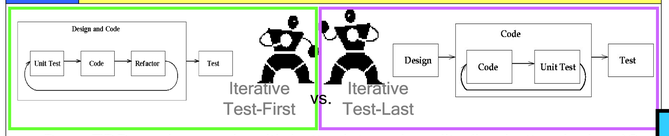
Software Architecture Improvement through Test-Driven Development

David Janzen
PhD Candidate
University of Kansas
Lawrence, KS USA

1 Problem 1: Does TDD improve Software Quality?



- Does TDD produce code with better internal quality (i.e. better designs)?
  - Is code simpler, easier to reuse, easier to maintain?
  - Does code have lower complexity, smaller classes/methods, lower coupling?
- Does TDD produce code with better external quality (i.e. fewer defects)?
- Does TDD take longer to develop the same functionality?

2 Problem 2: How do we teach TDD?

- Can undergraduates learn TDD?
- Where is the optimal level in the curriculum to teach TDD?
  - From the beginning (CS1)?
  - After students know some basics (CS2)?
  - Once students have some maturity (Software Engineering)?
  - In the field (Industry training)?
- Can tests be integrated throughout the curriculum?
  - Test-Driven Learning (TDL) is an approach to teach with tests
    - Is TDL effective?

3 Related Work: Focus on *defects* and *external quality*

TDD in Academia

Study Author(Year)	# of Programmers	Quality Effects	Productivity Effects
Edwards(2003)	59	54% fewer defects	n/a
Kaufmann(2003)	8	improved information flow	50% improvement
Muller(2002)	19	no change, but better reuse	no change
Pancur(2003)	38	no change	no change
Erdogmus(2005)	35	no change	improved productivity

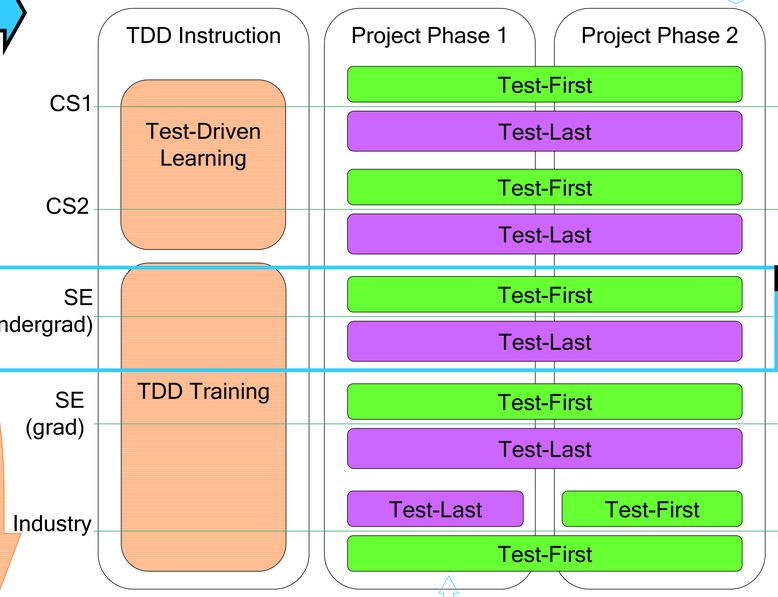
TDD in Industry

Study Author(Year)	# of Companies	# of Programmers	Quality Effects	Productivity Effects
George(2004)	3	24	TDD passed 18% more tests	TDD took 16% longer
Maximilien(2003)	1	9	50% reduction in defect density	minimal impact
Williams(2003)	1	9	40% reduction in defect density	no change

Related experiments focus on external quality (defect density), and indicate potential benefits of TDD with negligible productivity effects.

4 Solution: Conduct empirical studies that focus on *design* and *internal quality*

- Conduct formal experiments in academic and professional settings.
- Isolate TDD as independent variable.
- Experiment group develops software with Test-First approach.
- Control group develops software with Test-Last approach.



Measure productivity with total hours and hours per feature.

Measure reuse From one phase/ Iteration to another.

Measure internal design quality with size, complexity, structural, and OO metrics.

Measure external quality with defect density.

All academic programmers will develop the same application within a class. Industry programmers will complete different projects so we will compare Test-First projects with previous Test-Last projects by same team.

Test-Driven Learning (TDL) integrates automated unit-tests into traditional pedagogical examples, thus modeling good testing throughout the curriculum.

Traditional Example

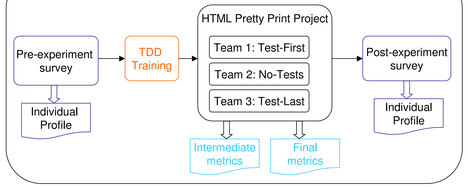
```
#include <iostream>
using namespace std;
int sum(int min, int max) {
  int sum = 0;
  for(int i=min;i<=max;i++)
    sum += i;
  return sum;
}
int main() {
  cout << sum(3,7) << endl;
  // above should print 25
}
```

TDL Example

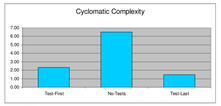
```
#include <cassert>
int sum(int min, int max) {
  int sum = 0;
  for(int i=min;i<=max;i++)
    sum += i;
  return sum;
}
int main() {
  assert(sum(3,7)==25);
}
```

5 Initial Results: Experiment in Undergraduate Software Engineering Course

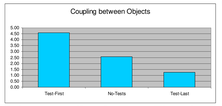
Experiment Design



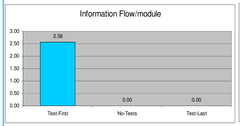
Software Metrics



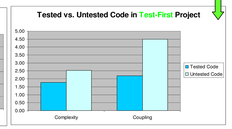
Tested code was simpler



Higher coupling in Test-First, but...



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



Code covered by tests had lower complexity and coupling in Test-First team

Testing Metrics

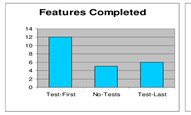
	Test LOC	% Classes Tested	Assertions/SLOC	Test Coverage (lines)	Test Coverage (branches)
Test-First	168	38.46%	0.086	31.00%	43.00%
No-Tests	0	0.00%	0.000	0.00%	0.00%
Test-Last	38	25.00%	0.045	29.00%	23.00%

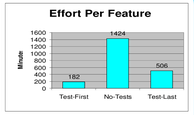
- Testing metrics from Phase 1 (text UI)
- Test-First wrote more tests and covered more code with tests
- No-Tests team was assigned to be Test-Last but "ran out of time for tests"

Context

- Undergraduate Software Engineering survey course
- Semester-long team programming project
- Phase 1: text UI, Phase 2: Graphical UI
- Summer 2005

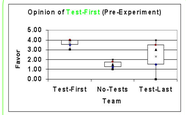
Productivity

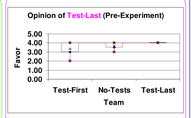




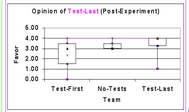
- Test-First spent 88% less effort/feature than No-Tests
- Test-First spent 57% less effort/feature than Test-Last
- Only Test-First completed both phases

Student Perceptions





Opinions of TF improved



Opinions of TL declined

Threats to Validity

- Small sample size (ten students)
- Test-First team had slightly more Java experience
- Majority of core code implemented by one developer on each team

6 Conclusions

- Test-First team was more productive
- Test-First team produced more Object-Oriented Design
- Test-First team wrote more tests
- Tested code was less complex and more decoupled
- Test-Last team produced more decoupled code
- Test-First perceptions improved with use while Test-Last perceptions worsened
- Additional industry and academic studies are warranted

7 You can contribute

Industry Professionals
- Contribute project data
- Email djanzen@ku.edu

Academic Scholars
- Adopt TDL
- Replicate studies

8 Acknowledgements

- Dr. Hossein Saiedian, advisor