Software Engineering and Verification Research Trends

2008-12-16 @ SAAB / RUAG Space

Robert Feldt

BTH & Chalmers & SWELL R. School (swell.se)

Overview

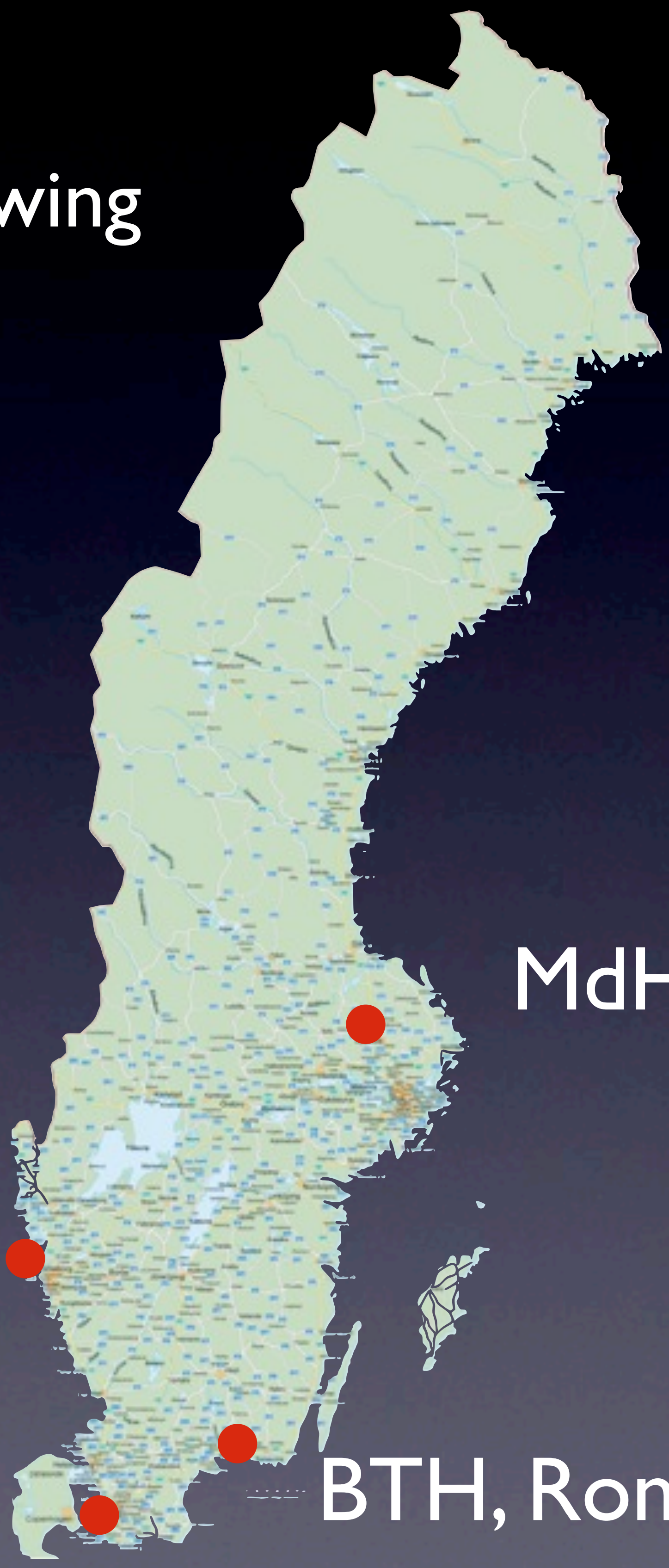
- BTH & SWELL
- "Smorgasbord" of a number of fairly recent results:
    - Selective, Homeworkless reviews
    - Capture / Re-capture models
    - Acceptance tests for clarifying requirements
    - Empirical evaluations of Test-Driven Development
    - PEX and CHESS
    - ICST 2009 statistics
    - Testability in practice
    - Test-Case Driven Inspections
    - Current study here: ECSS and more effective V&V

BTH/SERL

- SERL = Swedens largest SE research group
  - Req Eng, Automated V&V, Empirical
- 1 Professor (top 5 in world), 6 PhDs, 8 PhD students
- BTH = Blekinge Tekniska Högskola
  - Focused on IT & Sustainability
  - Largest number of international students
  - Bachelor SE, MSc, Master SE, EuroMaster SE

SWELL - Swedish V&V Excellence

Research School
7 PhD students and growing
4 Universities
10+ Companies



Reviews and Inspections

- Well established (both theory & practice) as
  - Most efficient (defects found / manhour)
  - Very cost effective
  - 80-90% of defects found with 25% less effort (than other V&V)
- Still: Not many use it!
- Two recent results:
  - Capture/Re-capture Models
  - Selective, Homeworkless reviews

Spectrum of Review Techniques

Technique	Key characteristics	Preparations
Formal	Rigorous, Planned, Many roles, Documented, Re-work / Re-review	Extensive
Team review	Roles, Some preparation, Less rigorous	Some
Walkthrough	Only some roles prepare	Maybe
Pair programming	Direct/Online, Only 2 roles	-
Peer desk check	Indirect, No preparation, Only 1	-
Ad hoc	Anything goes	-

Selective, Homeworkless Reviews

- Developed at IBM Haifa Labs 2003-
    - Presented 2008 (Farchi and Ur, ICST 2008)
- Problems:
    - People do not use reviews enough
    - With shorter lead times, reviews are skipped
- Problems are Organizational rather than Technical
- How get them to take seriously & do continuously?

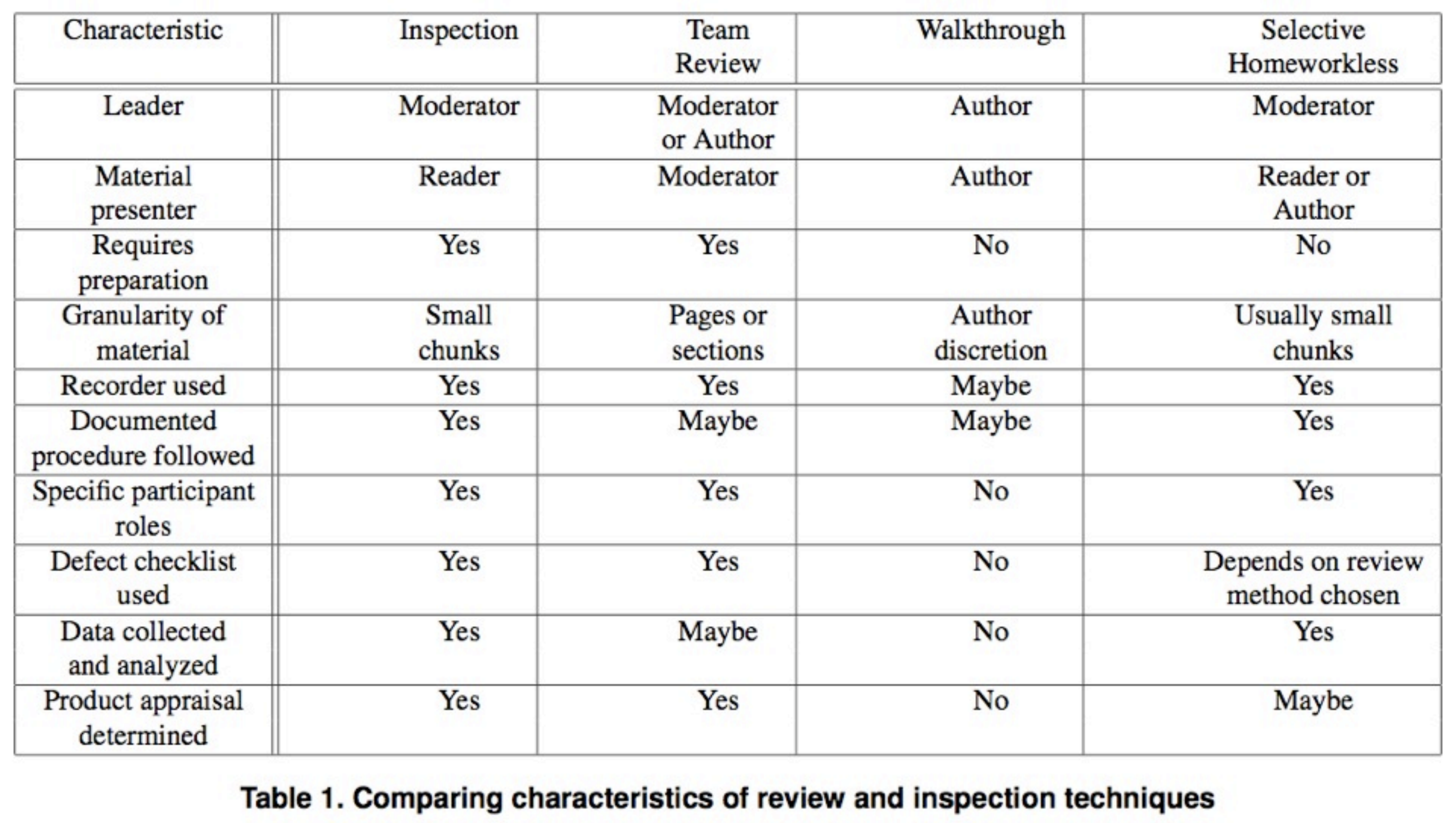
Selective, Homeworkless Reviews

- Solutions:
  - Don’t review everything => reduce cost
  - Artifact selection process => focus where effective
  - Homeworkless, no preparations => less time&cost
    - Specific versions for different artifacts
  - Fixed time part of regular project schedule (1.5 hours / week)
- Results:
  - Review rates similar to inspections (100-150Loc/hour)
  - 2.2 +/- 0.34 issues found / personhour (Fewer but major)
  - Much reduced costs

Selective, Homeworkless Reviews

- Changes:
  - Moderator prepares 15 minutes (select artifact & review tech.)
  - No one else prepares
  - Fixed time every week for reviews
  - Artifact-specific review techniques

Comparing SHRs to traditional



Characteristic	Inspection	Team Review	Walkthrough	Selective Homeworkless
Leader	Moderator	Moderator or Author	Author	Moderator
Material presenter	Reader	Moderator	Author	Reader or Author
Requires preparation	Yes	Yes	No	No
Granularity of material	Small chunks	Pages or sections	Author discretion	Usually small chunks
Recorder used	Yes	Yes	Maybe	Yes
Documented procedure followed	Yes	Maybe	Maybe	Yes
Specific participant roles	Yes	Yes	No	Yes
Defect checklist used	Yes	Yes	No	Depends on review method chosen
Data collected and analyzed	Yes	Maybe	No	Yes
Product appraisal determined	Yes	Yes	No	Maybe

Table 1. Comparing characteristics of review and inspection techniques

Artifact Selection Process

- Two main techniques used
  - Concern-based artifact selection
    - Create Table: (rows = artifacts) x (cols = “concerns”)
    - Prioritize artifacts & select top
    - Any relevant concern can be used
  - Test selection
    - Choose tests/scenarios (with normal test selection techn.)
    - Review code by “manually” executing tests

Concern-based artifact selection

Artifact	"First of a kind"	"Complex Synchronization"	...	"Developer experience"
Module 1 Selected 2nd!	X			High
Module 2 Not reviewed!				Low
...				
Module N Selected first!	X	X		Medium

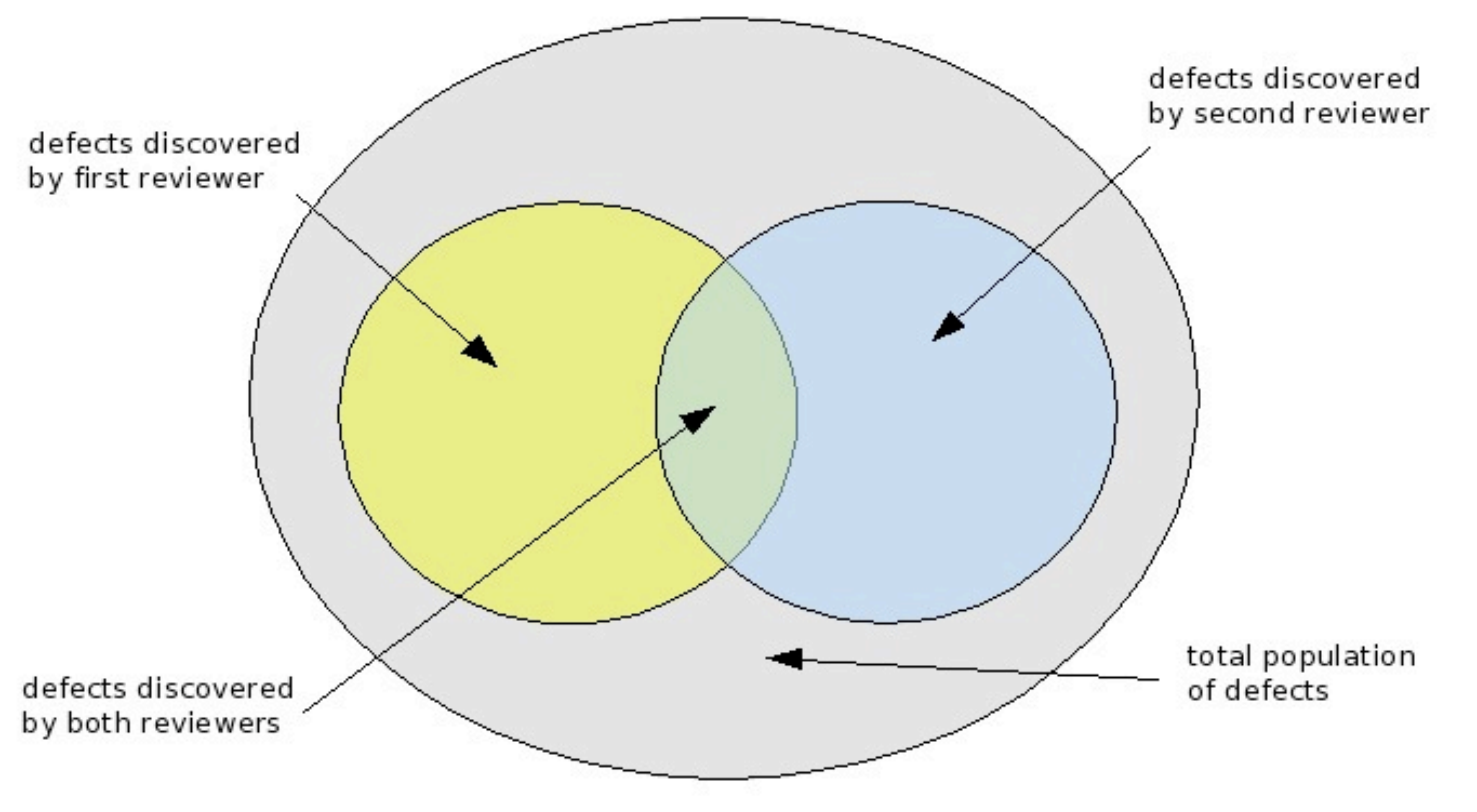
Artifact-specific review techniques used

- Paraphrasing is backbone technique
  - Sequential walkthrough while explaining logic, interface & behavior
  - Reader can be stopped when something unclear
- Contract reviews
  - Check specific obligations by “jumping around”
  - example: malloc() & free()
- Artifact comparison
  - example: code & design, design & documentation
- Checklist-based, State-machine-based, ...

Introducing SHR

- How reviews are introduced is key!
  - Without proper motivation, practice will decline
- IBM uses a “Pyramid scheme”
  - IBM Haifa experts, teaches local “champions”
  - “Champions” then teaches their peers & ensures continuity
- Teaching always based on reviewing actual artifacts
  - That persons being taught currently work with
  - “Must see value on their own problems”

Capture / Re-capture models


defects discovered by first reviewer
defects discovered by second reviewer
defects discovered by both reviewers
total population of defects

Idea: Estimate number of defects left based on overlap

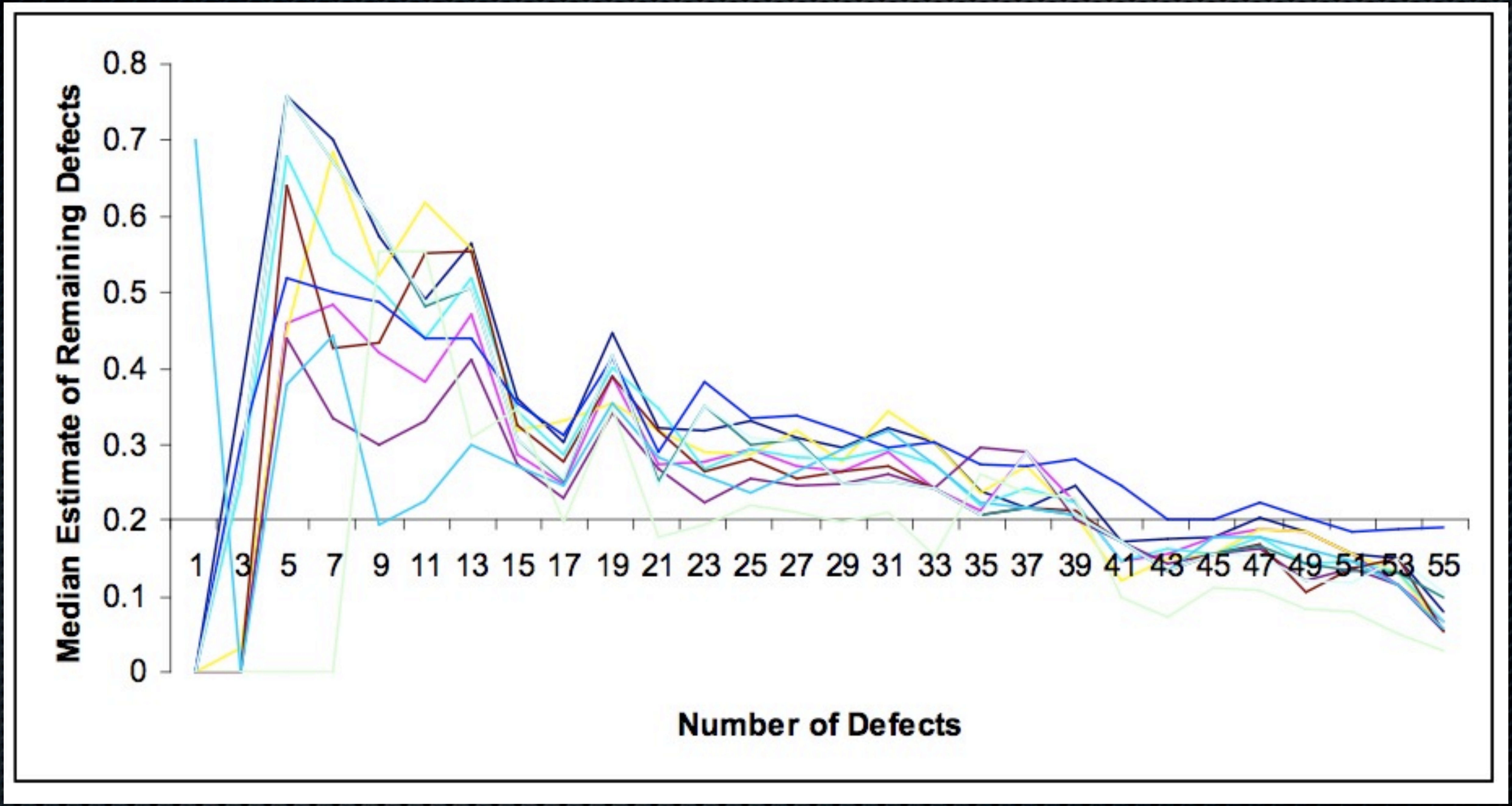
More objective than alternative methods

Suited to processes with lots of reviews

Capture / Re-capture

- Statistical method
  - Higher percentage of re-captures indicates smaller total
  - Many different estimators of remaining defects exist
- Extensive empirical research shows:
  - Min 4 independent reviewers + min 6 defects => effective
  - Adding more reviewers or having more faults has no effect
  - 60-70% of total defects need to be found
  - 20% fault in estimates remains even after best practice used

Capture / Re-capture



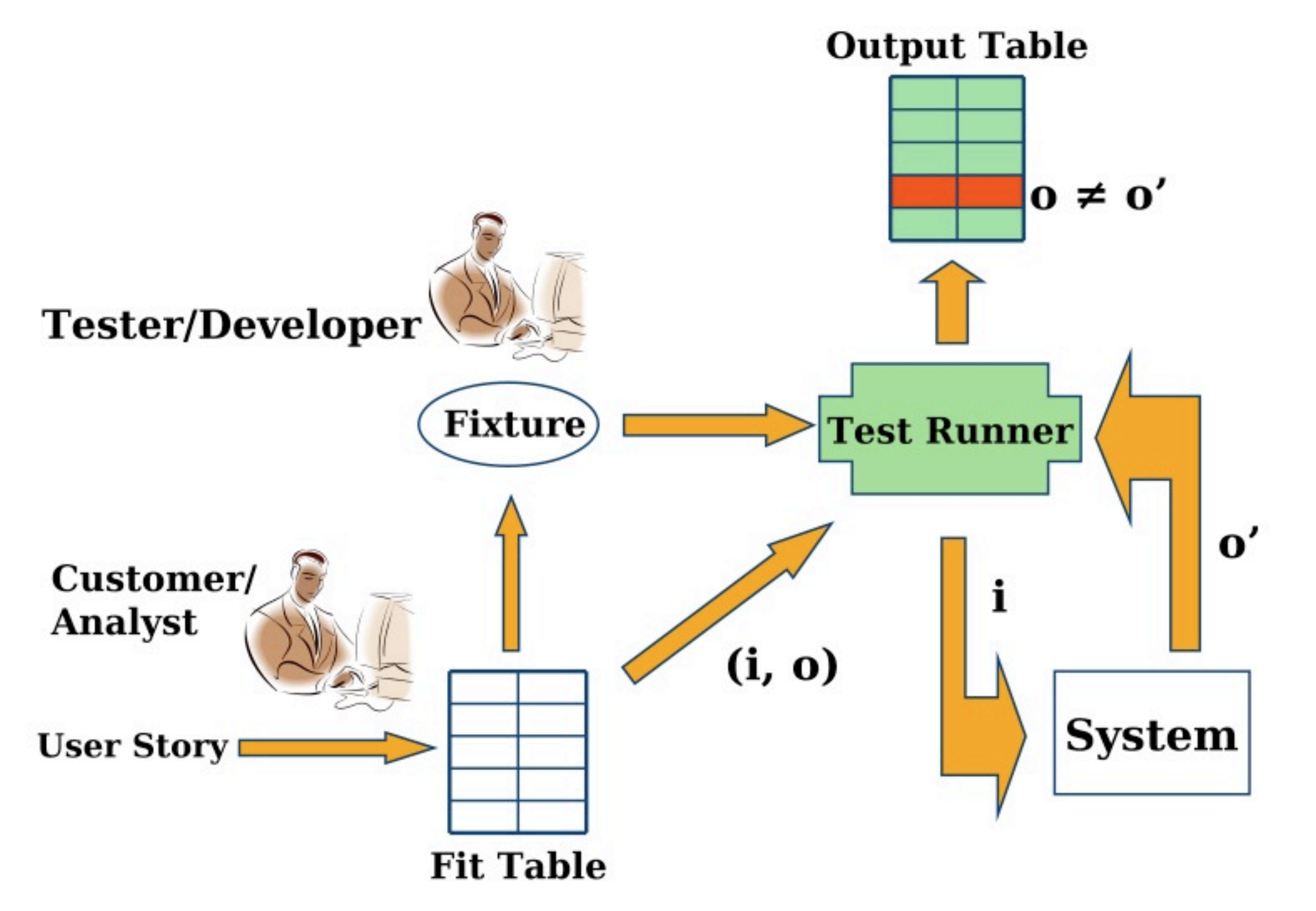
Acceptance Tests for Clarifying Requirements

- Study at two Italian universities, 30 students [1]
- Goal: Evaluate effect of FIT tables on comprehension level and effort
- Compare:
  - Group 1: Textual requirements
  - Group 2: Textual requirements + FIT tables
- Which group understood requirements best?
- Which group spent most effort?

Acceptance Testing

- Validating the systems behavior before release
- Often informal - “Demo” for customer
- Scenarios/User stories =>
- Input/output sequences for main/alternative/ exceptional paths
- FIT tables give customer easy specification format

Acceptance Testing with FIT tables



Acceptance Tests for Clarifying Requirements

- Results:

	Correct	Wrong
FIT+Text	56	34
Text	25	65

  - FIT Tables gave 400% better odds at answering requirements questions correctly
  - Same effort (i.e. no increased cost)
- However:
  - FIT tables not suited to all requirements

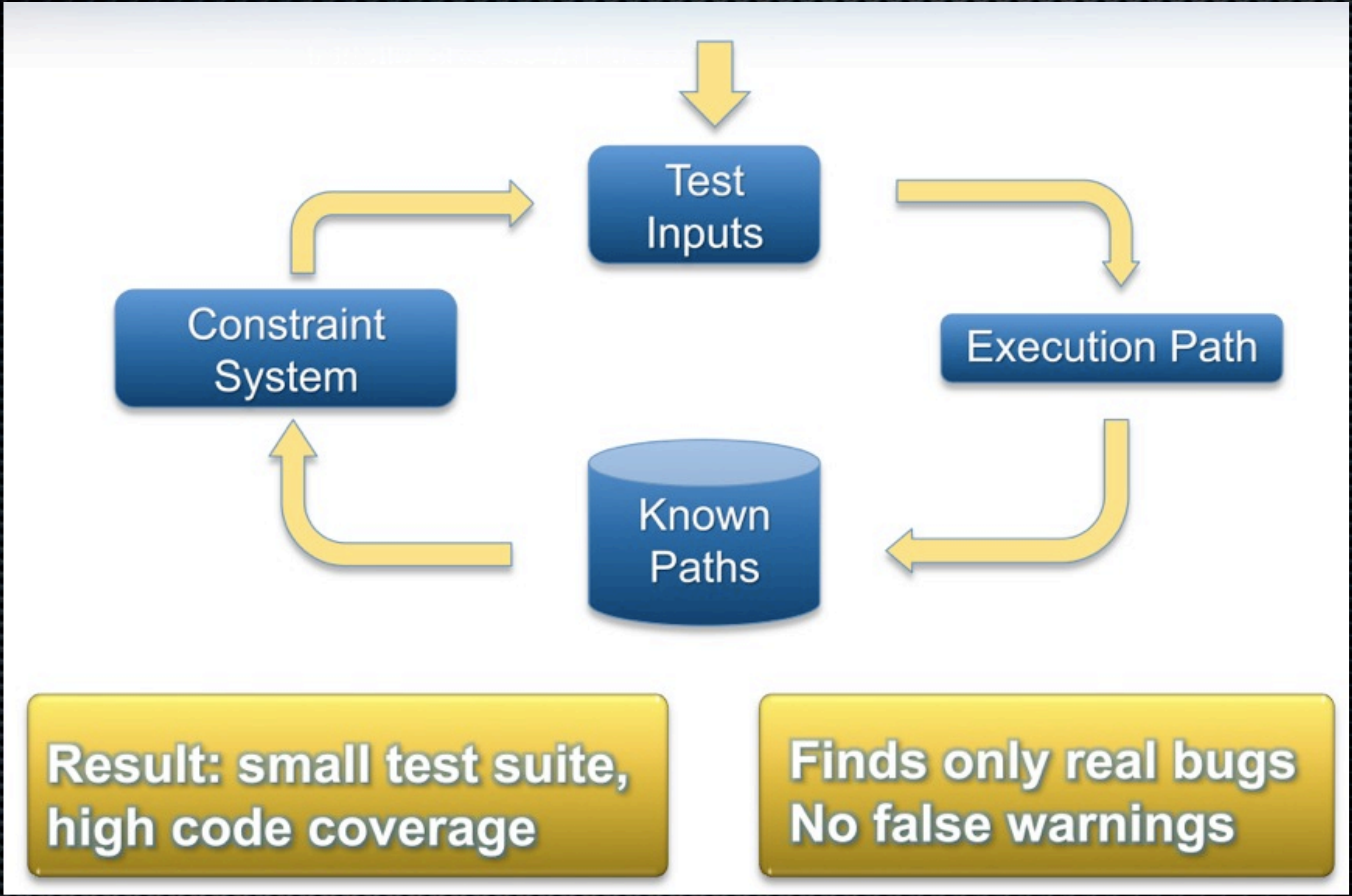
Evaluations of Test-Driven Development

- 1. Industrial TDD users [2]
  - produced code that passed 18-50% more tests
  - took 16% more time
- 2. TDD use at IBM reduced defect density 50% [3]
- Results from student experiments more mixed [1]

Automated White Box Testing

- Microsoft Research: PEX
  - Parameterized unit tests (general tests, “laws”)
  - Generate test inputs/outputs showing interesting behaviors
  - Tech: Dynamic, symbolic execution
    - Instrument (byte) code, monitor path conditions
    - Constraint solver determines inputs for paths
  - Results:
    - Found non-trivial bugs in DotNet core libraries

Pex overview



Result: small test suite, high code coverage

Finds only real bugs
No false warnings

From Microsofts “Overview of PEX” PPT

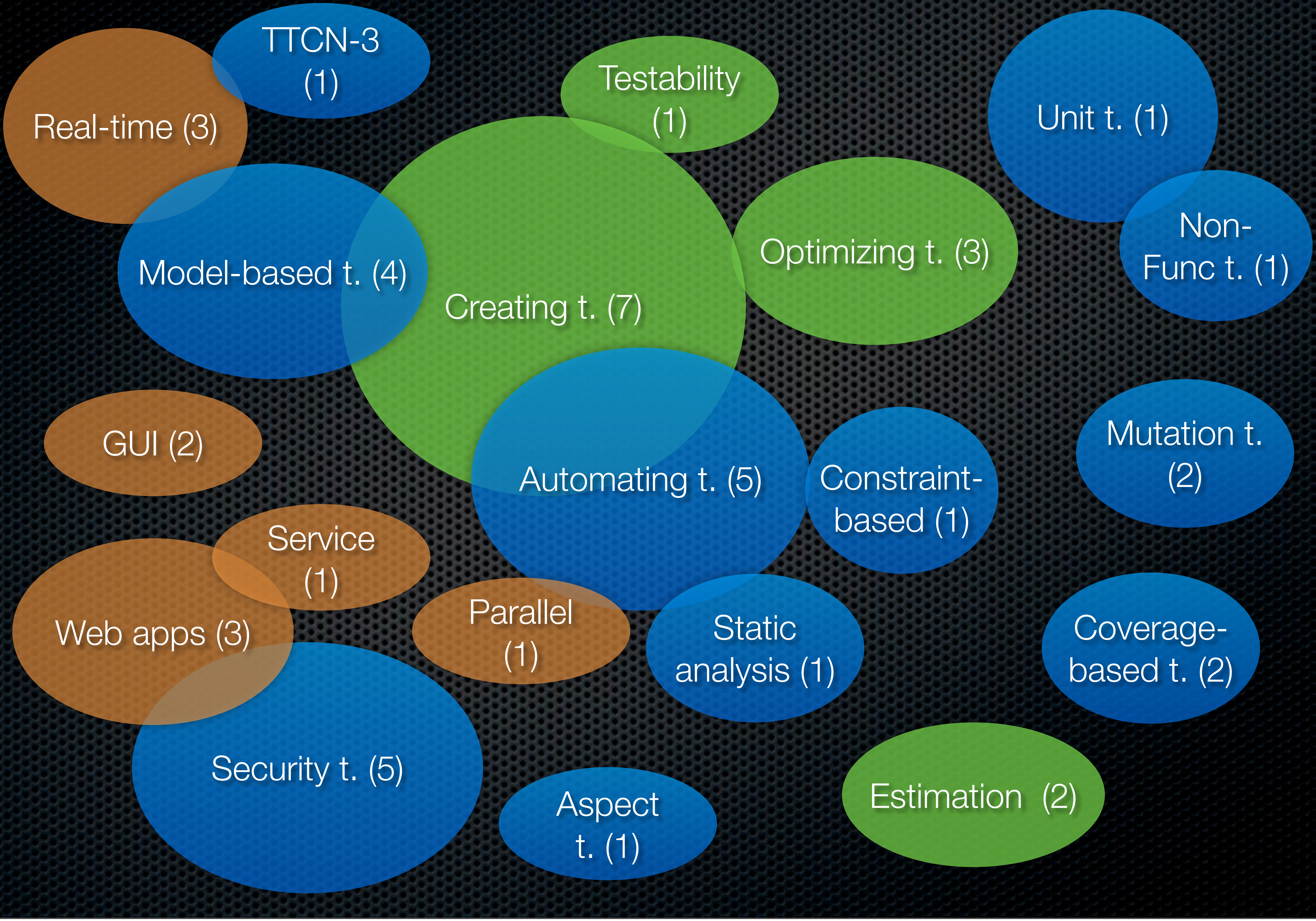
Automated Testing of Concurrent Programs

- Microsoft Research: CHESS
  - Automatically find defects in multi-threaded prgrms
  - Finds: data races, deadlocks, hangs, data-corruption access violations
- Tech: Systematic exploration of thread schedules
  - Based on model checking
- Results:
  - Found bugs in real-world, heavily stress tested software

Current study at SAAB / RUAG

- Goal: More efficient V&V
- So far:
  - Studied ECSS effects at Swedish Space Corporation
  - Will redo study here (survey + 17/12-18/12)
    - Please answer questionnaire asap if you have not already
- Then / Ongoing / Tentative:
  - Fault-Slip Through (Ericsson & SAAB Microwave) applicable?
  - More test automation?

Statistics from ICST 2009



Real-time (3)

TTCN-3 (1)

Testability (1)

Unit t. (1)

Model-based t. (4)

Creating t. (7)

Optimizing t. (3)

Non-Func t. (1)

GUI (2)

Automating t. (5)

Constraint-based (1)

Mutation t. (2)

Service (1)

Web apps (3)

Parallel (1)

Static analysis (1)

Coverage-based t. (2)

Security t. (5)

Aspect t. (1)

Estimation (2)

Testability in Practice

- Testability concepts - SOCK model
  - Simplicity - simpler components
  - Observability - exposing state
  - Control - access to all parts
  - Knowledge of expected results - behavior correct
- Not much progress in some time
- Current proposals: Checklists based on SOCK

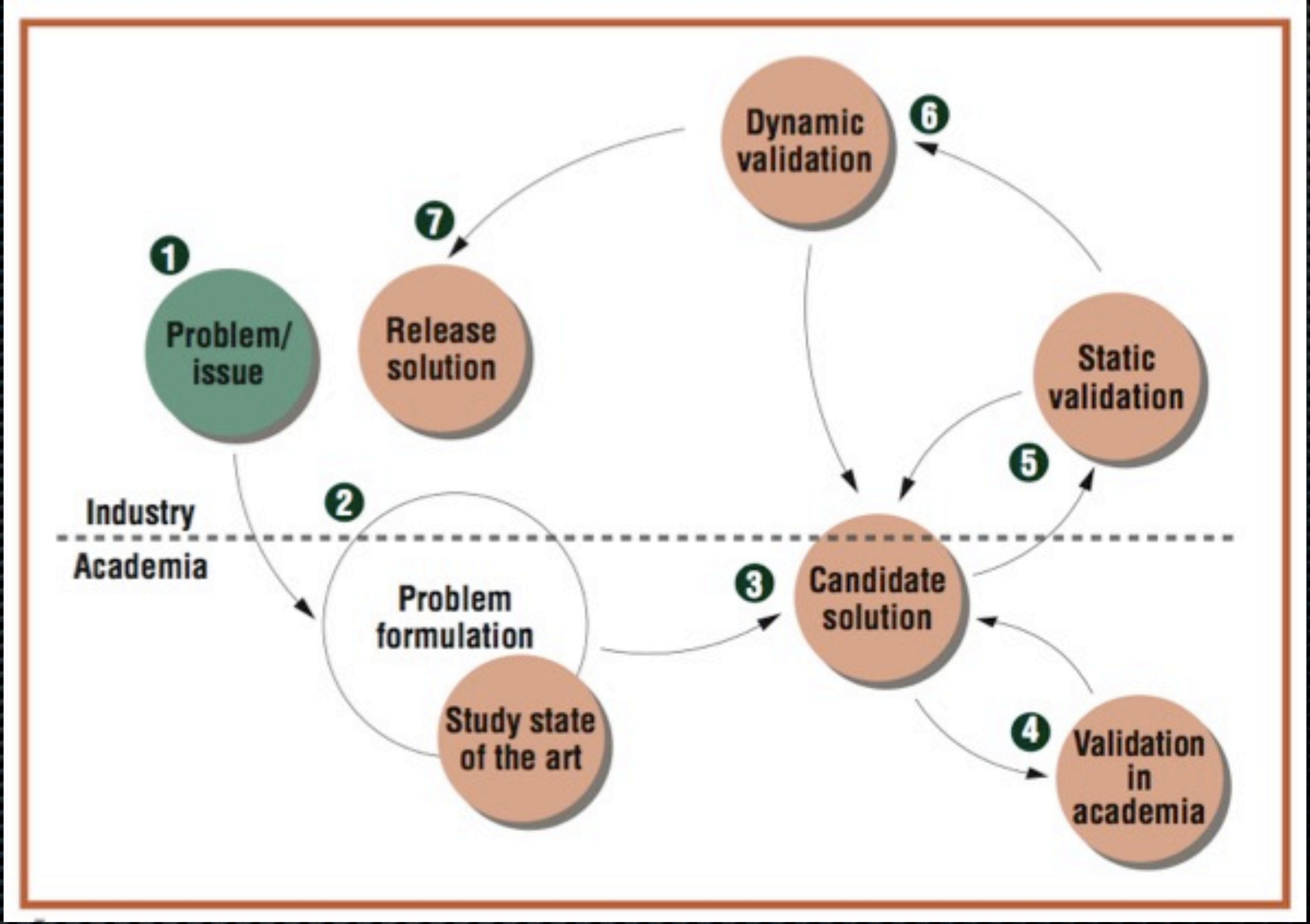
Example Testability Checklist: Observability

- 1. Can you write simple code that easily verifies result of your test execution?
- 2. Can you programmatically detect state changes? Internal state you cannot access?
- 3. In case of multiple options for inputs, can you easily observe which particular option has been exercised?
- 4. Can you capture unexpected errors, warnings and exceptions?
- 5. Can you easily analyze the test execution result to determine pass/fail?

Test-Case Driven Inspection

- Perspective-Based Reading technique for inspections
  - Perspective: Can (high-level) test cases be written?
  - Reader: Test engineer
  - Checks: Testability, Completeness, Conflicts
  - Testers often better at this than Req Engs
- Study compared TCD with Checklist-Based Reading [5]
  - TCD found more major faults, but took longer time
  - Test cases could often be created in parallel

Research <-> Industry/Org



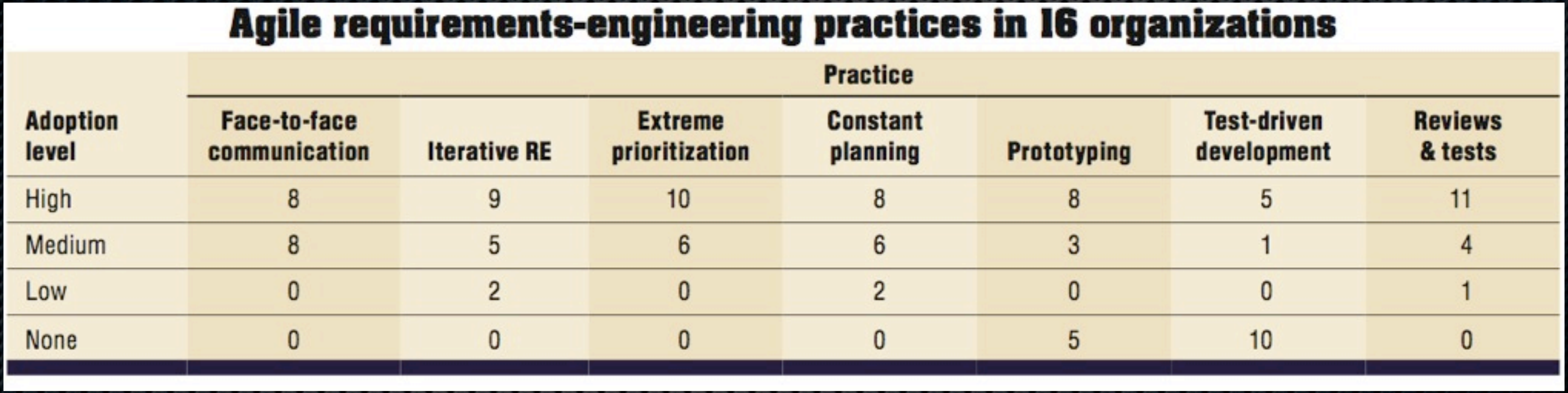
Agile RE practices in industry

- Interviews with 54 practitioners in 16 companies [4]
  - Companies used variants of XP or SCRUM
- Questions:
  - What RE practices do agile developers follow?
  - What benefits and challenges do these practices present?

Agile RE practices in industry

- 7 actual practices found:
  - Face-to-face communication over written specs
    - User stories, no formal docs
    - Saves time
    - Customer steers
    - Lack of trust
    - Customer groups
    - On-site customer
  - Iterative Requirements Engineering
    - High-level first, details in iterations
    - Better customer relation
    - Clearer reqs
    - Minimal docs
    - Cost estimates
    - Nonfunc Reqs
  - Requirements Prioritization goes Extreme
    - Recurrent prioritization
    - Focus: business value
    - Clearer view on reasons
    - Business value to narrow
    - Instability
  - Manage Req change w. constant planning
    - Few & small changes
    - Inappropriate architecture
    - Refactoring not enough
  - Prototyping
    - Quicker customer feedback
    - Customers unrealistic about dev time
  - Test-Driven Development
    - Tests part of RE
    - Tests capture reqs
    - Freedom / experimenting
    - Requires tight customer interaction
    - Devs unwilling
  - Reviews & Acceptance tests
    - Reviews for Req validation
    - QA personnel must help customer
    - Progress report to customer
    - Hard to develop ATs

Agile RE practices in industry



Agile requirements-engineering practices in 16 organizations

Adoption level	Practice						
	Face-to-face communication	Iterative RE	Extreme prioritization	Constant planning	Prototyping	Test-driven development	Reviews & tests
High	8	9	10	8	8	5	11
Medium	8	5	6	6	3	1	4
Low	0	2	0	2	0	0	1
None	0	0	0	0	5	10	0

Papers

[1] Filippo Ricca, Marco Torchiano et al, “Using acceptance tests as a support for clarifying requirements: A series of experiments”, Information and Software Technology, In Press, Corrected Proof, Available online 8 February 2008.

[2] B. George, L. Williams, A structured experiment of test-driven development, Information and Software Technology 46 (5) (2004), pp. 337–342.

[3] E. Maximilien, L. Williams, “Assessing test-driven development at IBM”, Int. Conf. on Software Engineering, IEEE Computer Society Washington, DC, USA, 2003, pp. 564–569.

[4] Lan Cao. B. Ramesh, “Agile Requirements Engineering Practices: An Empirical Study”, IEEE Software, 25 (1), 2008, pp. 60-67.

[5] Dzamashvili-Fogelström, Gorschek, “Test-case Driven versus Checklist-based Inspections of Software Requirements – An Experimental Evaluation”, 10th Workshop on Requirements Engineering (WER’07), Toronto, 2007.