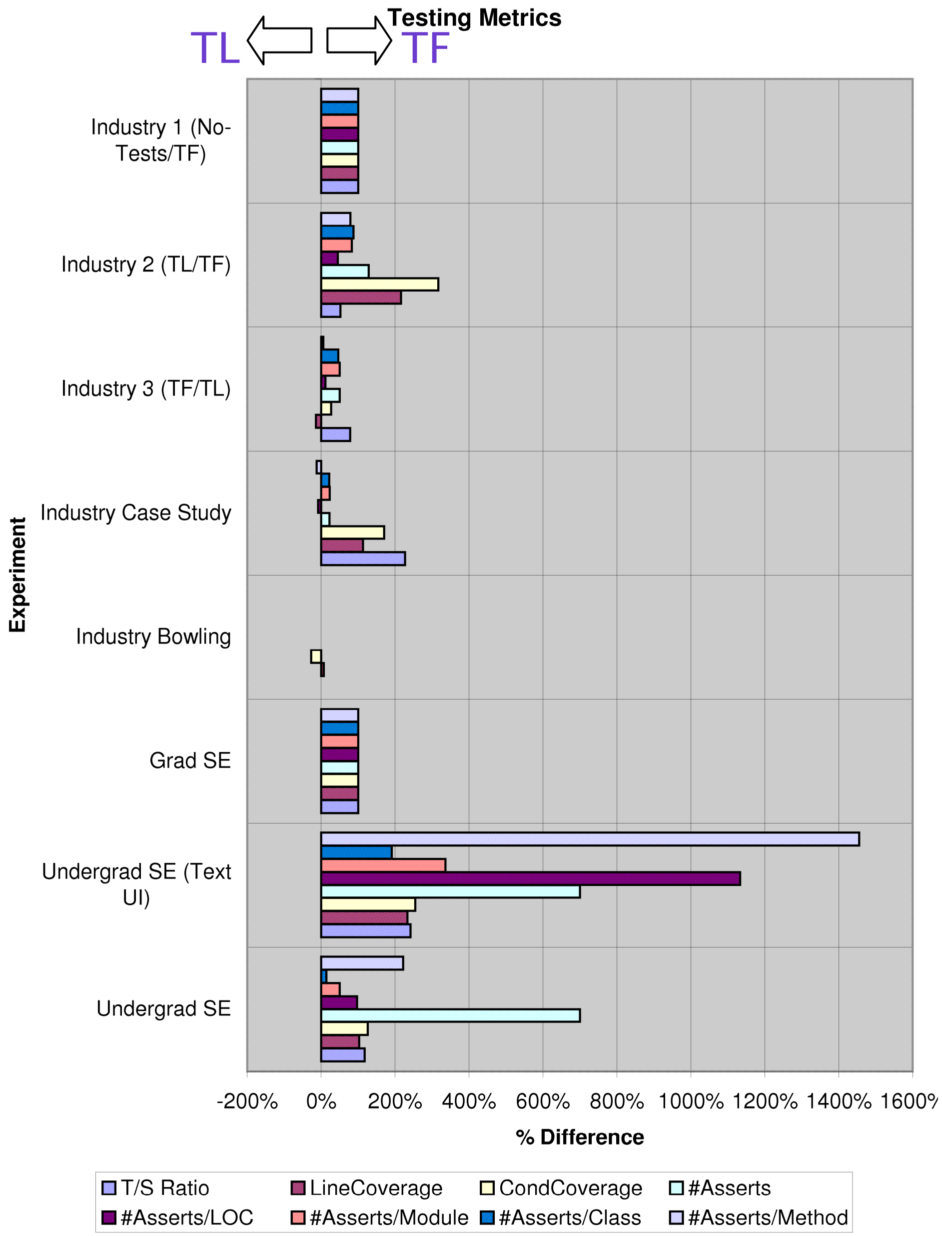


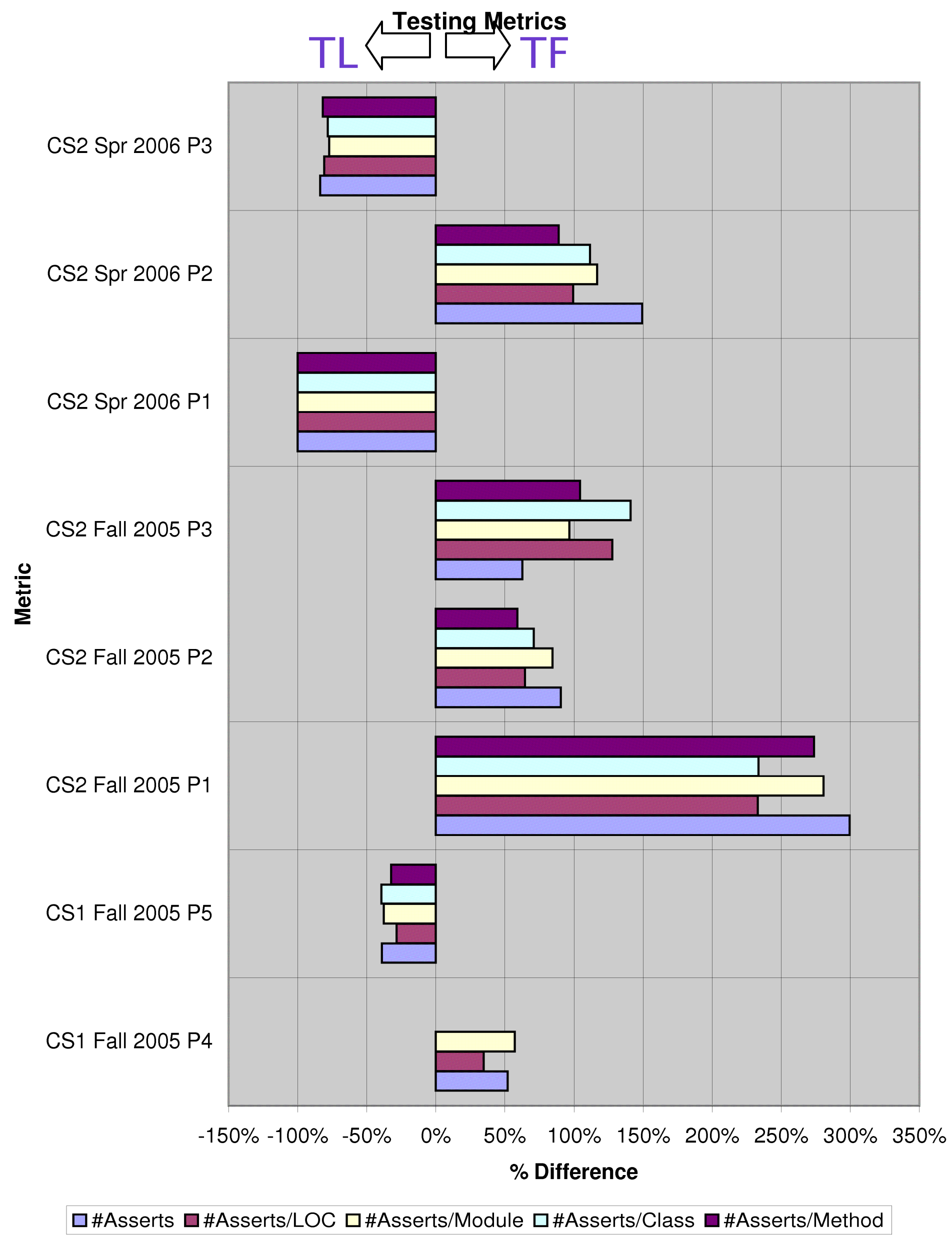
Opinions of TL declined 19% – paired t-test not statistically significant

1. Janzen, "Software Architecture Improvement through Test-Driven Development," *OOPSLA,* 2005

Testing Results

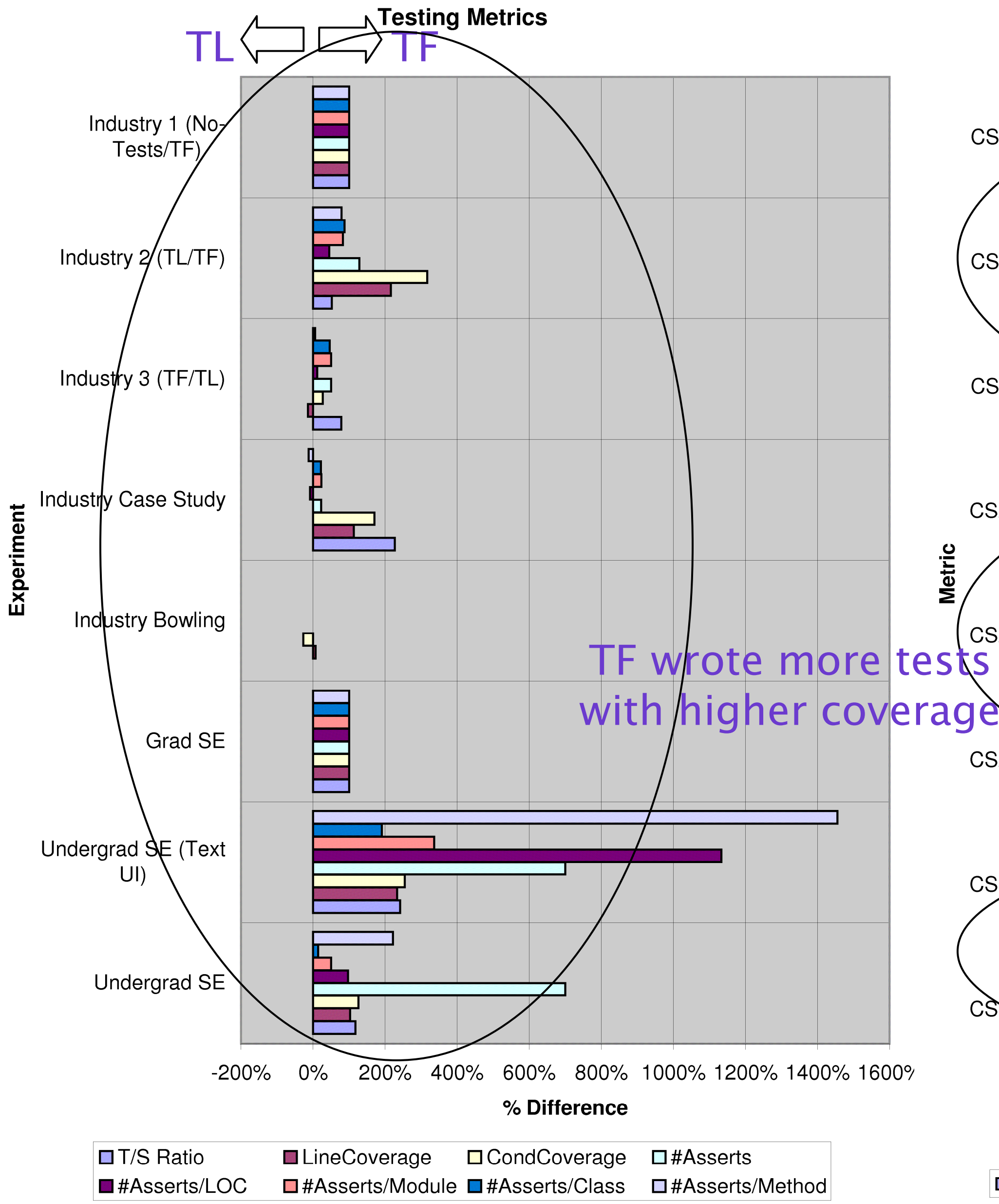
Test-last is better ⇦ ⇨ Test-first is better

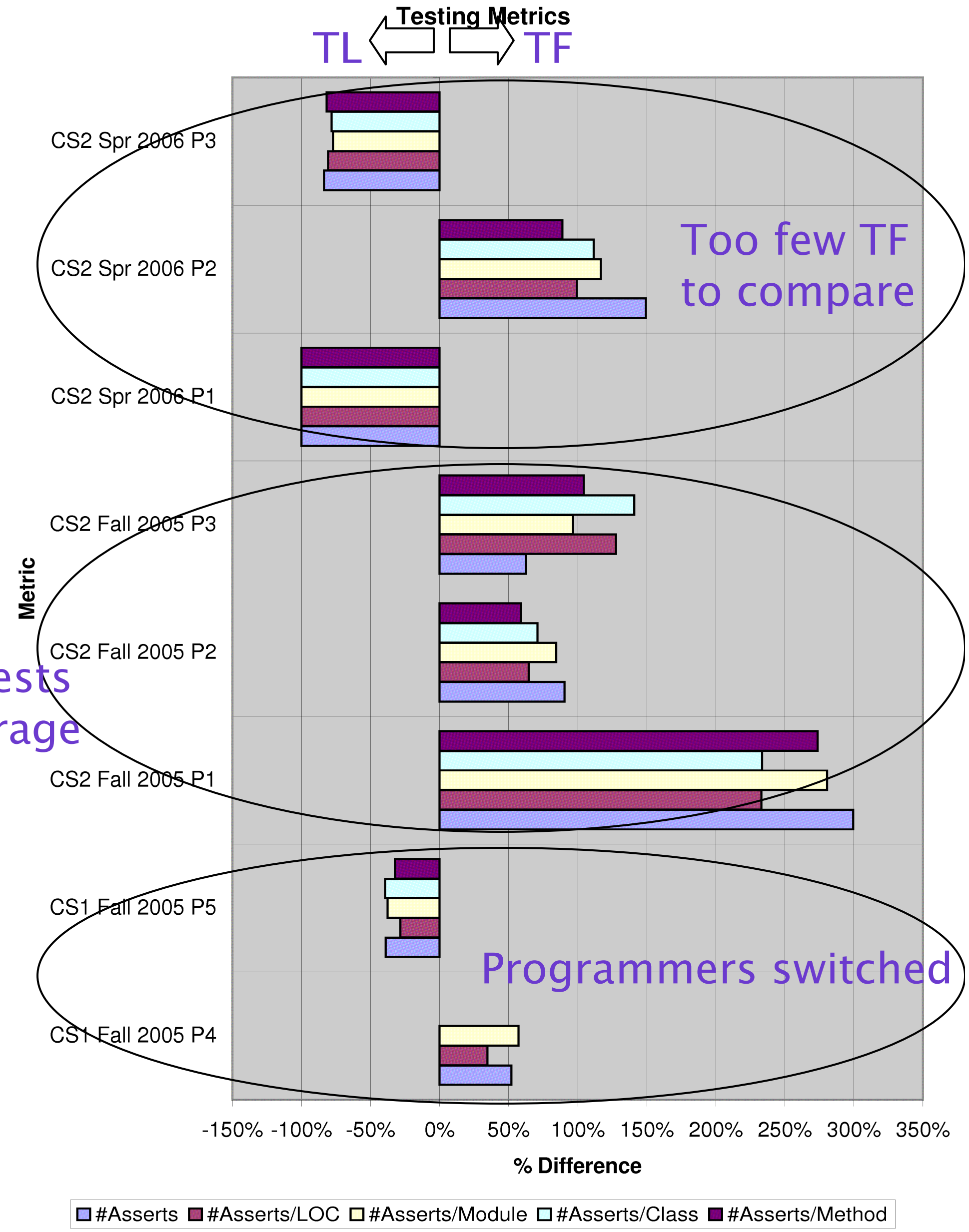




Testing Results

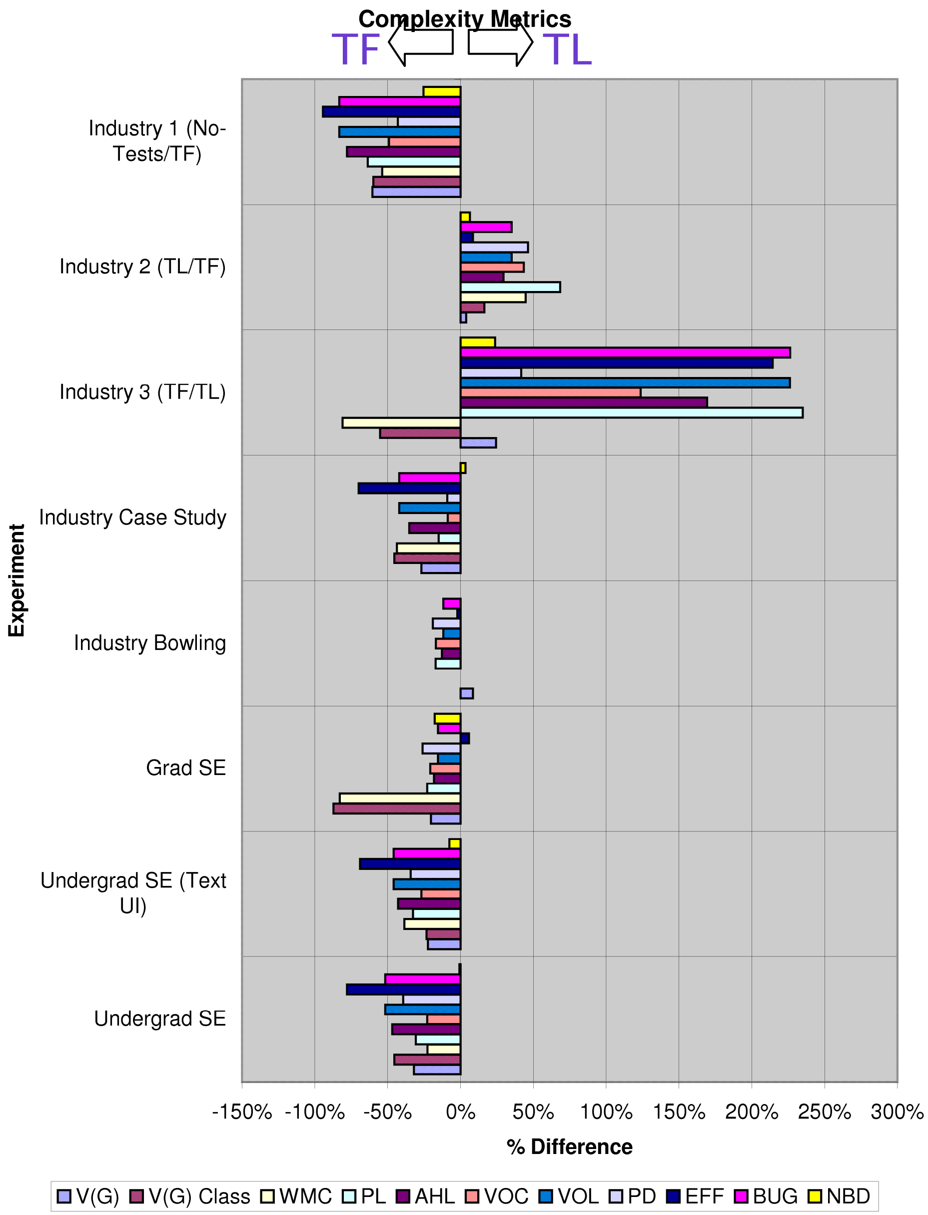
Test-last is better ⇔ Test-first is better

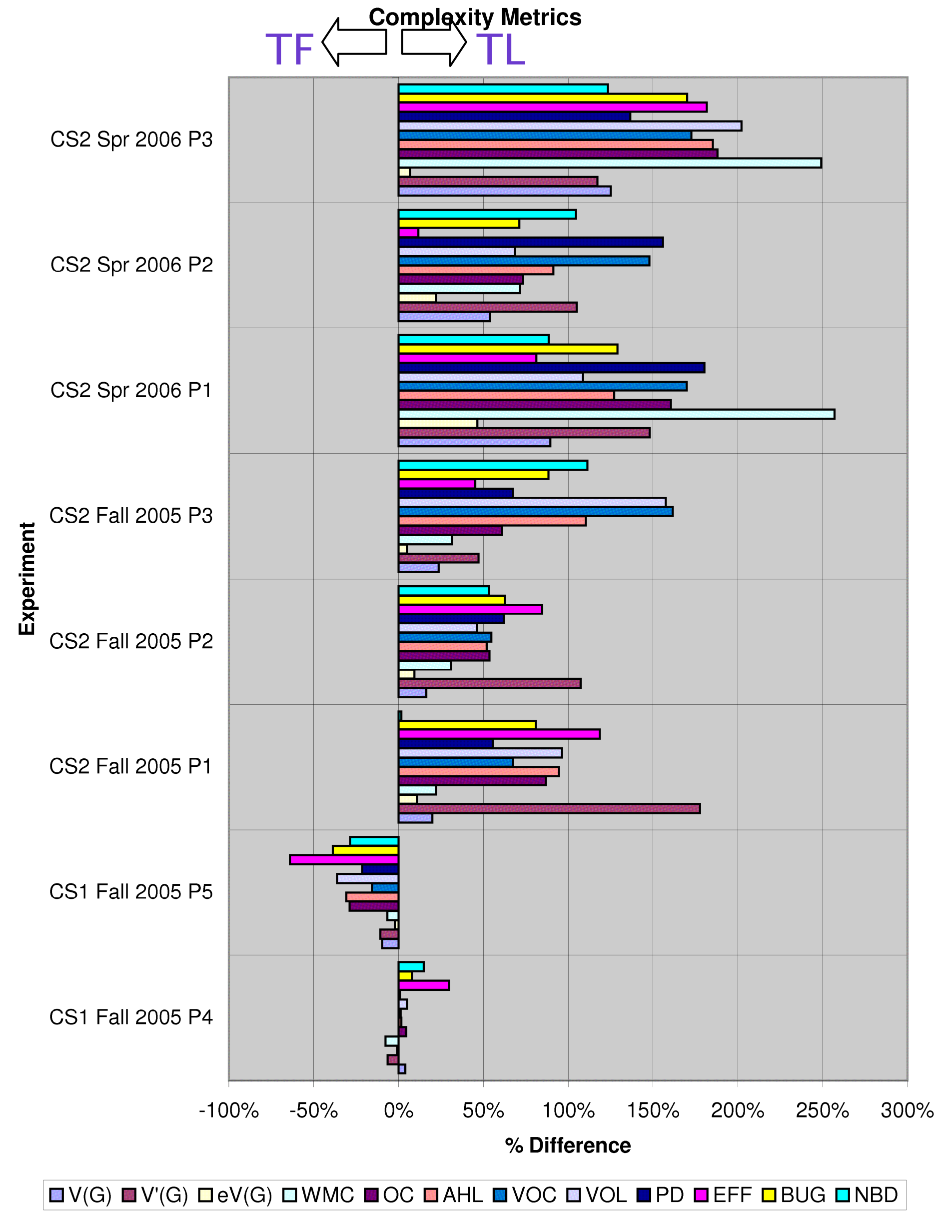




Complexity Results

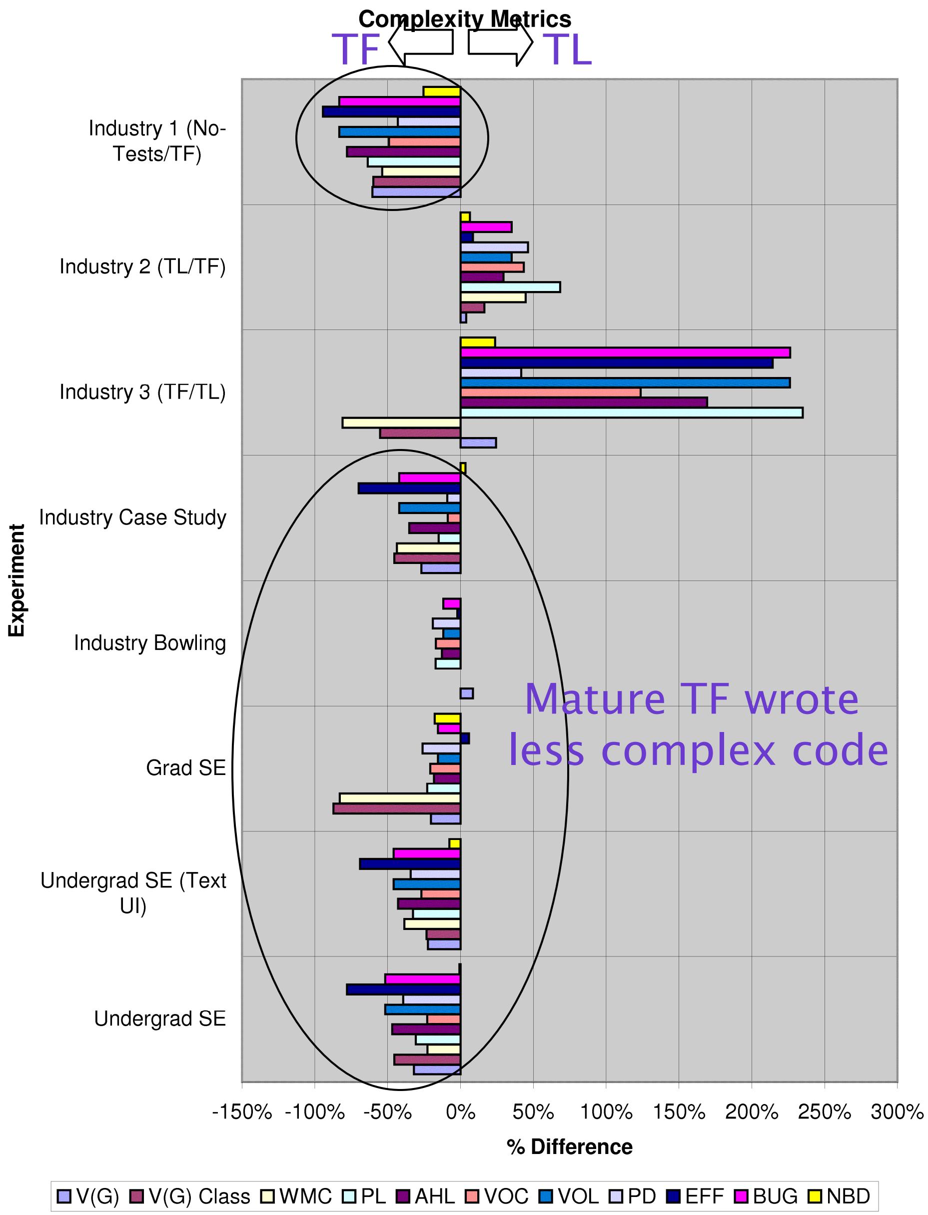
Test-first is less complex ⇦ ⇨ Test-last is less complex

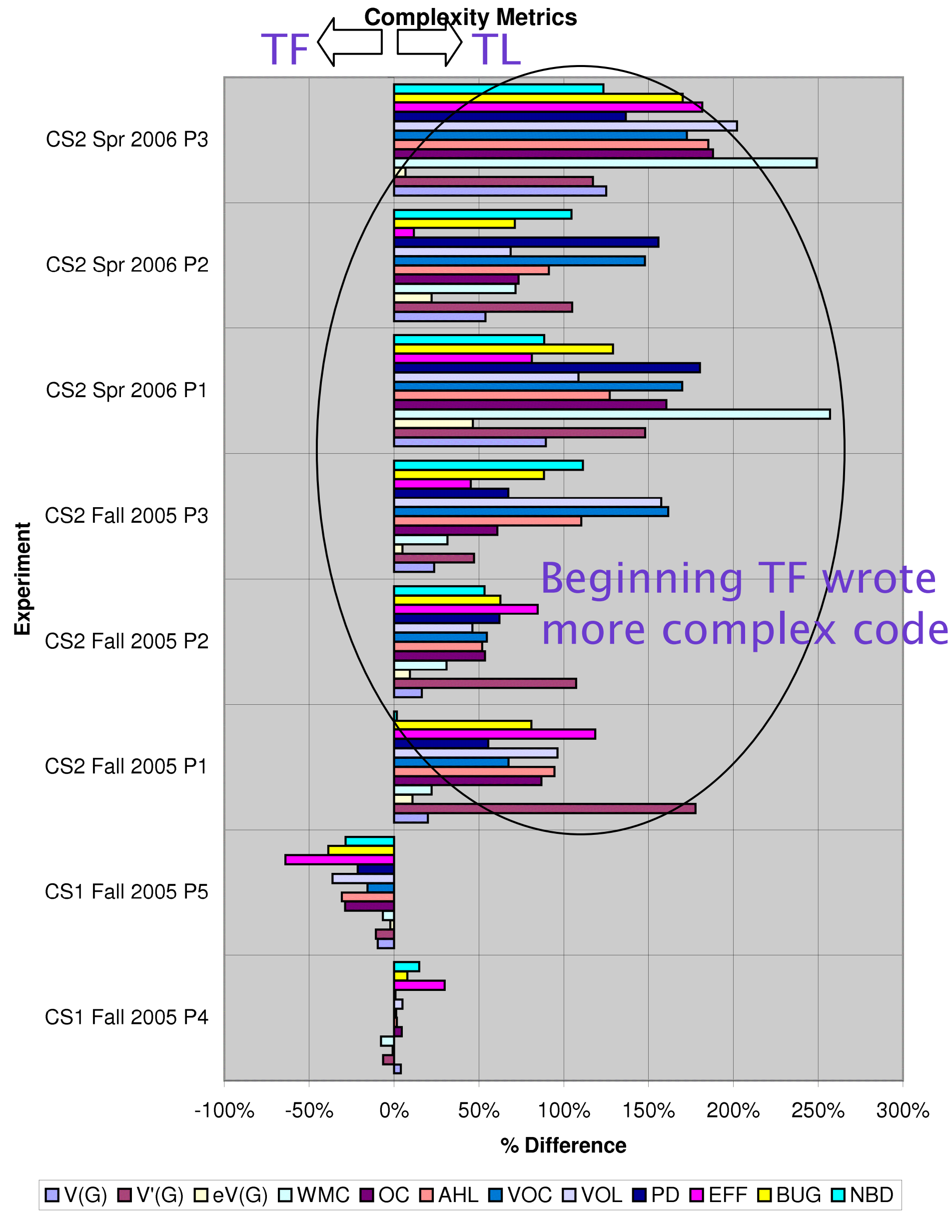




Complexity Results

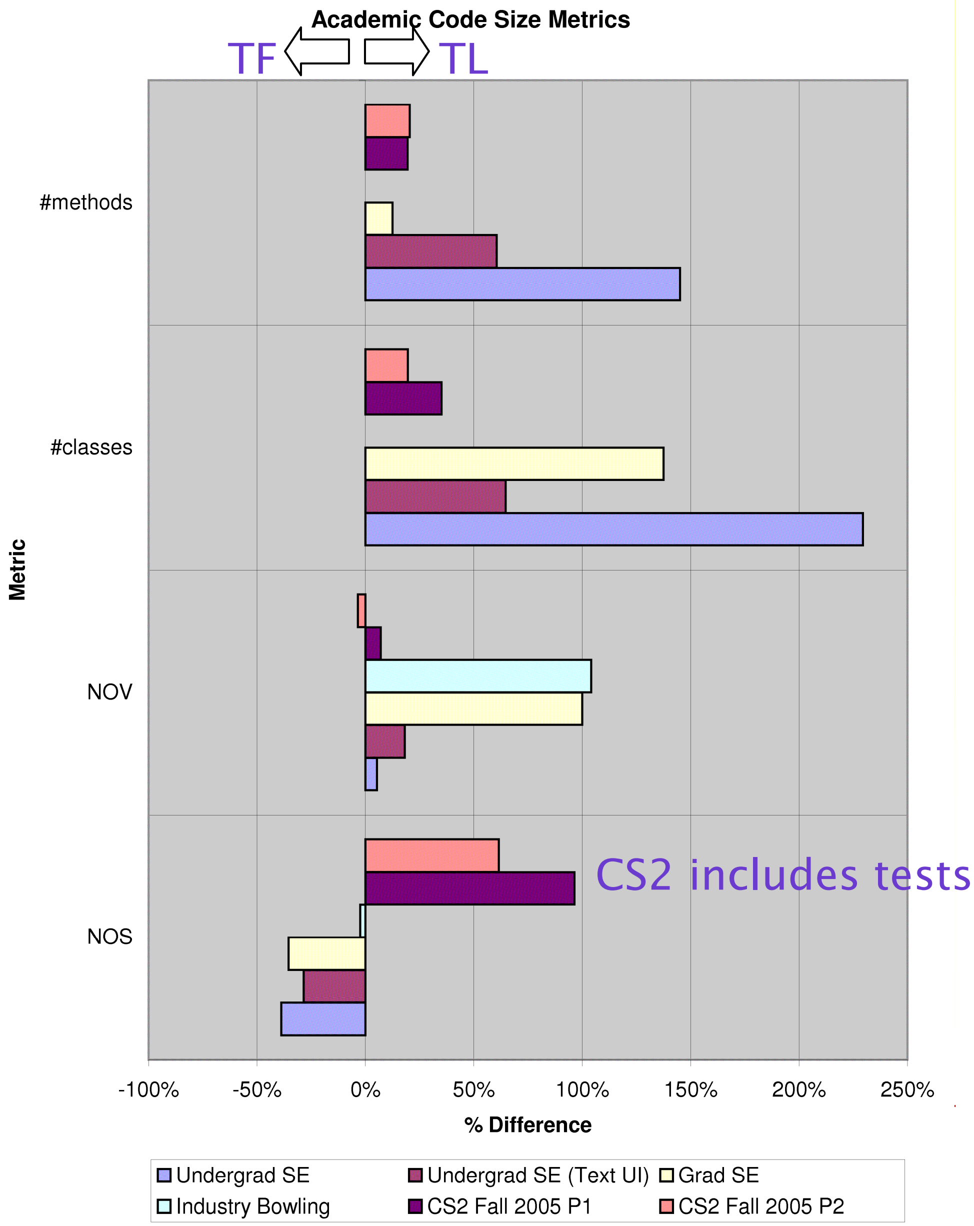
Test-first is less complex ⇦ ⇨ Test-last is less complex

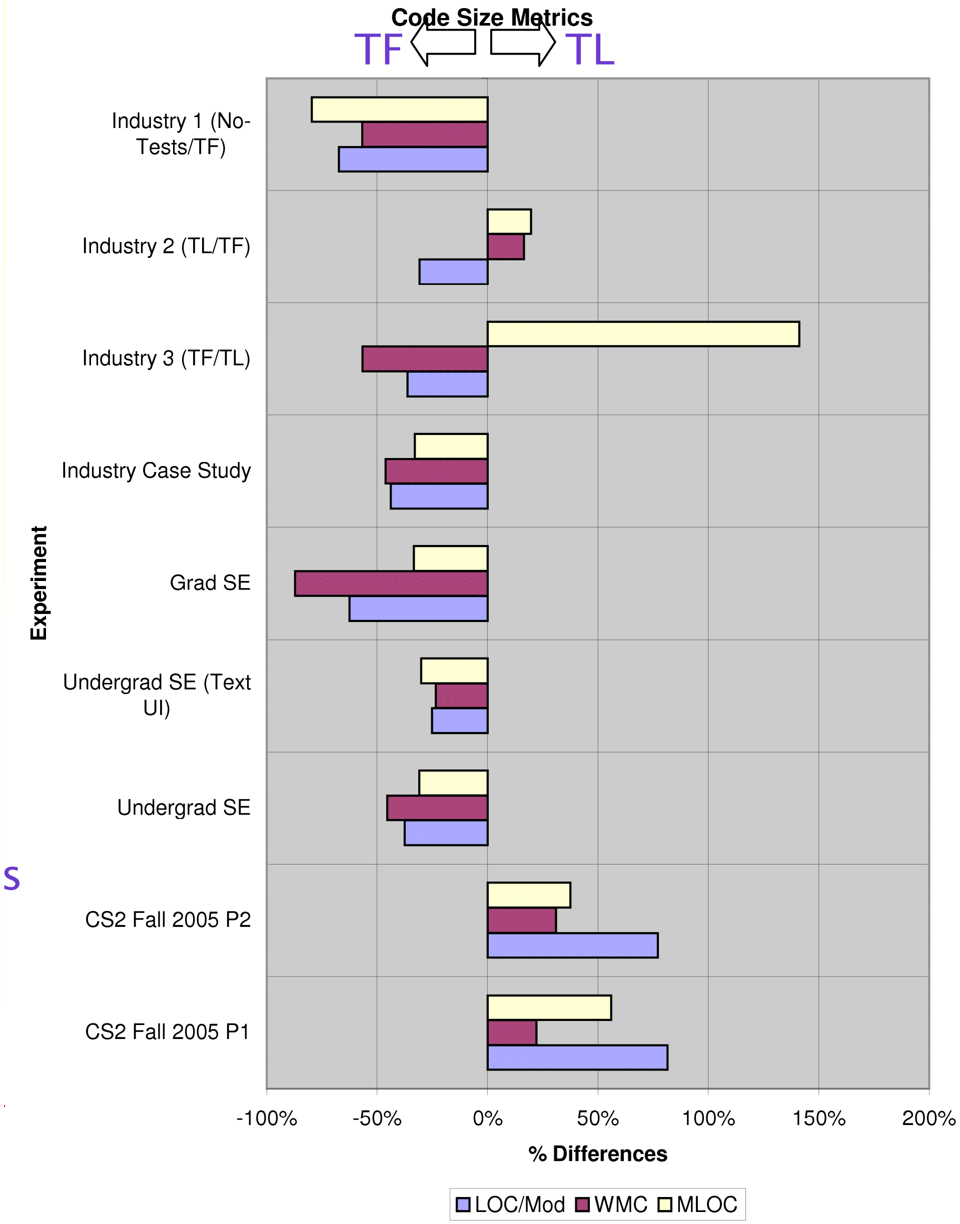




Size Results

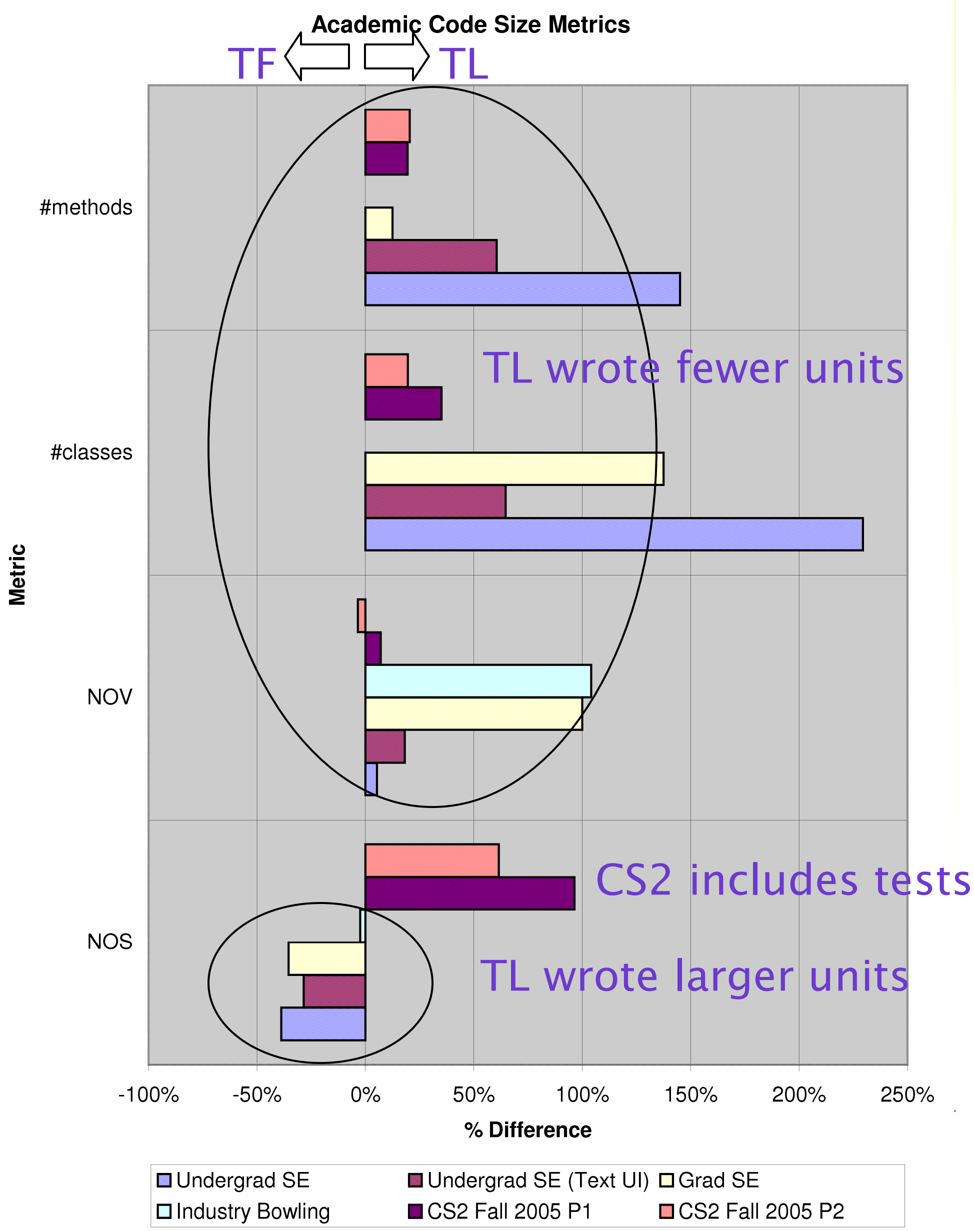
Test-first is smaller ⇦ ⇨ Test-last is smaller

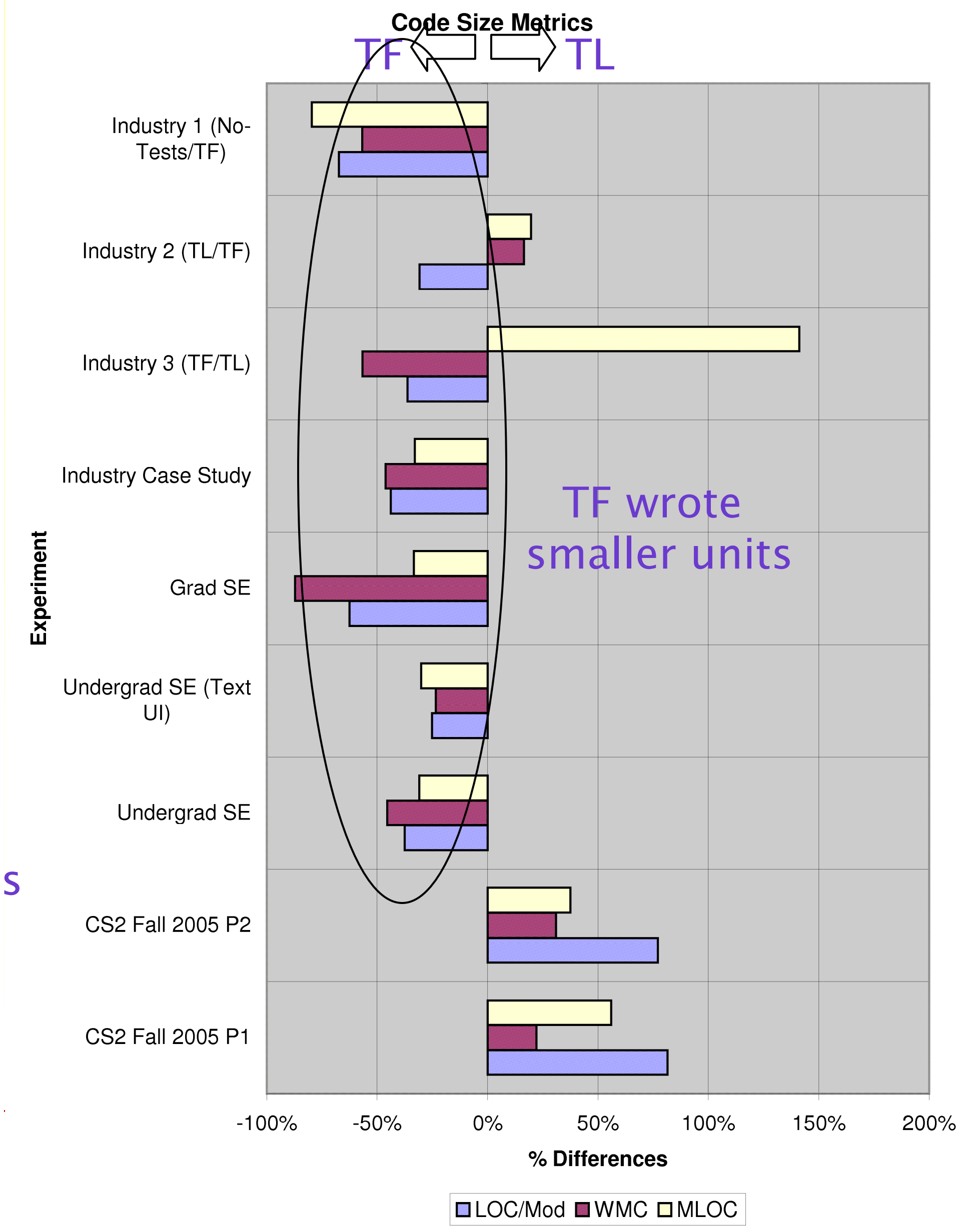




Size Results

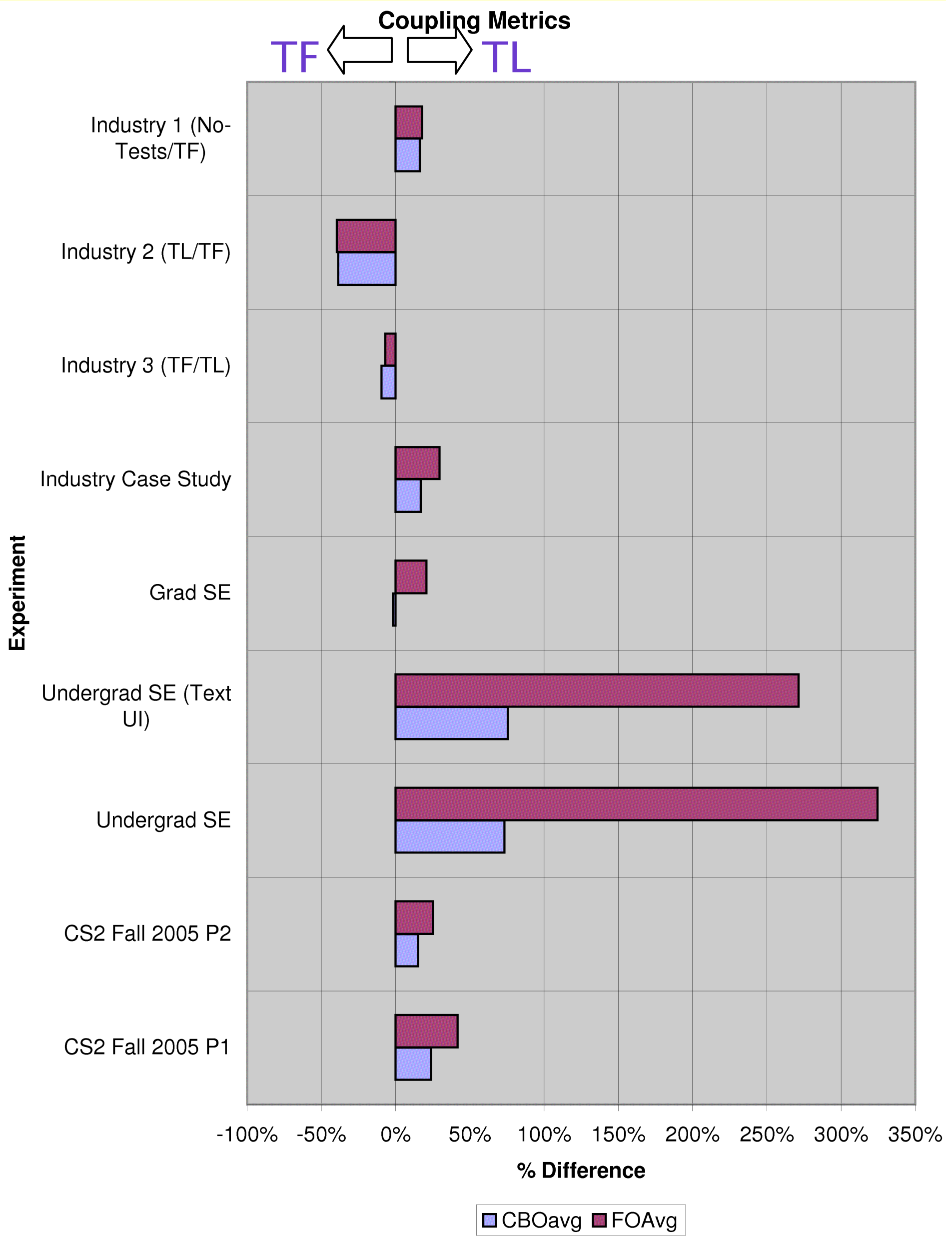
Test-first is smaller ⇦ ⇨ Test-last is smaller

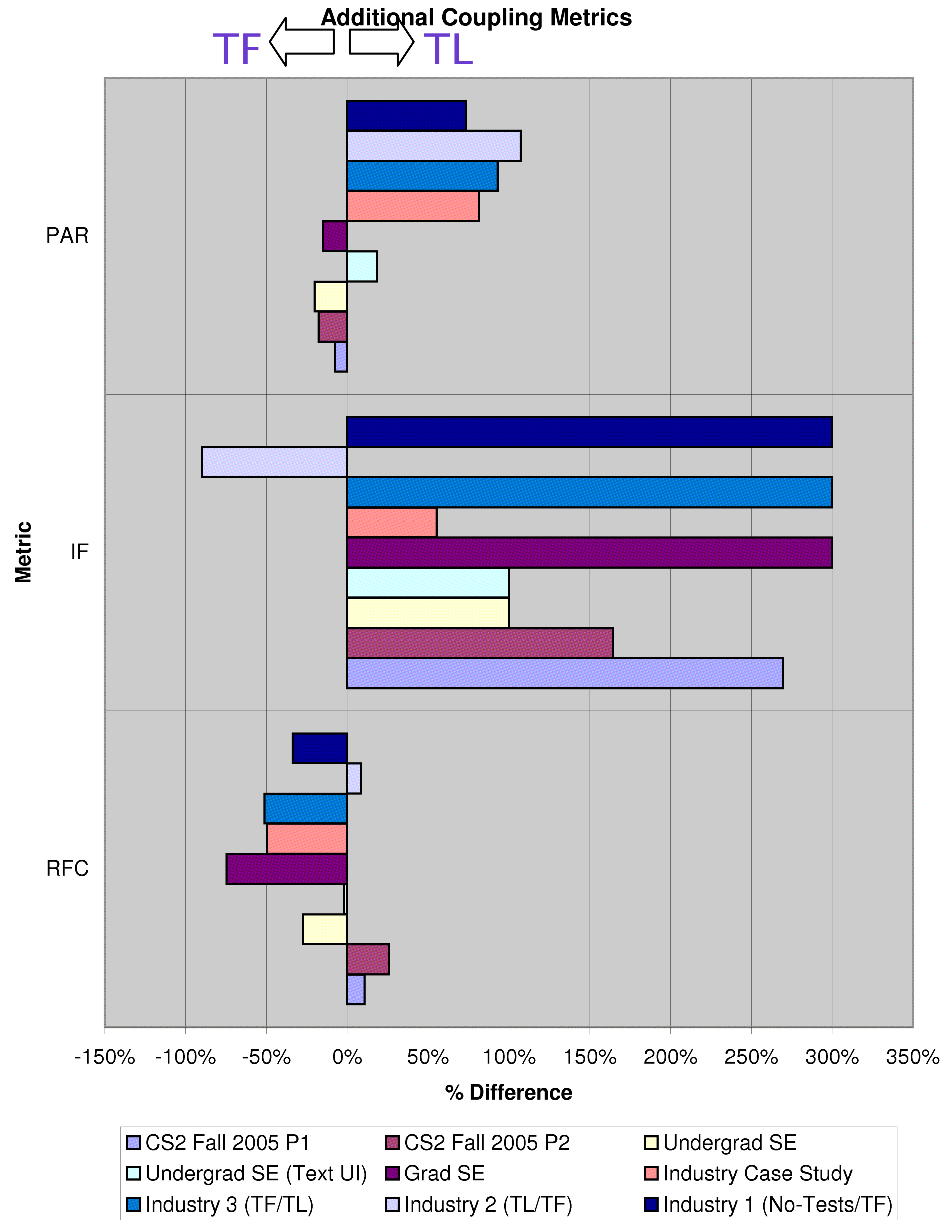




Coupling Results

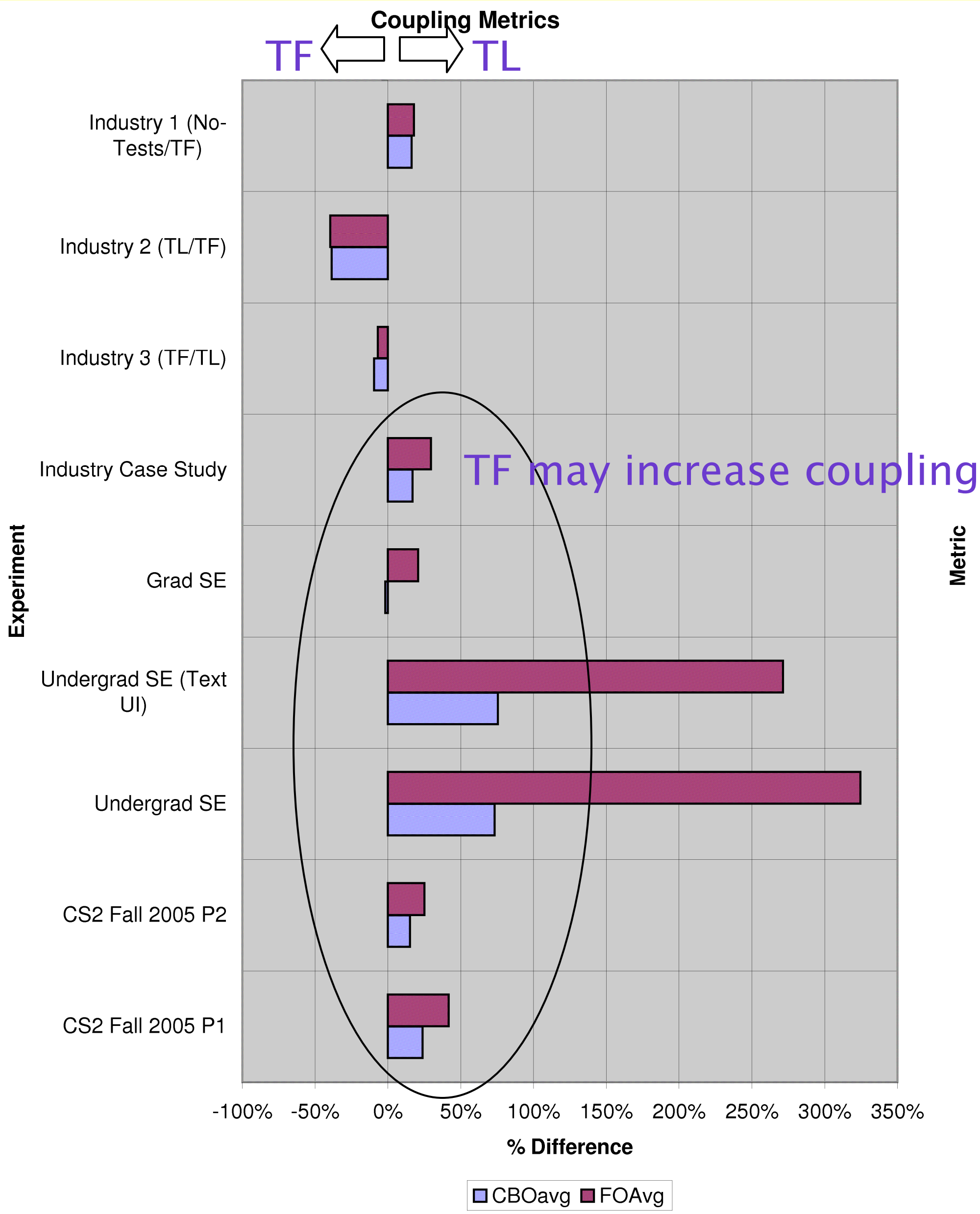
Test-first has lower coupling ⇦ ⇨ Test-last has lower coupling

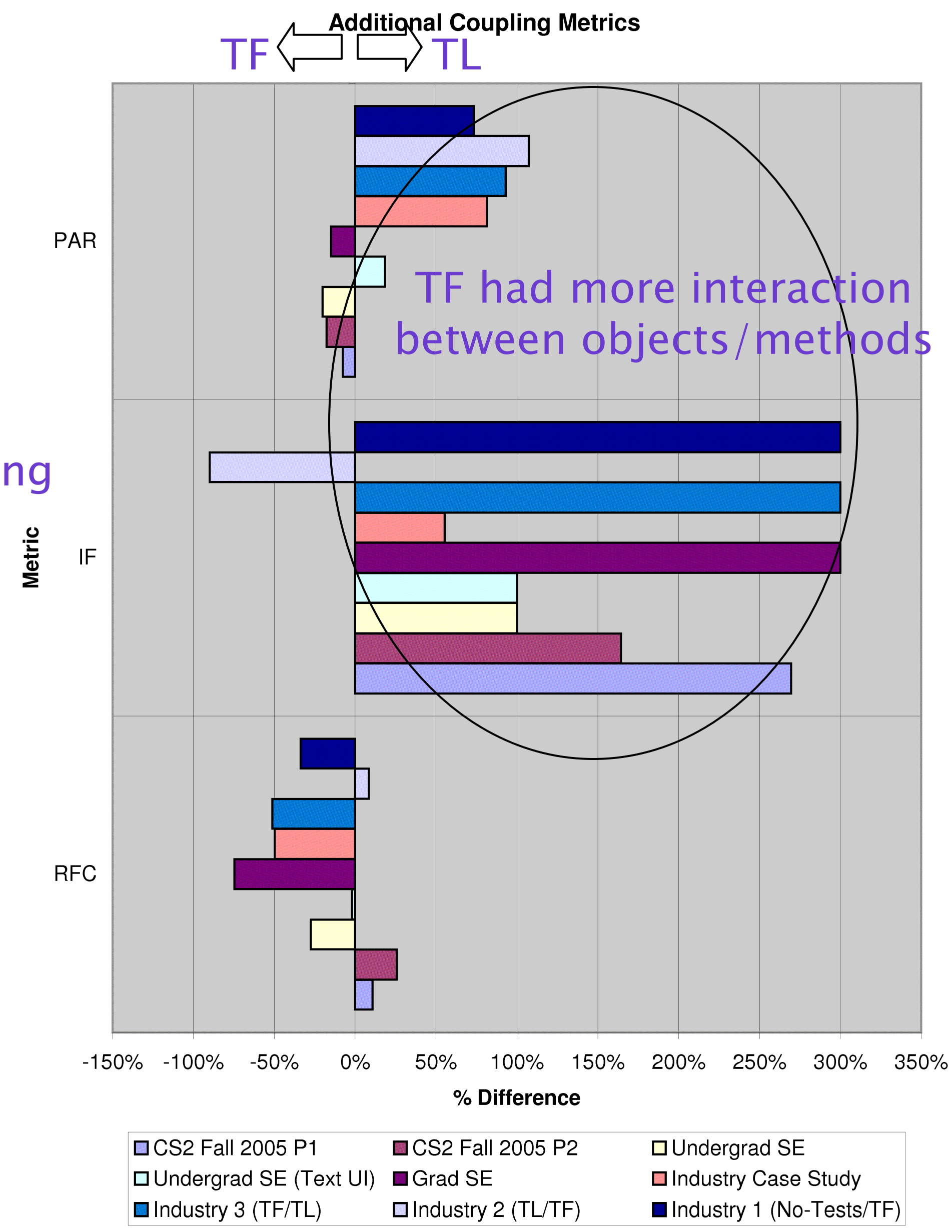




Coupling Results

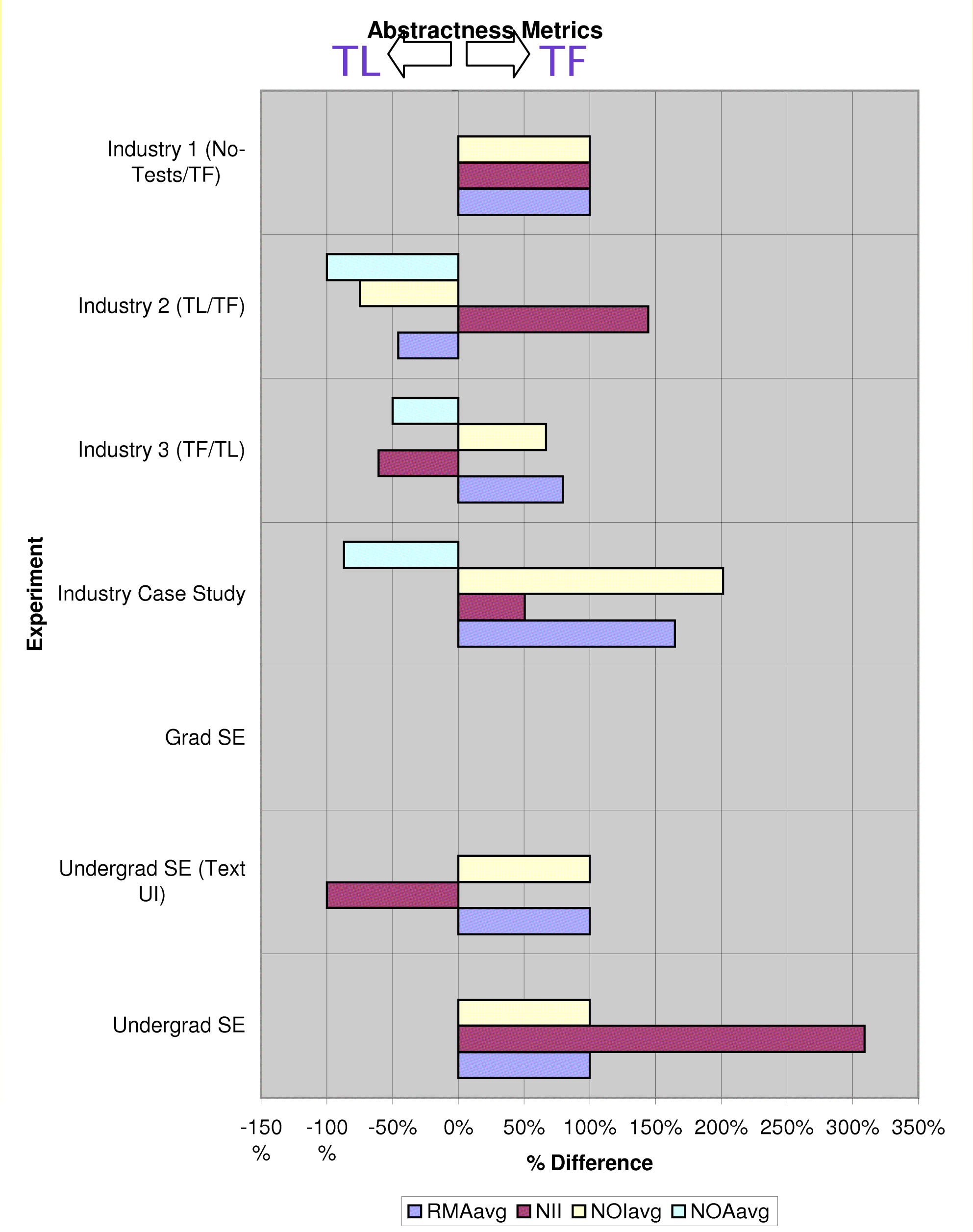
Test-first has lower coupling ⇦ ⇨ Test-last has lower coupling





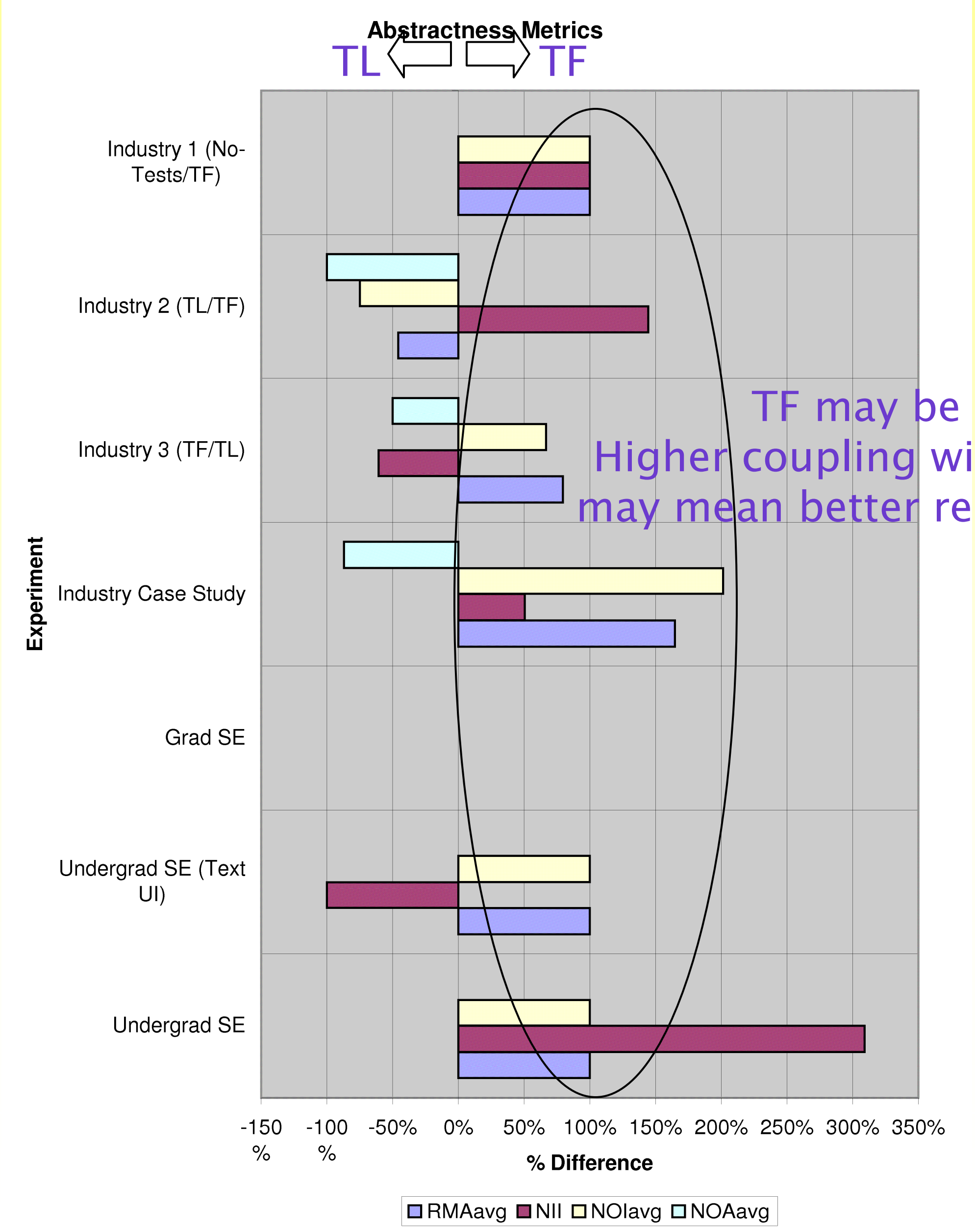
Abstractness Results

Test-last is more abstract ⇦ ⇨ Test-first is more abstract



Abstractness Results

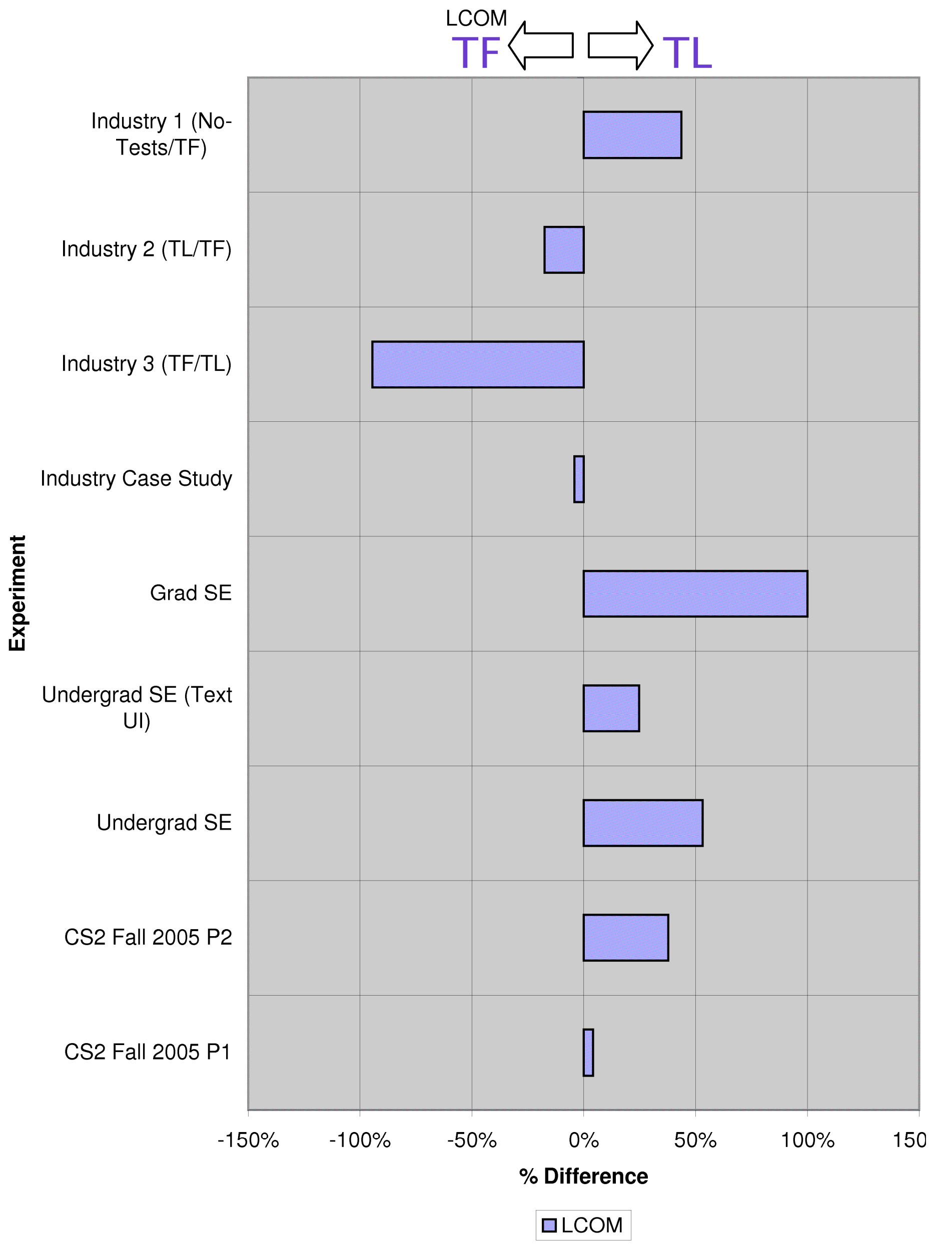
Test-last is more abstract ⇦ ⇨ Test-first is more abstract

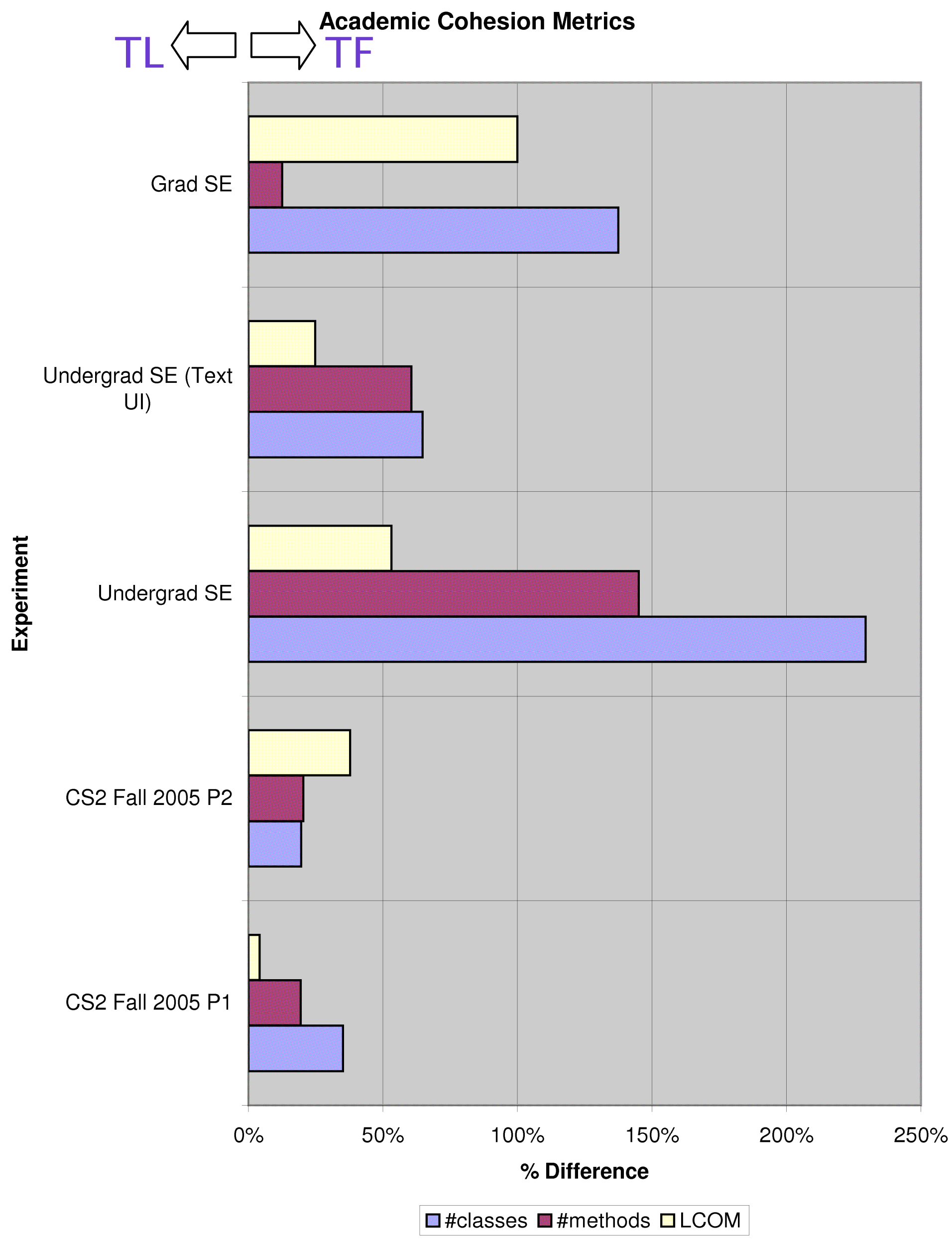


TF may be more abstract;
Higher coupling with higher abstractness may mean better reuse and maintainability

Cohesion Results

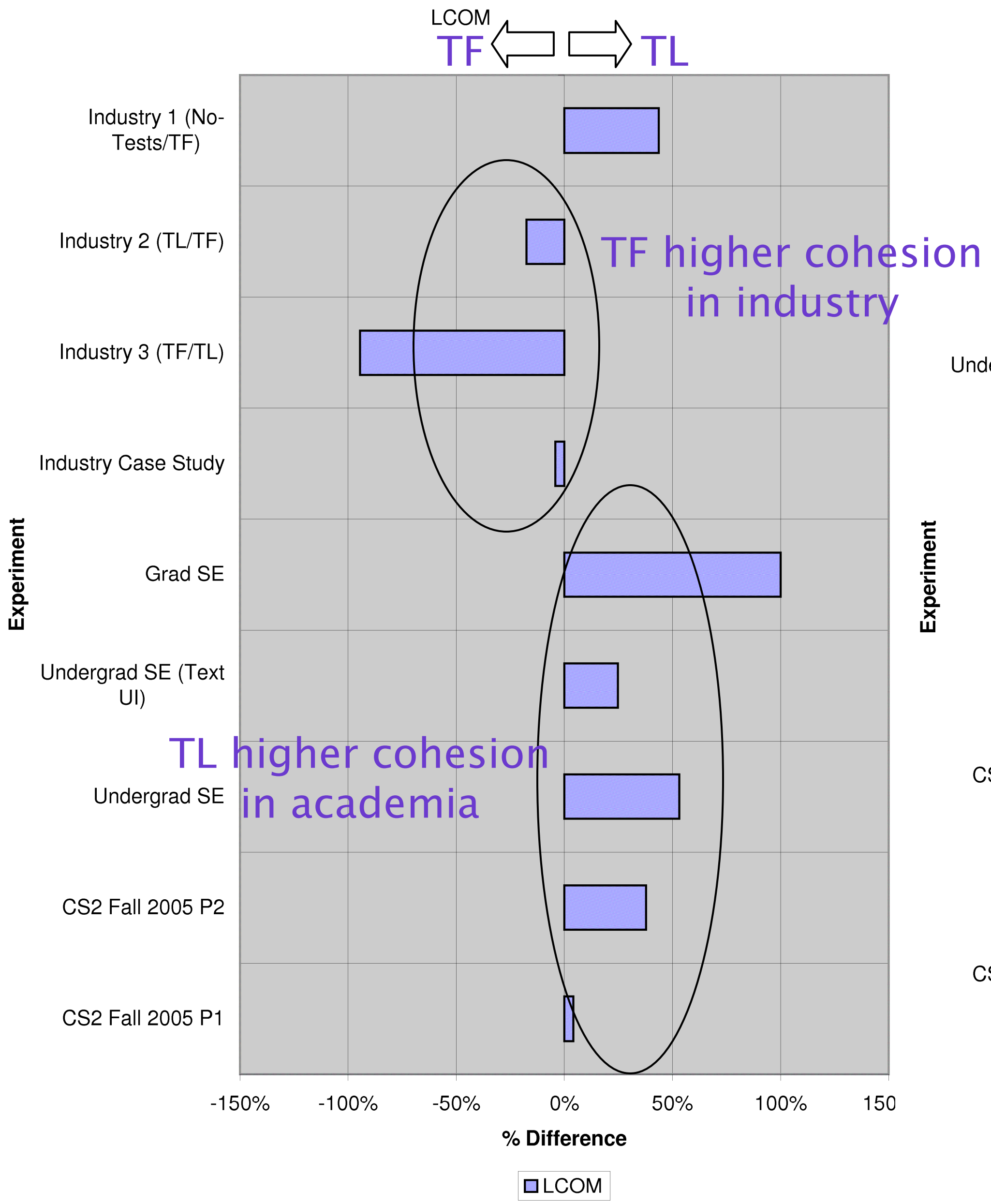
Test-first has higher cohesion ⇦ ⇨ Test-last has higher cohesion

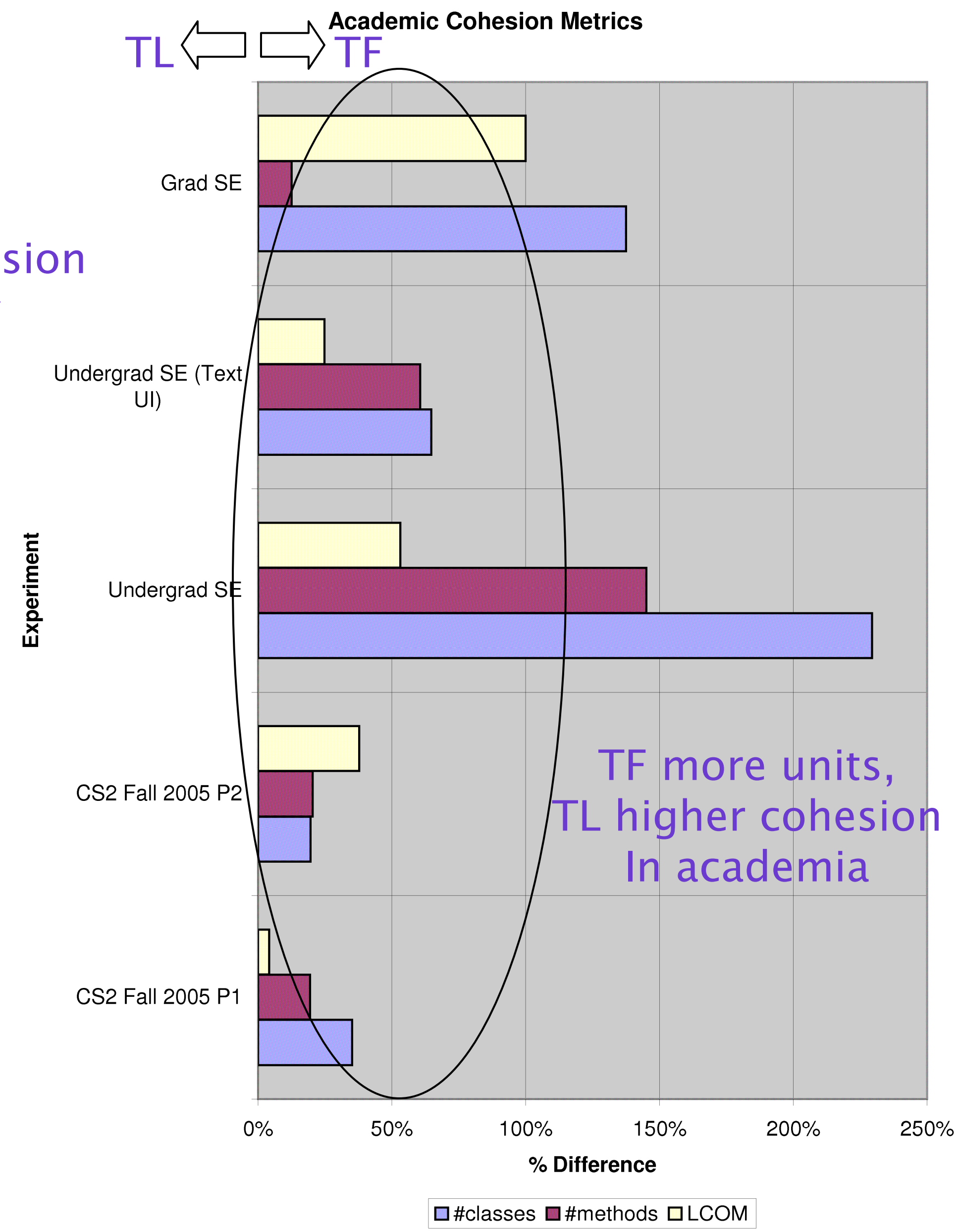




Cohesion Results

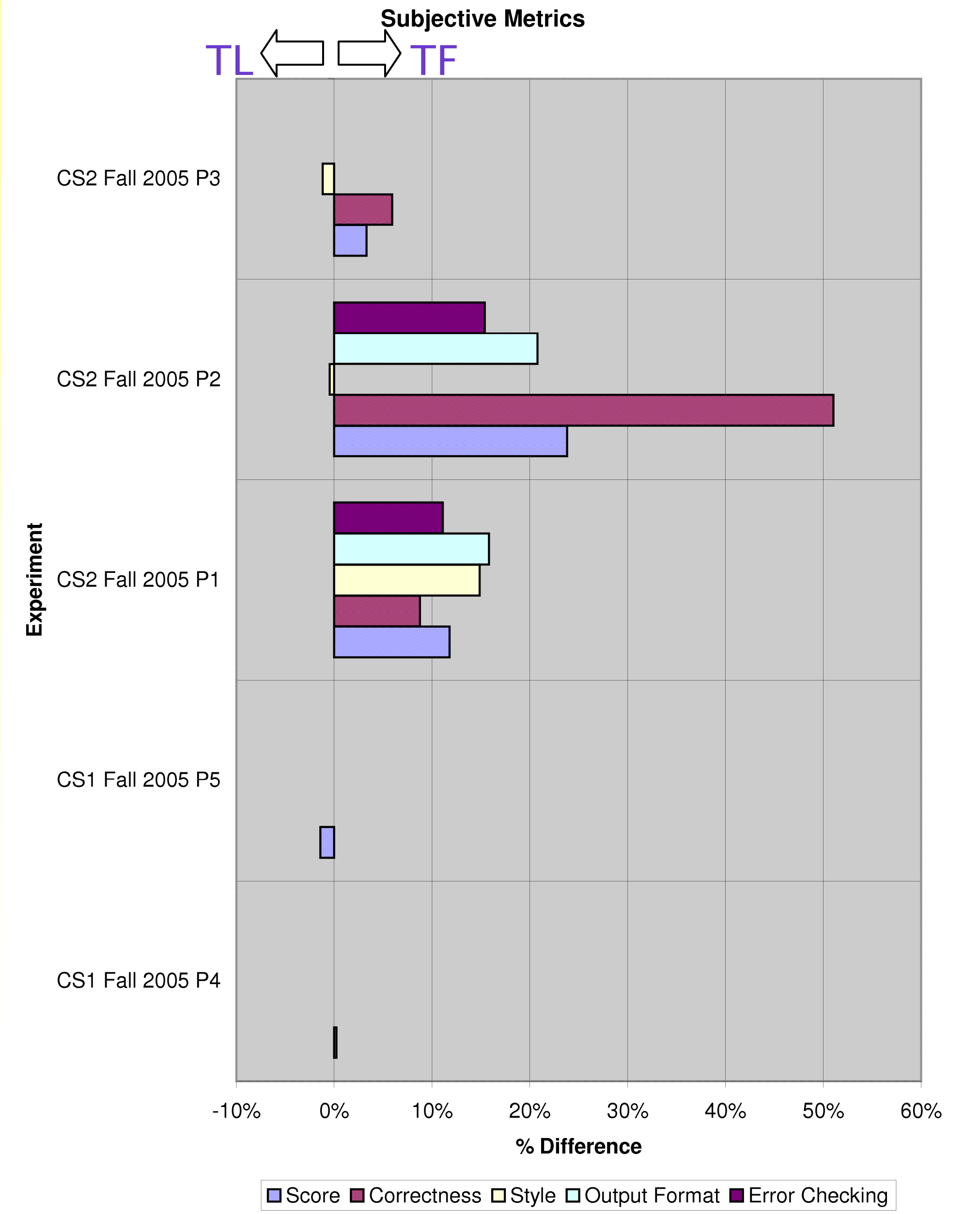
Test-first has higher cohesion ⇦ ⇨ Test-last has higher cohesion





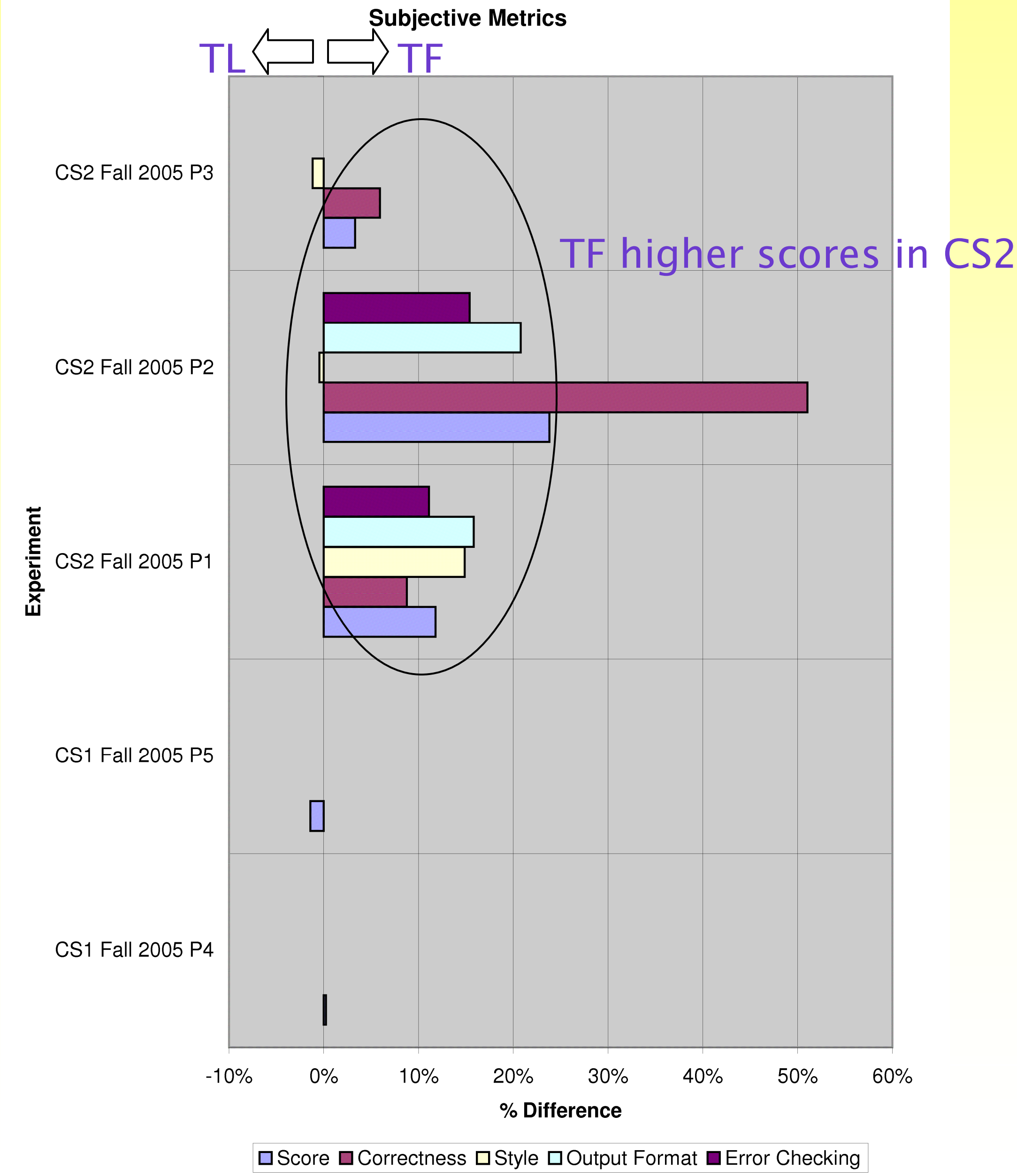
Subjective Evaluation Results

Test-last has higher scores ⇦ ⇨ Test-first has higher scores



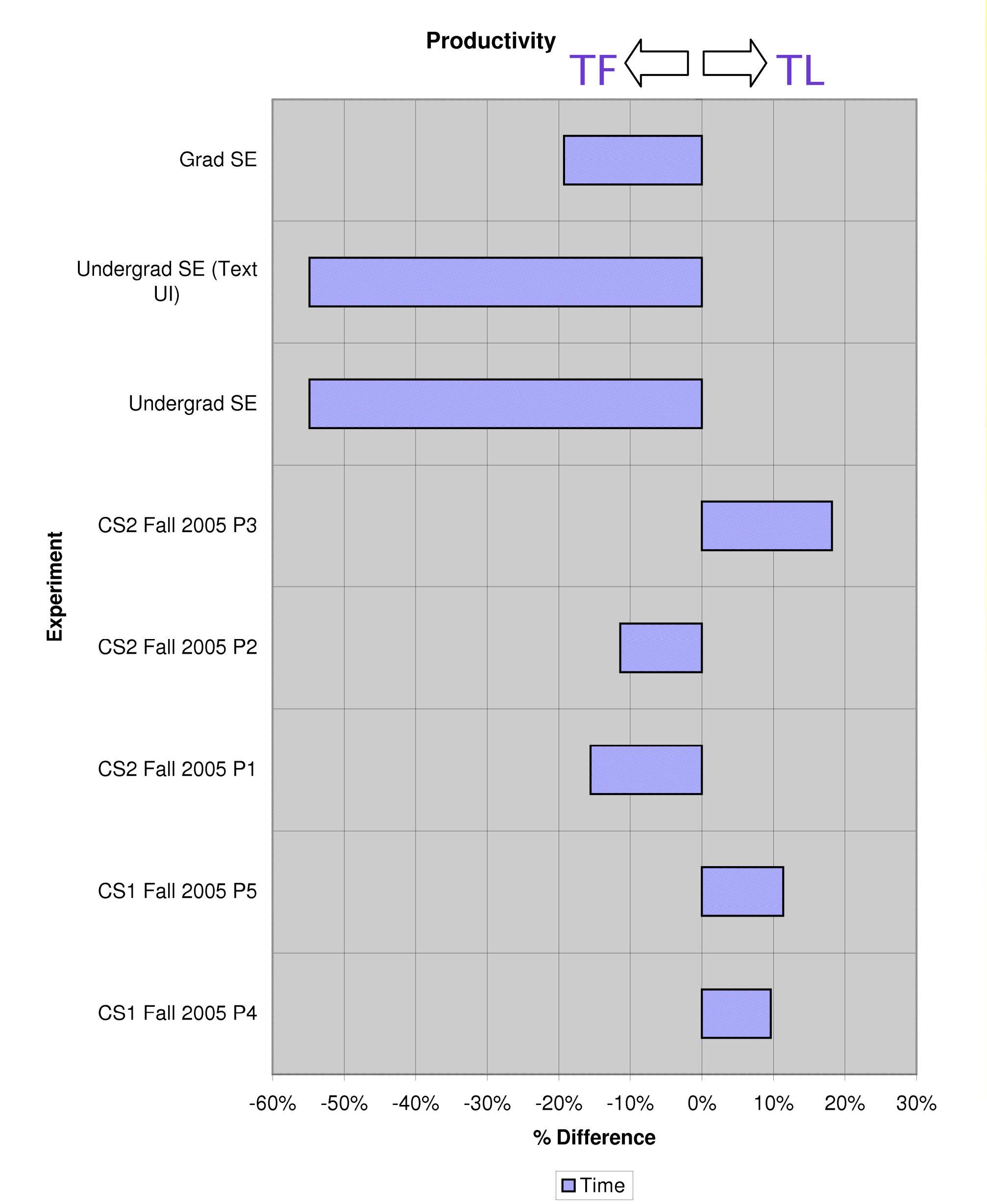
Subjective Evaluation Results

Test-last has higher scores ⇦ ⇨ Test-first has higher scores



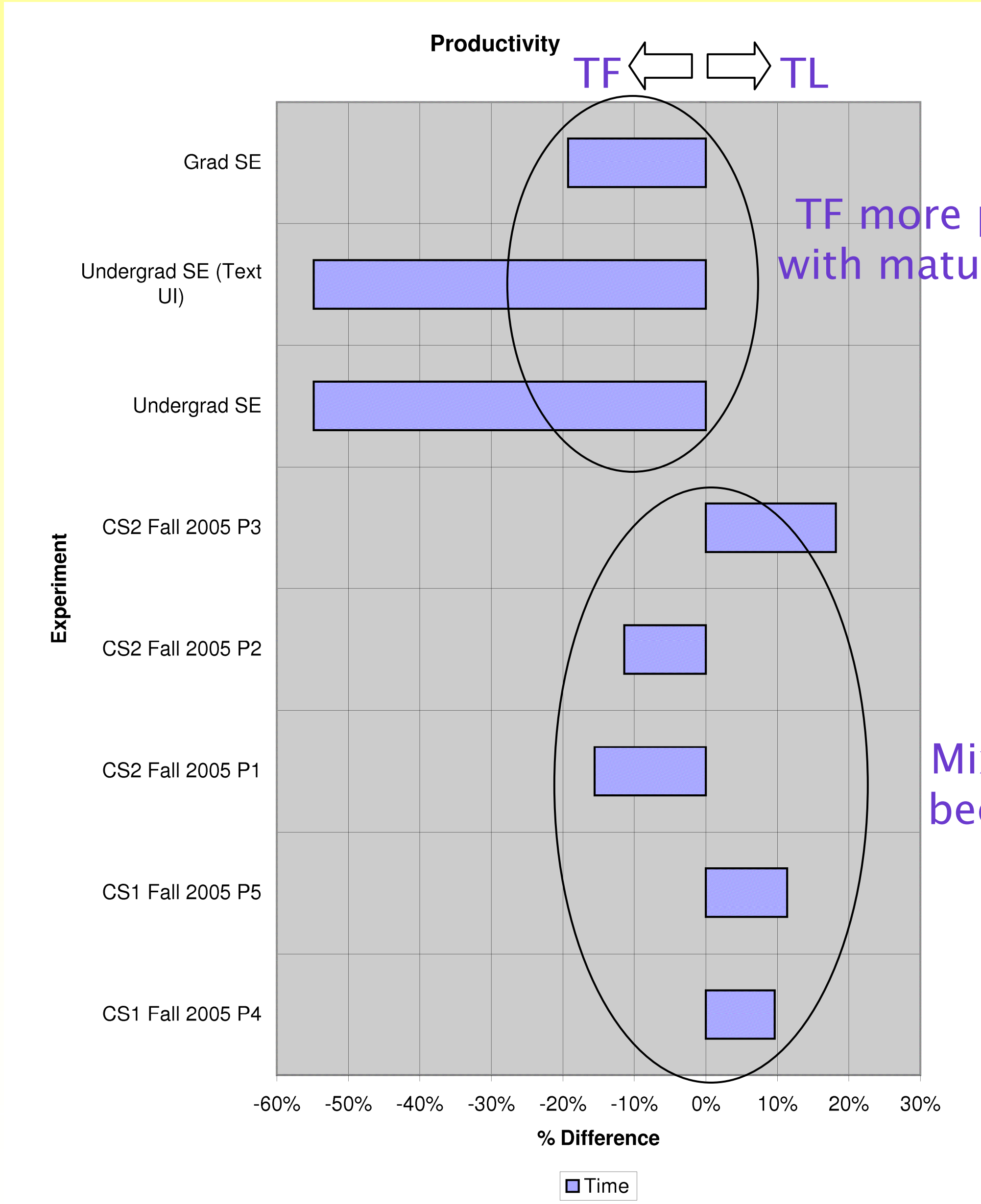
Productivity Results

Test-first was more productive ⇦ ⇨ Test-last was more productive



Productivity Results

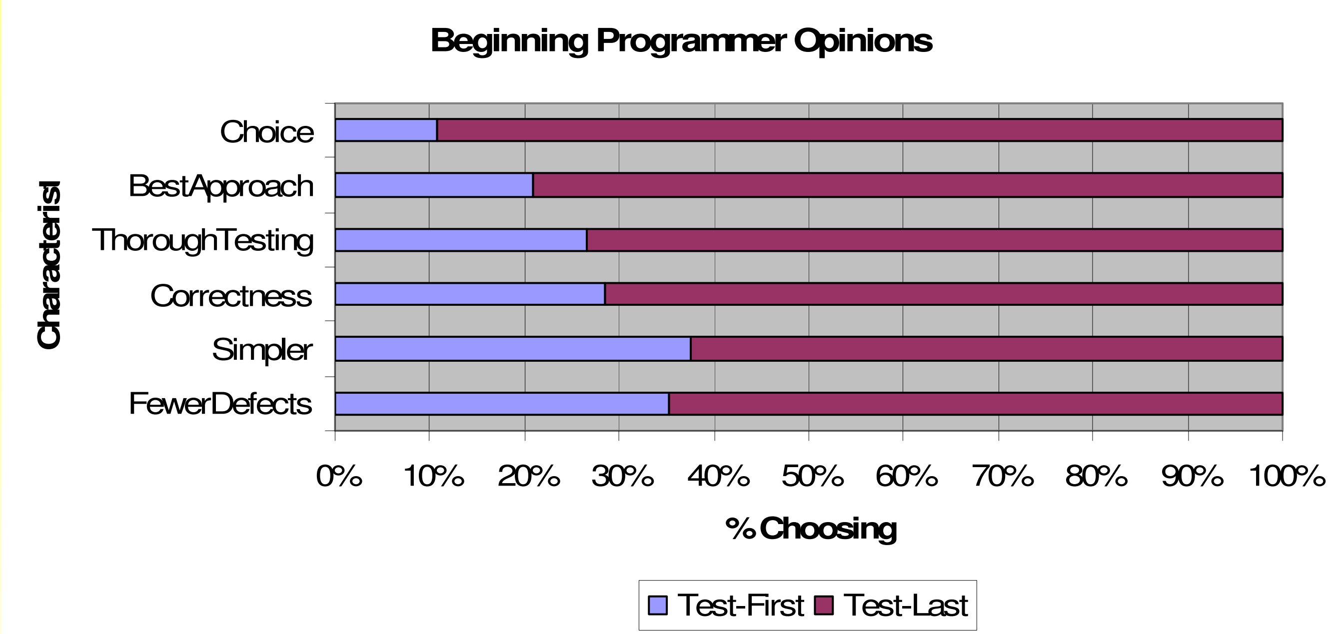
Test-first was more productive ⇦ ⇨ Test-last was more productive

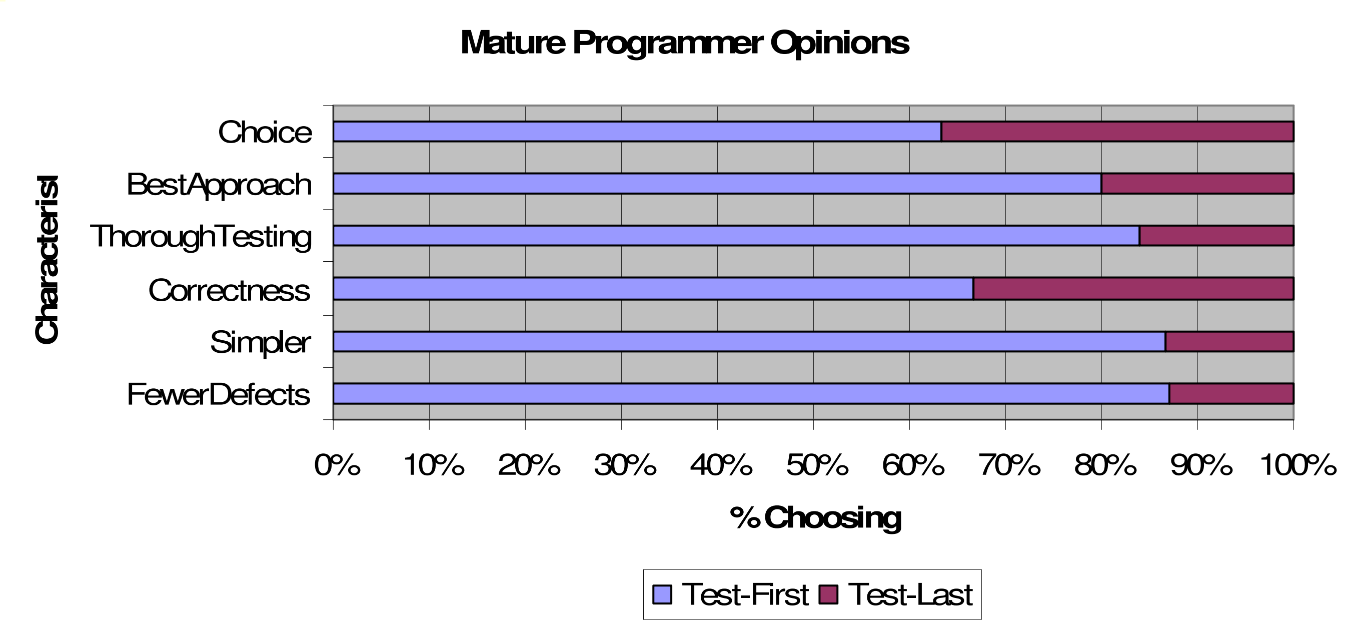


TF more productive with mature students

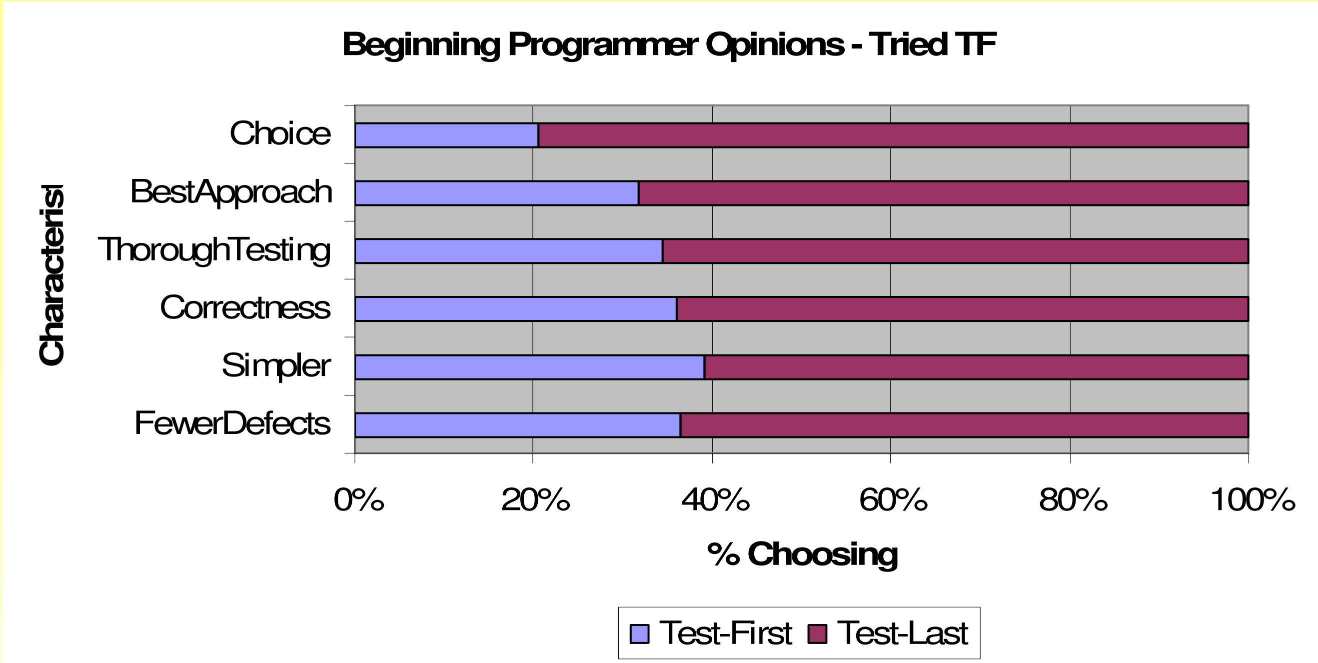
Mixed results with beginning students

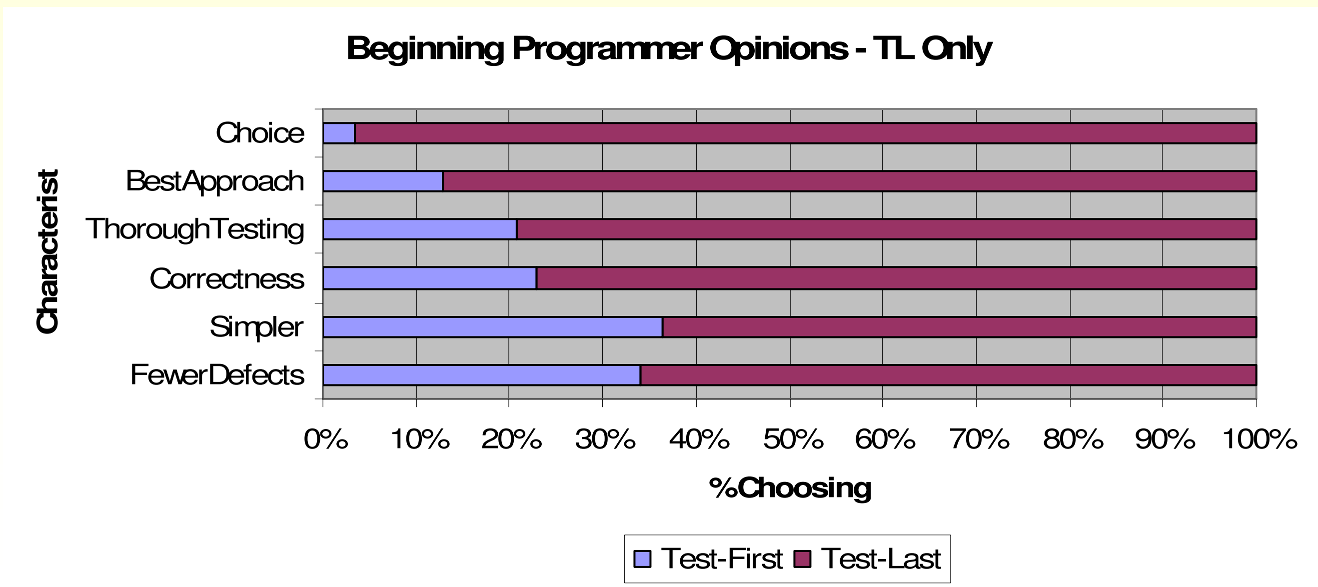
Programmer Opinions



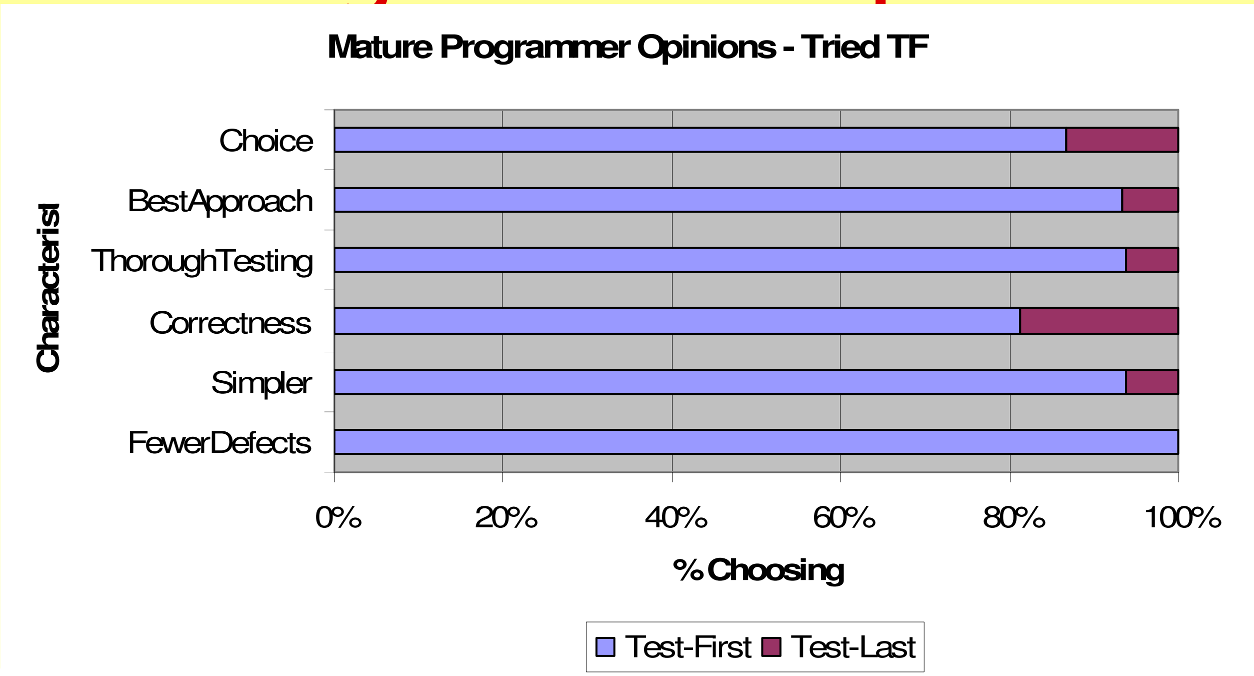


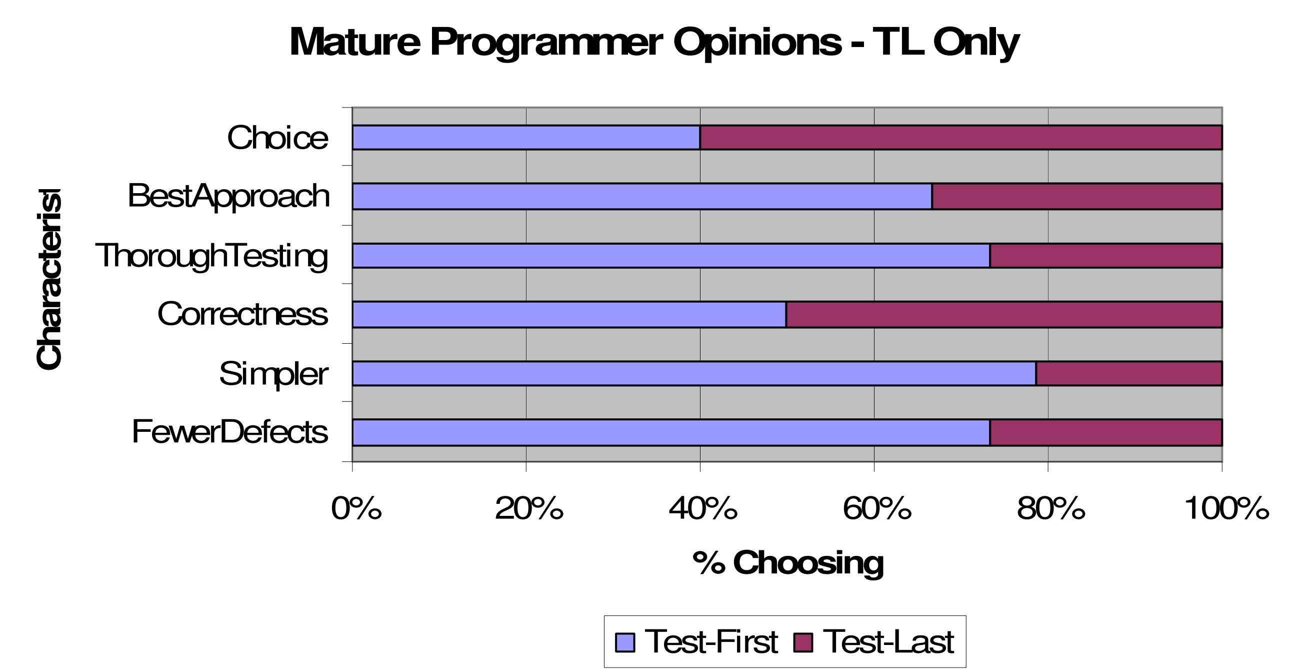
Beginning Programmer Opinions





Mature Programmer Opinions





Organization

- Problem Statement
- Background
- Research Methodology
- Evaluation and Results
- Conclusions and Future Work

Quality Comparison Chart

Experiment	Complexity	Coupling	Cohesion	Size	Testing
CS1 Fall 2005 P4	TL				TF
CS1 Fall 2005 P5	TF				TL
CS2 Fall 2005 P1	TL	TL	TL	TL	TF
CS2 Fall 2005 P2	TL	TL	TL	TL	TF
CS2 Fall 2005 P3	TL				TF
CS2 Spr 2006 P1	TL				TL
CS2 Spr 2006 P2	TL				TL
CS2 Spr 2006 P3	TL				TL
Undergrad SE	TF	TL	TL	TF	TF
Undergrad SE (Text UI)	TF	TL	TL	TF	TF
Grad SE	TF	Mixed	TL	TF	TF
Industry Bowling	TF				
Industry Case Study	TF	TL	TF	TF	TF
Industry 3 (TF/TL)	TL	TF	TF	Mixed	TF
Industry 2 (TL/TF)	TL	TF	TF	Mixed	TF
Industry 1 (No-Tests/TF)	TF	TL	TL	TF	TF

Quality Comparison Chart Clusters

Experiment	Complexity	Coupling	Cohesion	Size	Testing
CS1 Fall 2005 P4	TL				TF
CS1 Fall 2005 P5	TF				TL
CS2 Fall 2005 P1	TL	TL	TL	TL	TF
CS2 Fall 2005 P2	TL	TL	TL	TL	TF
CS2 Fall 2005 P3	TL				TF
CS2 Spr 2006 P1	TL				TL
CS2 Spr 2006 P2	TL				TL
CS2 Spr 2006 P3	TL				TL
Undergrad SE	TF	TL	TL	TF	TF
Undergrad SE (Text UI)	TF	TL	TL	TF	TF
Grad SE	TF	Mixed	TL	TF	TF
Industry Bowling	TF				
Industry Case Study	TF	TL	TF	TF	TF
Industry 3 (TF/TL)	TL	TF	TF	Mixed	TF
Industry 2 (TL/TF)	TL	TF	TF	Mixed	TF
Industry 1 (No-Tests/TF)	TF	TL	TL	TF	TF

Conclusions

1. Mature developers applying the test-first approach are likely to write *less complex code* than they would write with a test-last approach.
2. Mature developers applying the test-first approach are likely to write *more smaller units* (methods and classes) than they would write with a test-last approach.
3. Developers at all levels applying the test-first approach are likely to write *more tests* and achieve *higher test coverage* than with a test-last approach.
4. Mature developers who have applied both the test-first and test-last approach are more likely to *choose the test-first approach*.

Future Work

- Replicate experiment in additional environments
- Replicate experiment with beginning developers using Java
- Examine residual effects of TDD
  - For how long do TDD programmers sustain high test-coverage and quality effects?
  - Are residual effects better with continued test-first and test-last use?
- Does a more comprehensive TDL approach improve beginning programmer acceptance and quality?
- Examine various levels of up-front architecture/design detail
- Compare TDD with a process containing formal inspections

Key References

- D. Janzen and H. Saiedian, "Test-Driven Learning: Intrinsic Integration of Testing into the CS/SE Curriculum," *Technical Symposium on Computer Science Education (SIGCSE'06)*, March, 2006, Houston, TX
- D. Janzen and H. Saiedian, "Test-Driven Development: Concepts, Taxonomy and Future Directions," *IEEE Computer*, 38(9), 2005
- D. Janzen, "Software Architecture Improvement through Test-Driven Development," *Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'05) Student Research Competition*, October, 2005, San Diego, CA
- D. Janzen, H. Saiedian, "On the Influence of Test-Driven Development on Software Design," *Conference on Software Engineering Education and Training (CSEE&T'06),* April 2006, North Shore Oahu, Hawaii
- D. Janzen, "An Empirical Examination of Test-Driven Development," ACM Student Research Competition Grand Finals Third-Place Winner, *ACM Digital Library,* May 2006

Acknowledgements

- Karen Janzen, Hossein Saiedian
- SIGCSE Special Projects Grant