Software Quality Overview

“Upstream”/“Downstream” Defect Removal Activities

Testing vs. Defect Removal

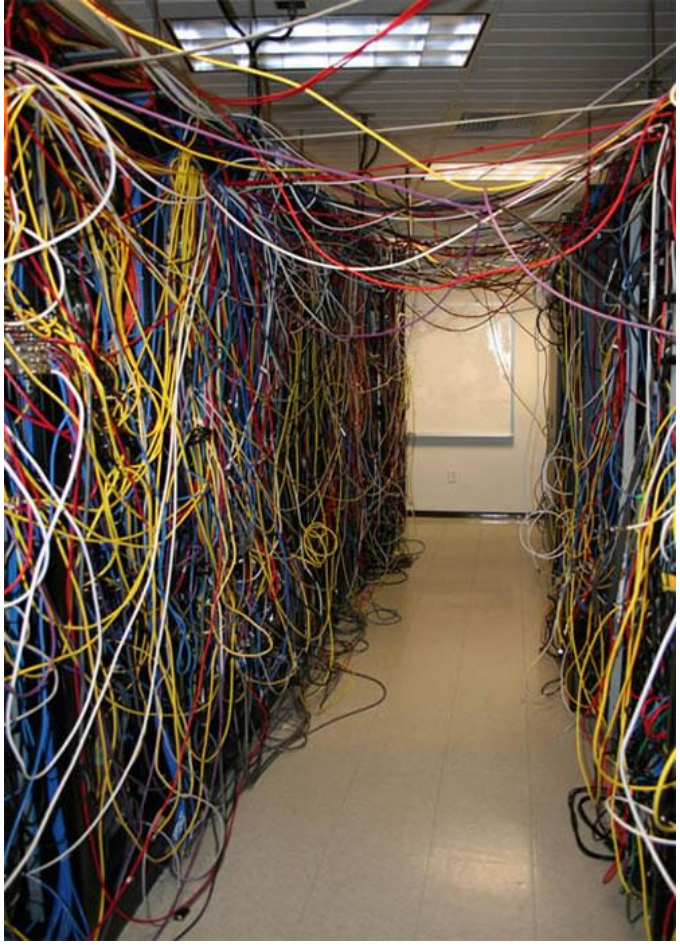
Testing vs. Defect Removal

- *Testing* can constrain the software quality discussion to the execution of a program for the purpose of removing defects
- *Defect Removal* emphasizes a broader context of activities including
  - pair programming
  - code reviews
  - test-driven development
  - static analysis

- Ideally… what we really want is… *Defect Prevention*!

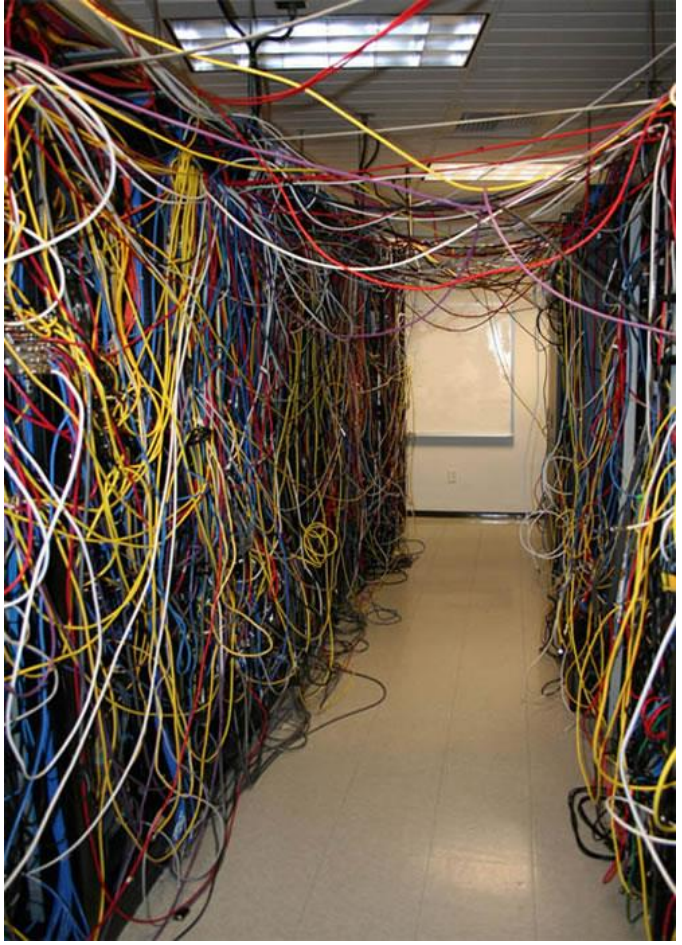
What is a *Defect*?

What is a *Defect*?



Servers are used as a metaphor for software

What is a *Defect*?

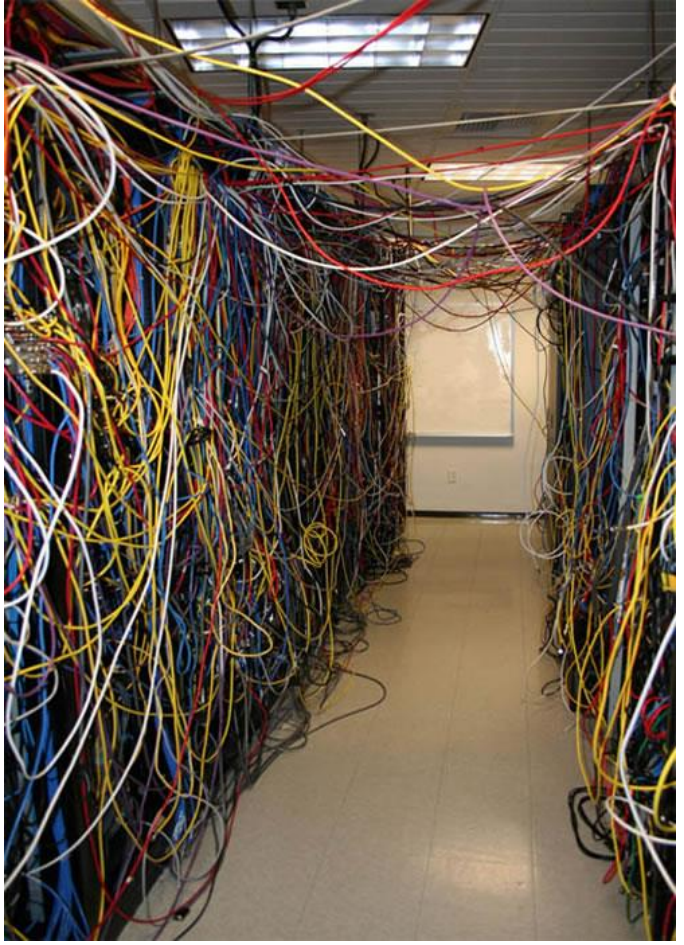


Engineering perspective:

- High Quality "Structure"
- Maintainable

- Low Quality "Structure"
- Difficult to maintain

What is a *Defect*?



Engineering perspective:

- High Quality "Structure"
- Maintainable

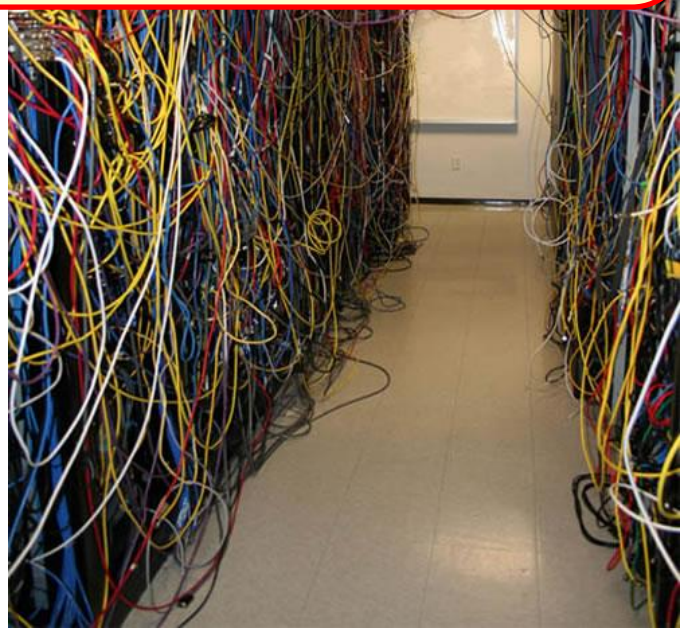
- Low Quality "Structure"
- Difficult to maintain

Client perspective (assumption):

- Does NOT produce ALL expected output

- Produces expected output

Which software will be "preferred" by the customer?



Engineering perspective:

- High Quality "Structure"
- Maintainable

- Low Quality "Structure"
- Difficult to maintain

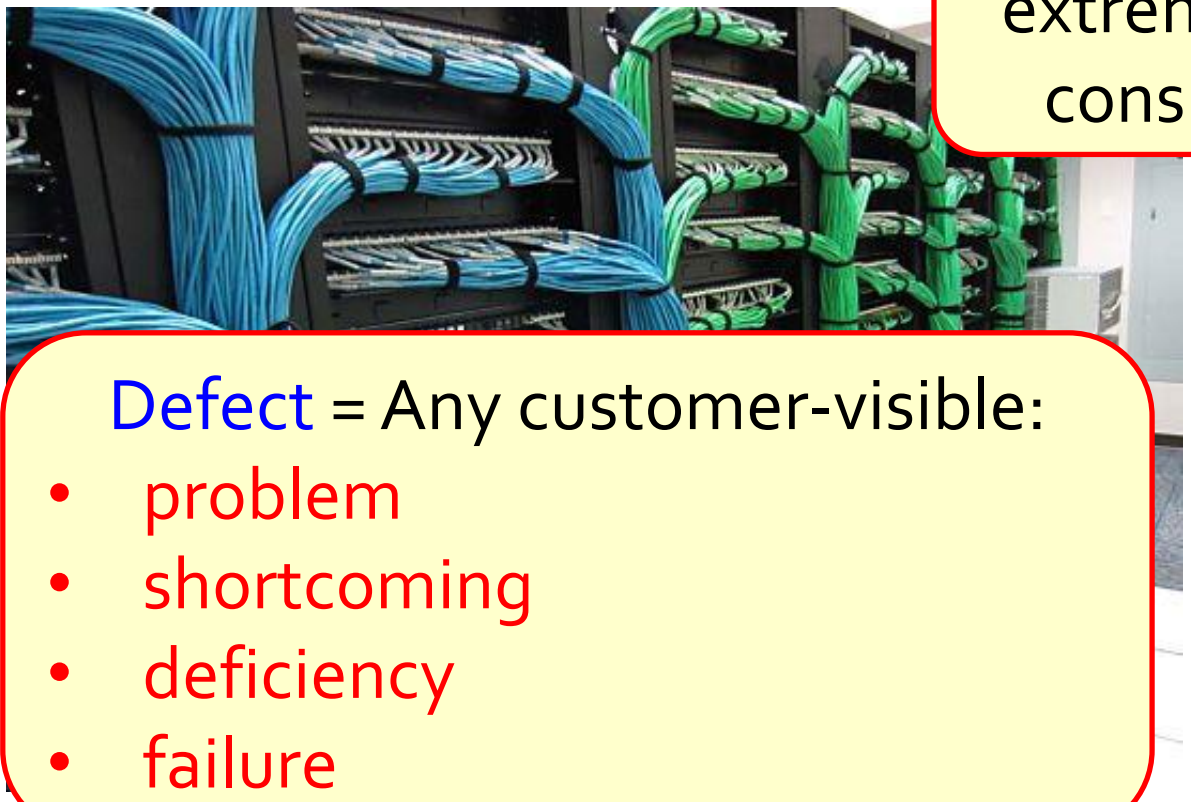
Client perspective (assumption):

- Does NOT produce ALL expected output

- Produces expected output

What is a *Defect*?

"Code Health" (which is extremely important) is not considered a defect…yet



Defect = Any customer-visible:

- problem
- shortcoming
- deficiency
- failure

Engineering perspective:

- High Quality "Structure"
- Maintainable

- Low Quality "Structure"
- Difficult to maintain

Client perspective (assumption):

- Does NOT produce ALL expected output

- Produces expected output

Verification and Validation

Verification and Validation

- *Verification* refers to whether the product does what the development team expects it to do (i.e., does the project implement its specification?)
  - "are be building the product correctly"?
    - Example specification: "function add should return the sum of two values" ⇒ engineering team focuses on whether function add provides the correct sum

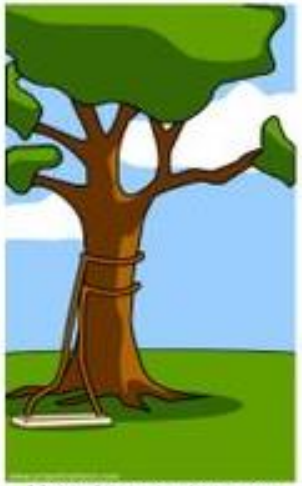
Verification and Validation

- *Verification* refers to whether the product does what the development team expects it to do (i.e., does the project implement its specification?)
  - "are be building the product correctly"?
    - Example specification: "function add should return the sum of two values" ⇒ engineering team focuses on whether function add provides the correct sum
- *Validation* refers to whether the product does what the customer needs it to do (i.e., does the project satisfy the customer's needs?)
  - "are we building the correct product?"
    - Example: "function add may have incorrect specifications"

How the customer explained it

How the project leader understood it

How the analyst designed it



How the programmer wrote it

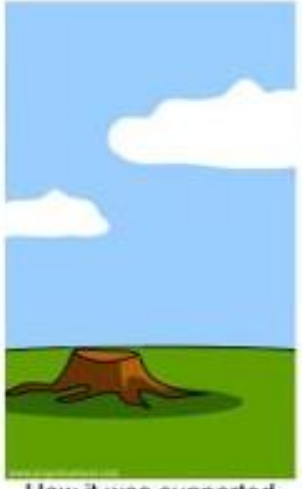
What the beta testers received

How the business consultant described it

How the project was documented

What operations installed

How the customer was billed



How it was supported

What marketing advertised

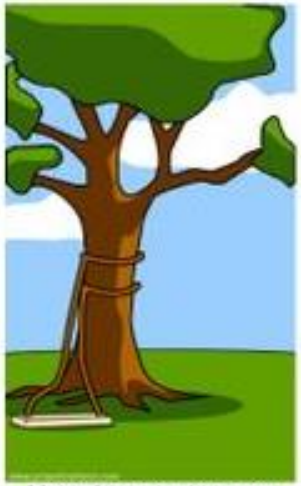
What the customer really needed

Specification


How the customer explained it


How the project leader understood it


How the analyst designed it


How the programmer wrote it


What the beta testers received


How the business consultant described it


How the project was documented


What operations installed


How the customer was billed

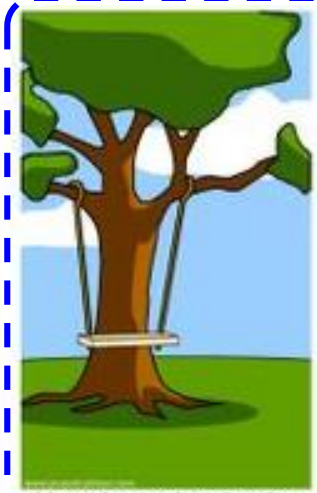

How it was supported

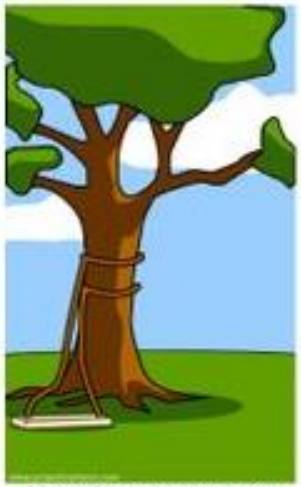
What marketing advertised


What the customer really needed

Specification

Verification (engineering)

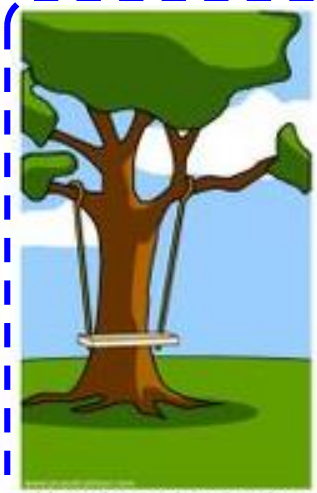




Specification

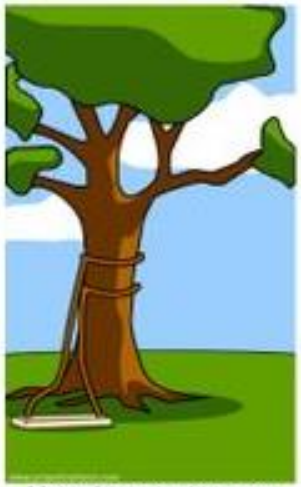
Verification (engineering)

How the customer explained it



How the project leader understood it

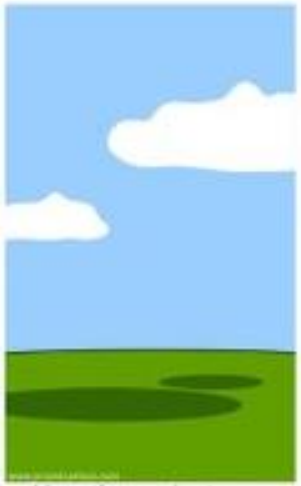
How the analyst designed it



How the programmer wrote it

What the beta testers received

How the business consultant described it



How the project was documented

What operations installed

How the customer was billed

How it was supported

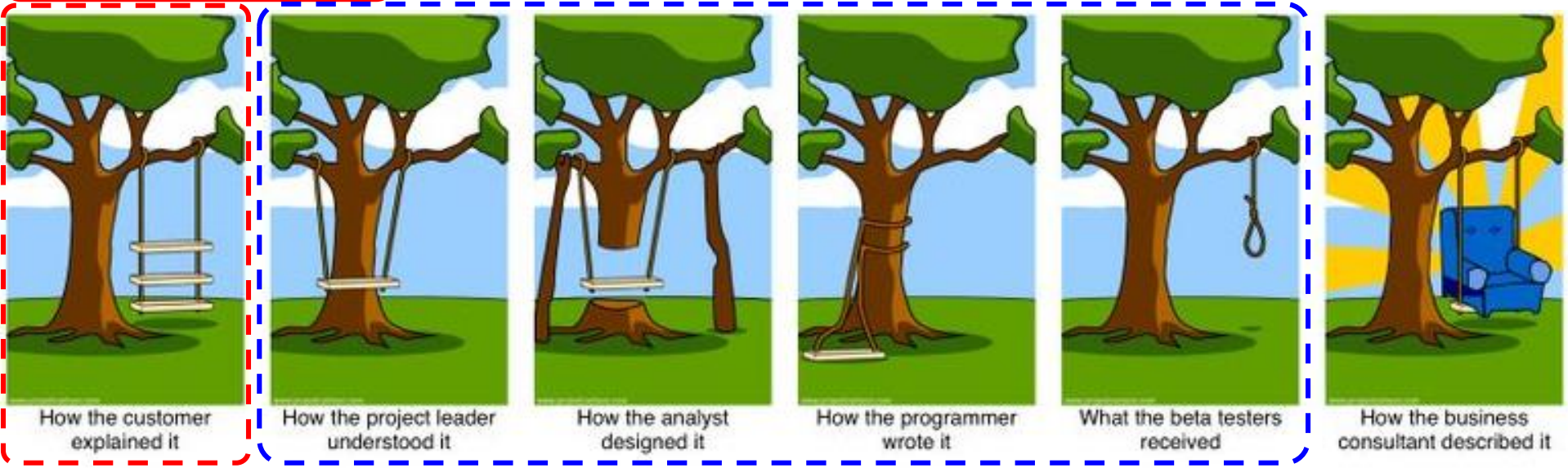
What marketing advertised

What the customer really needed

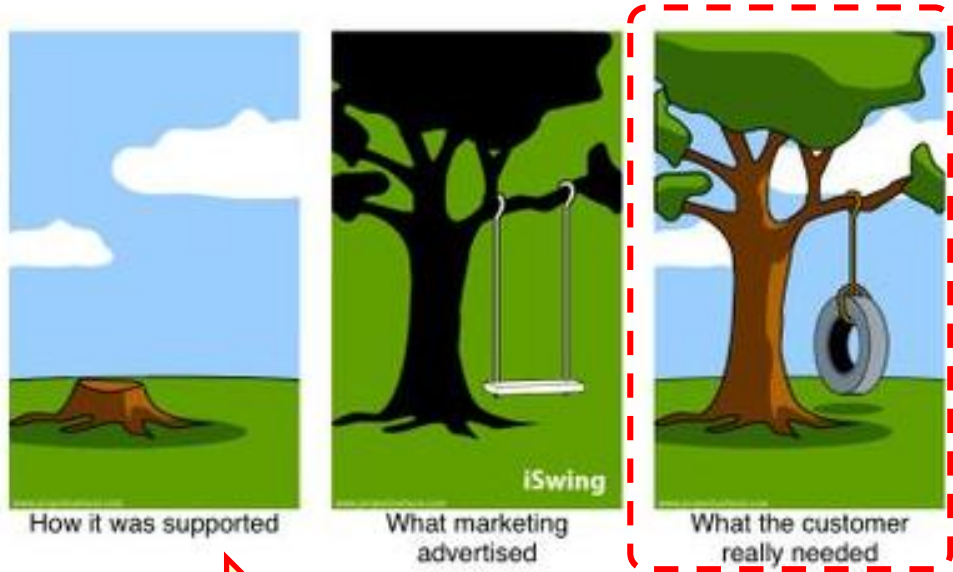
Validation (client)

Specification

Verification (engineering)



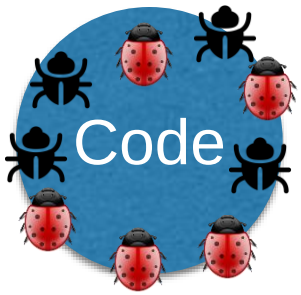
NB: Most defect removal activities focus on verification. But which do you think matters most to the customer?



Validation (client)

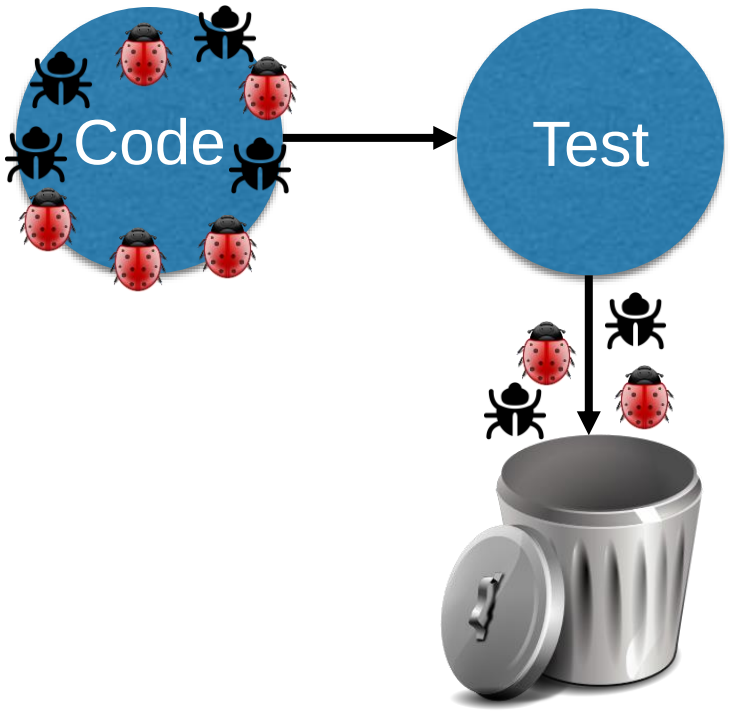
Modeling Defect Removal

Modeling Defect Removal



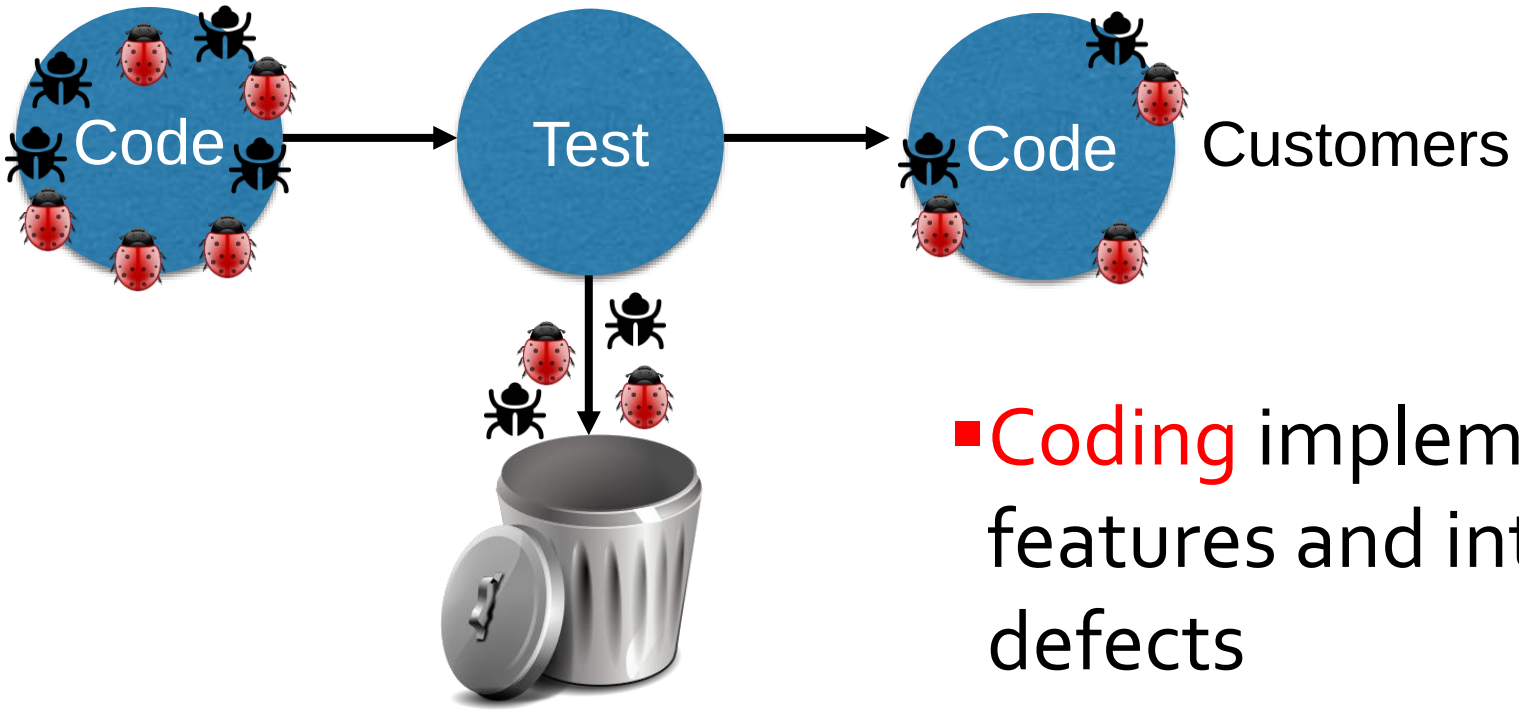
- Coding implements features and introduces defects

Modeling Defect Removal



- Coding implements features and introduces defects
- Testing removes some defects and preserves (most) features

Modeling Defect Removal



- Coding implements features and introduces defects
- Testing removes some defects and preserves (most) features
- We deliver features and defects to customers

Defect Removal Economics

- Cost ($$$) to remove a defect rises with the time it remains in the code

Defect Removal Economics

- Cost ($$$) to remove a defect rises with the time it remains in the code

A defect found by …	might be removed in …
Pair Programming	seconds
Unit-Level testing	minutes
Acceptance Testing	about 1 day
Customers	days, weeks, months!

Defect Removal Economics

- If my team can remove defects faster than your team, we can invest a larger fraction of our available effort delivering features ⇒ generate more $$$

Measuring Software Quality: Defect Density

Measuring Software Quality: Defect Density

- Defect Density = the number of defects confirmed in software/module during a specific period of operation or development divided by the size of the software/module

- Defect Density is the most common instrument of quality today
  - Expressed as NumberOfDefects / ThousandLinesOfCode
  - e.g., 5 defects/KLOC

Measuring Software Quality: Defect Density

- Defect Density Applications:
  - For comparing the relative number of defects in various software components
  - identify high-risk components for resource allocation
- Yes… there are shortcomings with this instrument
- Better instruments remain a research thread

Software Quality Examples

Product	Defect Density (defects/KLOC)	Source
Apache 2.1	0.53	http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.145.1586&rep=rep1&type=pdf
IBM Server Product	0.81	Kan, S. Metrics and Models in Software Quality Engineering: 2nd Edition. Addison Wesley. 2003
MySQL 4.0.16	0.09	http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.145.1586&rep=rep1&type=pdf
Space Shuttle Avionics	0.10	http://www.cs.colostate.edu/~malaiya/p/li98.pdf
Typical Application	5.25	Jones1996
Mobile Web App	4.75	Williams2011
Unknown Infrastructure	2.81	Williams2011

What's a Reasonable Goal for CS471?

What's a Reasonable Goal for CS471?

- Of course it depends on the project

- If you have no other direction, consider these ballpark figures:
  - Client side code typically delivers about 5 defects/KLOC*
  - Server side code may need better than 2 defects/KLOC

*96Jones, 92Grady

What's a Reasonable Goal for CS471?

- Life-critical software (avionics, aerospace, medical equipment, automotive, self-driving cars, etc.) needs even better quality (<0.1 - 1 defect/KLOC)

- Note: The shift to software-as-a-service may change the focus from *defect density* to *availability*

Defect Removal Effectiveness

- No single defect removal activity is perfectly effective
  - No one activity will remove all the defects from our product
  - No one activity is sufficient by itself (for most products)
- Solution?

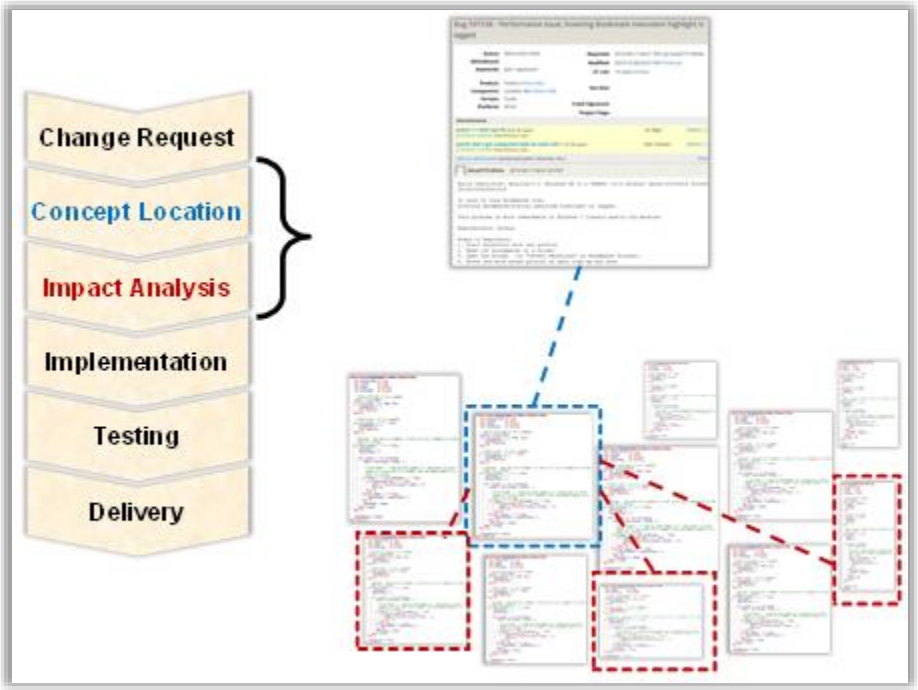
Defect Removal Effectiveness

- To obtain quality higher than that offered by a single activity, we construct a defect removal model consisting of a pipeline of multiple activities

- Real-world projects will chain
  - pair programming
  - unit-level testing
  - acceptance testing
  - beta testing, etc.

Defect Finding vs. Removal

- Software testing merely finds defects
- We are focused on removing defects (which requires feature location, impact analysis, etc.)
- Once again… defect removal directs our attention to our real-world goal… actually removing defects from the code



Defect Removal Activities in CS471

Examined in CS471	Not Examined in CS471
Pair Programming	Proof of Correctness
Unit-Level Testing	Code Inspections
Test-Driven Development	Reliability Testing
Static Analysis	Usability Testing
Informal Code Reviews	Stress (Load) Testing
Integration Testing	Fuzzing
Regression Testing	Performance Testing
Acceptance Testing	Design Reviews
Beta Testing	

Defect Removal: Example Order of the Activities

1. Pair Programming (PP) and Test-Driven Development (TDD)
2. (Execution of the) Unit-Level Test (UT)
3. Static Analysis (SA)
4. Code Review (CR)
5. Integration/Regression Test (RT)
6. System-Level (Acceptance) Test (AT)
7. Beta Test (BT)

Estimated Overall Defect Removal Effectiveness

	Estimated Overall Effectiveness	Approximate Delivered Density
AT	50%	17 Defects/KLOC
UT + AT	70%	10 Defects/KLOC
PP + UT + AT	75%	9 Defects/KLOC
PP+ UT + TDD + AT	82%	6 Defects/KLOC
PP+UT+TDD+SA+AT	87%	5 Defects/KLOC
PP+UT+TDD+SA+CR+AT	90%	3.5 Defects/KLOC
PP+UT+TDD+SA+CR+RT+AT	90%	3.4 Defects/KLOC
PP+UT+TDD+SA+CR+RT+AT+BT	93%	2.4 Defects/KLOC

Important Assumptions: Coding=35 Defects/KLOC, PP=15%, UT=40%, TDD=30%, SA=25%, CR=25%, RT=4%, AT=50%, BT=40%