Software Engineering Research in my group(s)

27th of January 2010

ISCAS, Beijing

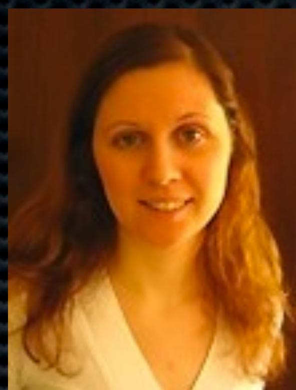
Robert Feldt, Chalmers & Blekinge Inst of Tech, Sweden

Sweden, Chalmers and BTH

Sweden, Chalmers and BTH

- Chalmers University of Technology
  - Top 2 in Sweden, ~10,000 students, ~200/year in CS and SE
  - CS group strong in: Functional Programming, Logic, Security Programming, Telecommunication
- BTH - Blekinge Institute of Technology
  - Top 10 in World in Software Engineering (Prof. Wohlin)
  - ~6,000 students, ~200/year in CS and SE
  - Strong in: Empirical SE, Industry-collaboration

My PhD students



TBD

Robustness

Cost Estimation

Auto System Test Agile

Chalmers, Göteborg

SW Customization

Search-based

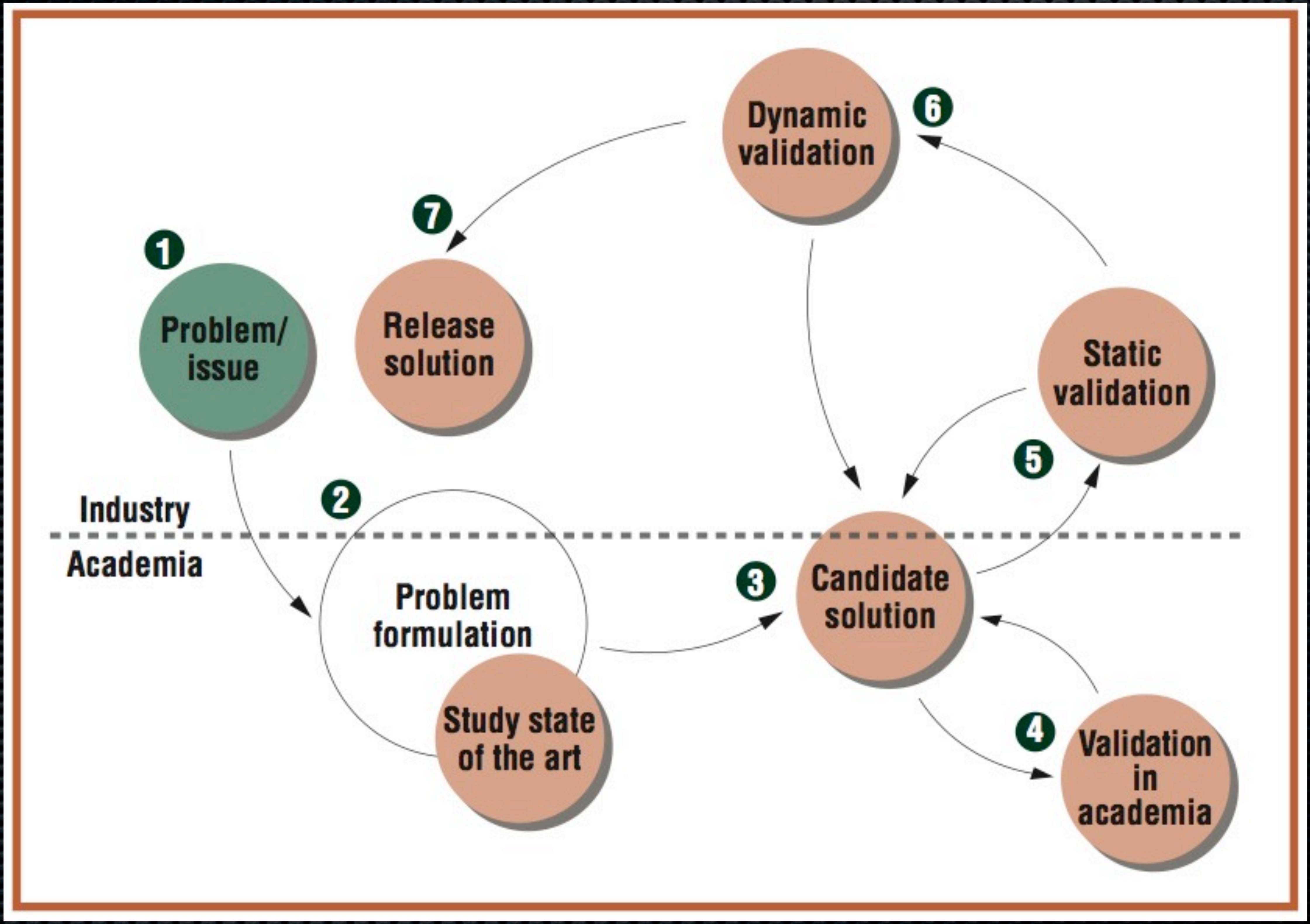
Req<->Test

BTH, Ronneby

Our Scientific Approach

- Empirical - data from real world/experiments
- Statistical - design & analyze experiments/data rigorously
- Broad -
  - cover breadth of SE, many disciplines, not only tech
  - no predetermined solutions in company collaborations
  - breadth of research methods, qualitative <-> quantitative
- Engineering - theory can support but ultimately SE is engineering
- Theory - but we need some...

How we often work

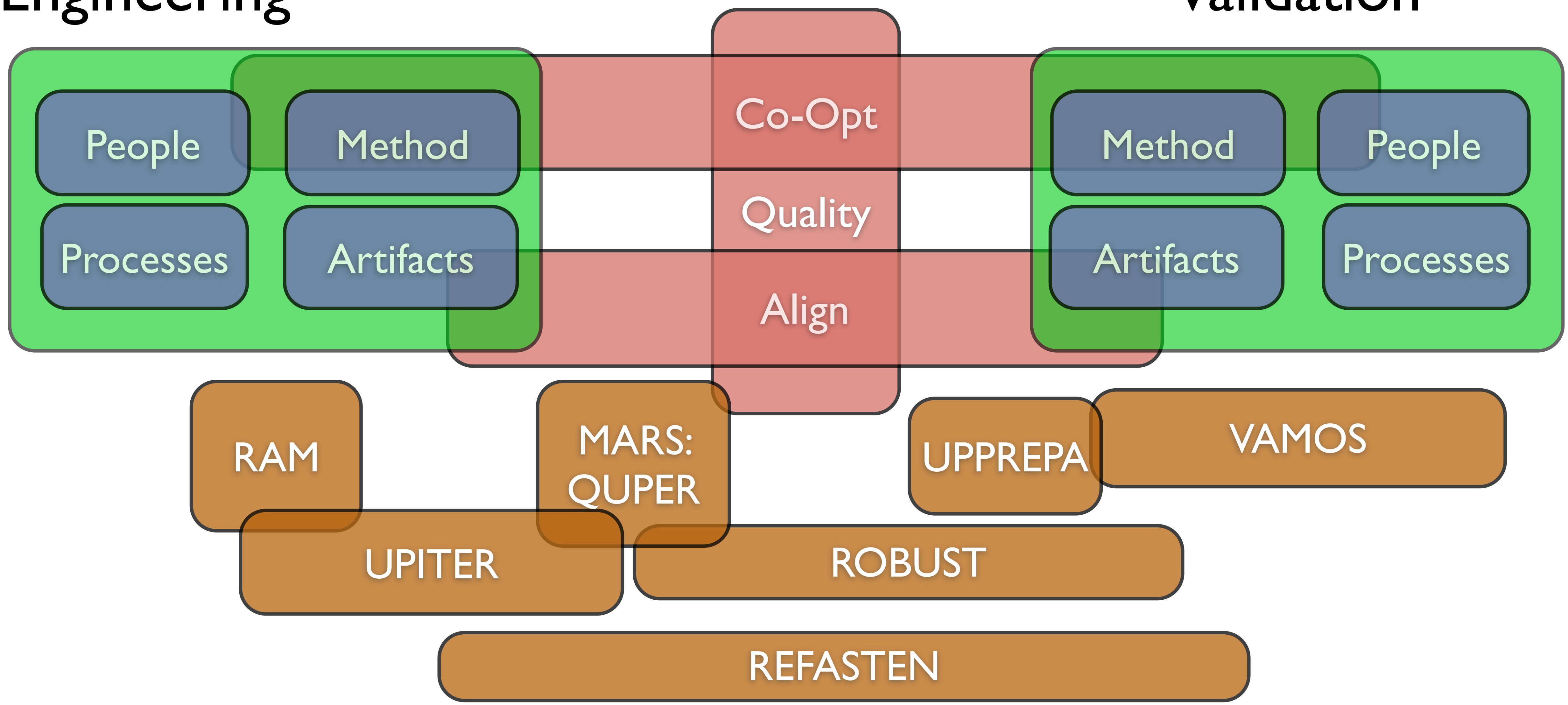


Company Collaborations

- RUAG Aerospace Sweden - Optimizing V&V, Standards, Cost models
- Swedish Space Corporation - Optimizing V&V
- Ericsson (Karlskrona) - SW Customizations
- ABB, Sony Ericsson, Softhouse - Aligning Req & Test Activities *
- Volvo Technology - Robustness Req & Testing
- Wireless Car & Ericsson (Gothenburg) - Robustness
- SAAB Security ATM & Systems - Agile testing, Test Creation f. Legacy Code *
- ST Ericsson - Data Mining V&V Metrics Data *
- Volvo Car Corp - Interface SW Development <-> Manufacturing

Requirements Engineering

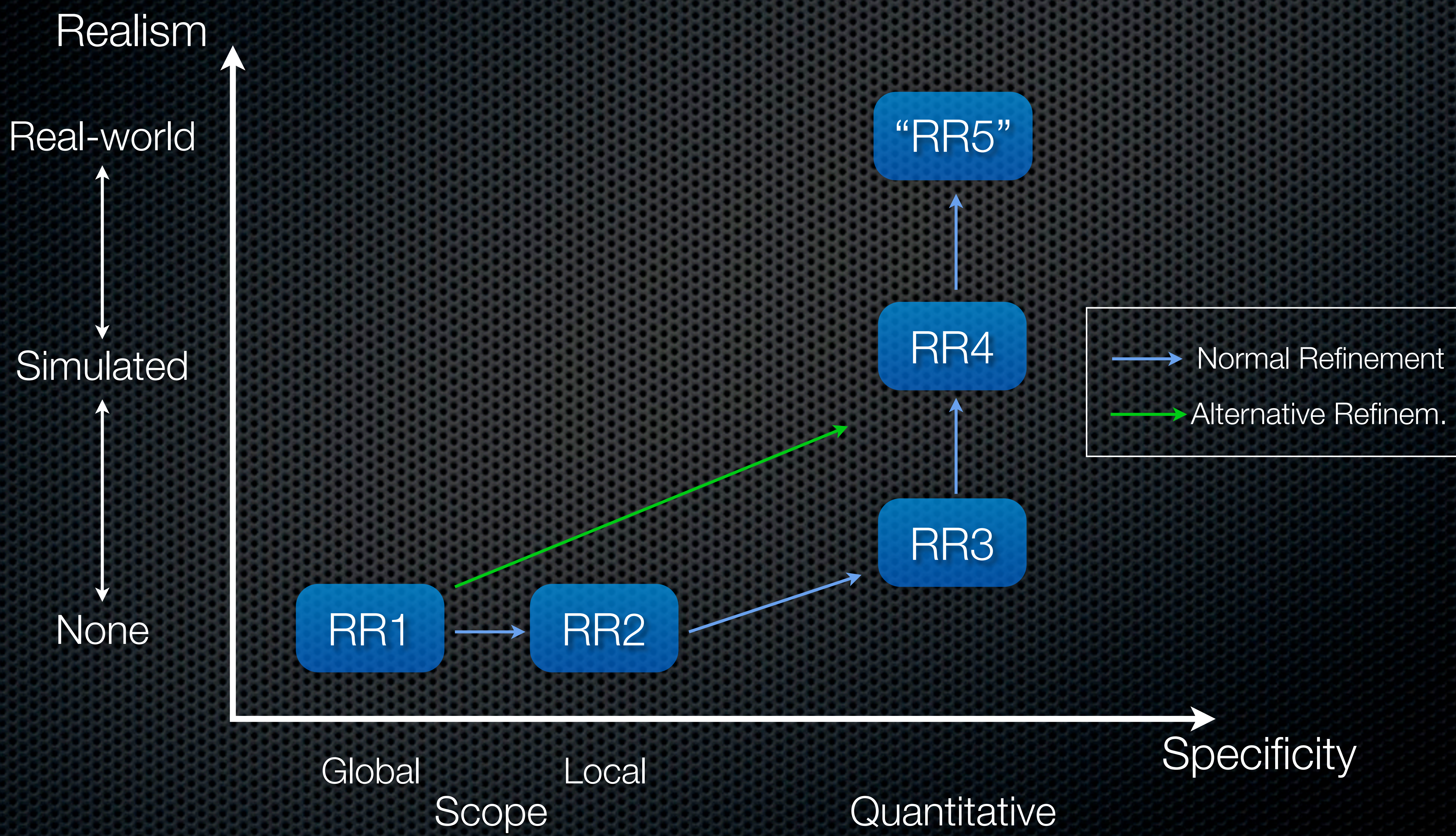
Verification & Validation



ROAST Overview

- Levels of Robustness (LoR)
  - Robustness Requirements refined from level 1 to 5
- Robustness (Specification) Patterns
  - Pattern template similar to Design Patterns
- Testing methods aligned with each Pattern/Level
  - Different level of Verification for different LoR’s - Checklists/ Reviews, Test methods, ...

Refining Robustness Reqs



Robustness Patterns

- Template similar to “Design Patterns” but adapted:
  - Name, Robustness Area, Intent, Motivation, Constraints, Applicability, Participants, Scope, Factors, Measures, Verification
- Different Robustness Areas:
  - Input validation, Exception/Failure handling, Service degradation & Resource Mngmnt, (Availability/Reliability/Security/Dependability)
- One pattern can specify several levels
  - Scope gives localization examples (for LoR1 -> LoR2)
  - Measures gives quantification examples (for LoR2 -> LoR3)
  - Factors list the robustness factors (for LoR3 -> LoR4&5)

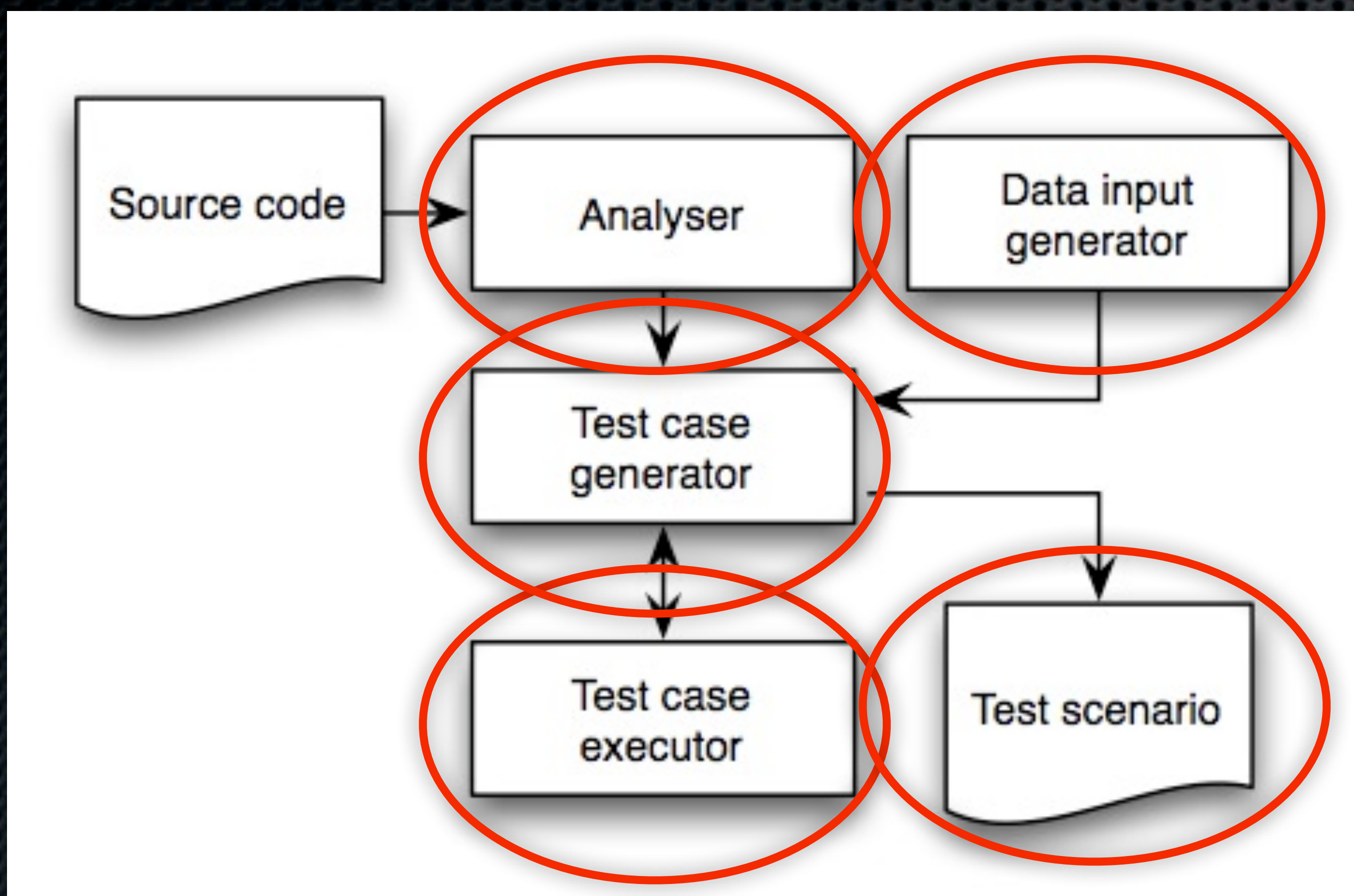
SBST for Complex Test Data & DynLang

New

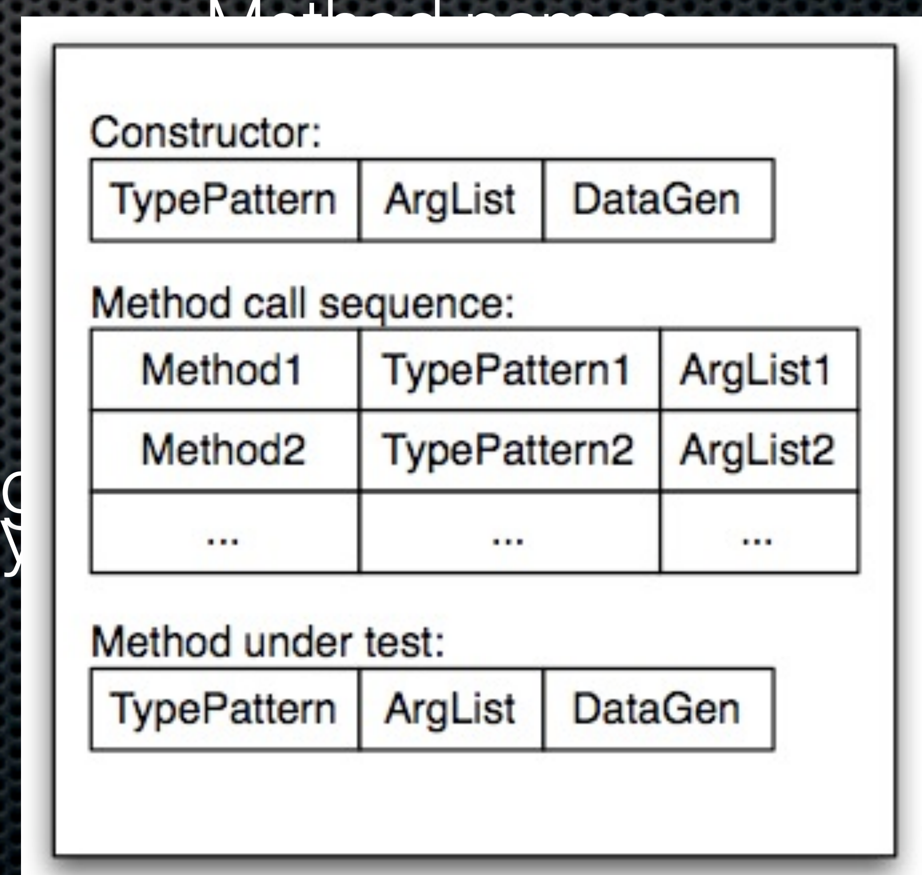
- SB SW Testing for Code Coverage: Well researched, but:
  - Mostly simple test data (Numbers)
  - Statically typed languages
- This project:
  - Complex data types
  - Dynamic programming language (Ruby)

New

RUTEG = RUby Test Case Generator



- Static code analysis
- Goal: Reduce search space
- Problem-specific generators
- Returns info on:
- Simple OO design



GA with individuals:

Runs test case and collects coverag

Individuals can be dumped as Ruby

New

Argument Type Selection

- Fitness of *type* for *arg*: $$f_{type} = \begin{cases} 1 - \left(\frac{|(M_{arg} - M_{type})|}{|M_{arg}|}\right) & \text{if } |M_{arg}| > 0 \\ 1 & \text{otherwise} \end{cases}$$
  - For fitness-proportionate type selection
- Not enough since not independent between arguments:

```
def add(a, b)
  a+b
end
```

(Fixnum, Fixnum) or (String, String) ok!

(String, Fixnum) or (Fixnum, String) not!

For each method application:

Applicable

Suspicious

Discarded

Experiment: Results

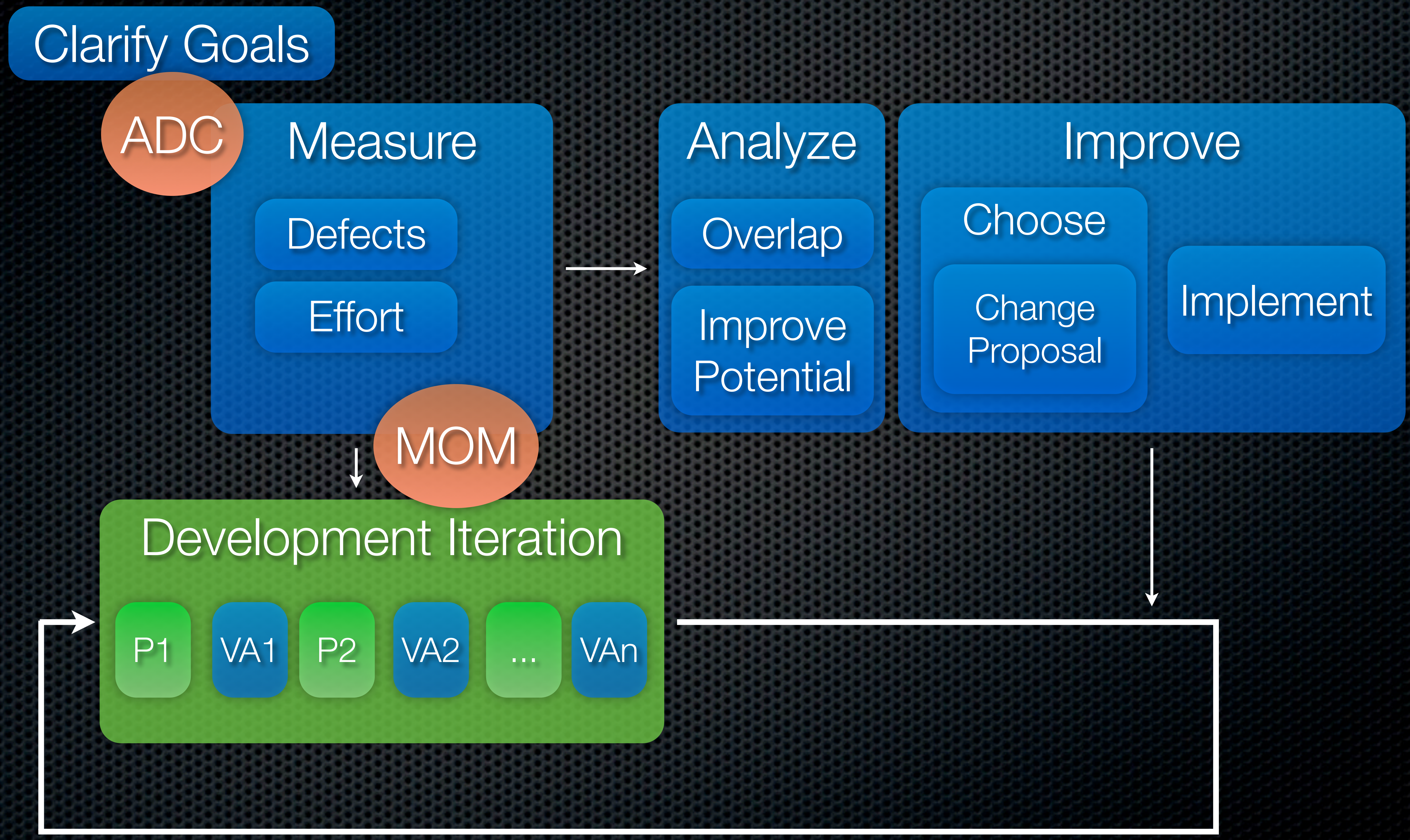
New

Table II. Average code coverage achieved by RuTeG and Random Testing (RT), with *t*-test where * indicates $p < 0.05$ and ** indicates $p < 0.01$; and the time to maximum coverage expressed in seconds.

Methods	**Cov. RuTeG**	**Cov. RT**	***t*-test**	**Time RuTeG**	**Time RT**
triangle_type	100%	81%	**	59	99
valid_isbn10?	100%	100%		29	84
valid_isbn13?	100%	100%		34	80
add_address	100%	100%		56	97
rb_insert	100%	88%	**	68	92
bootstrapping	100%	86%	*	54	88
gamma	98%	92%	**	209	213
bfs	100%	93%	*	79	86
dfs	100%	96%	*	70	72
warshall_floyd_shortest_paths	100%	100%		155	196
rank	100%	92%	*	111	202
** (power!)	100%	96%	**	274	356
canBlockACheck	94%	74%	**	285	333
move	88%	68%	**	356	143

Optimizing Space SW Verification&Validation

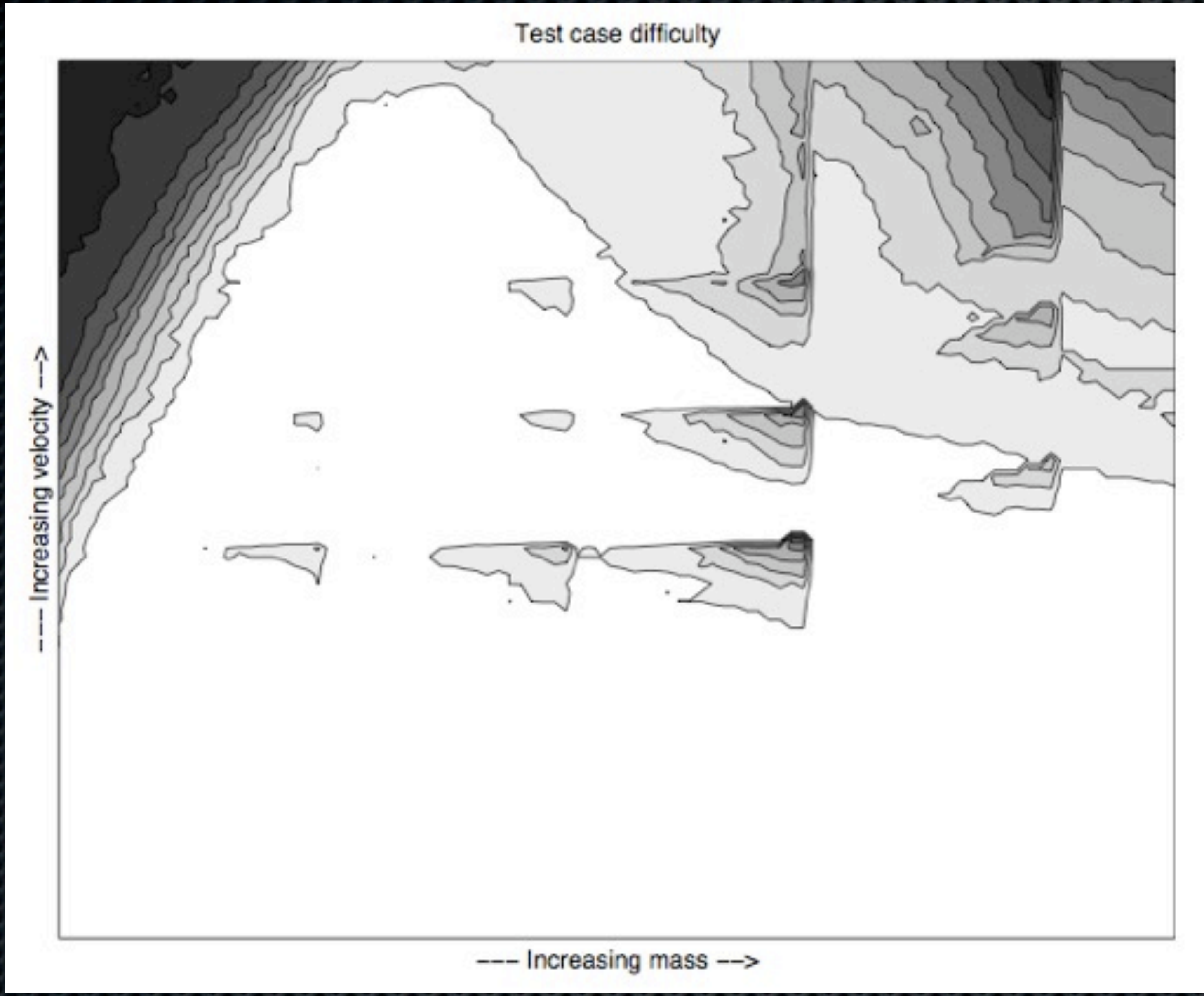
VAMOS



Clarify Goals
ADC
Measure
Defects
Effort
MOM
Analyze
Overlap
Improve Potential
Improve
Choose
Change Proposal
Implement
Development Iteration
P1
VA1
P2
VA2
...
VAn

Early Software Difficulty Visualisation

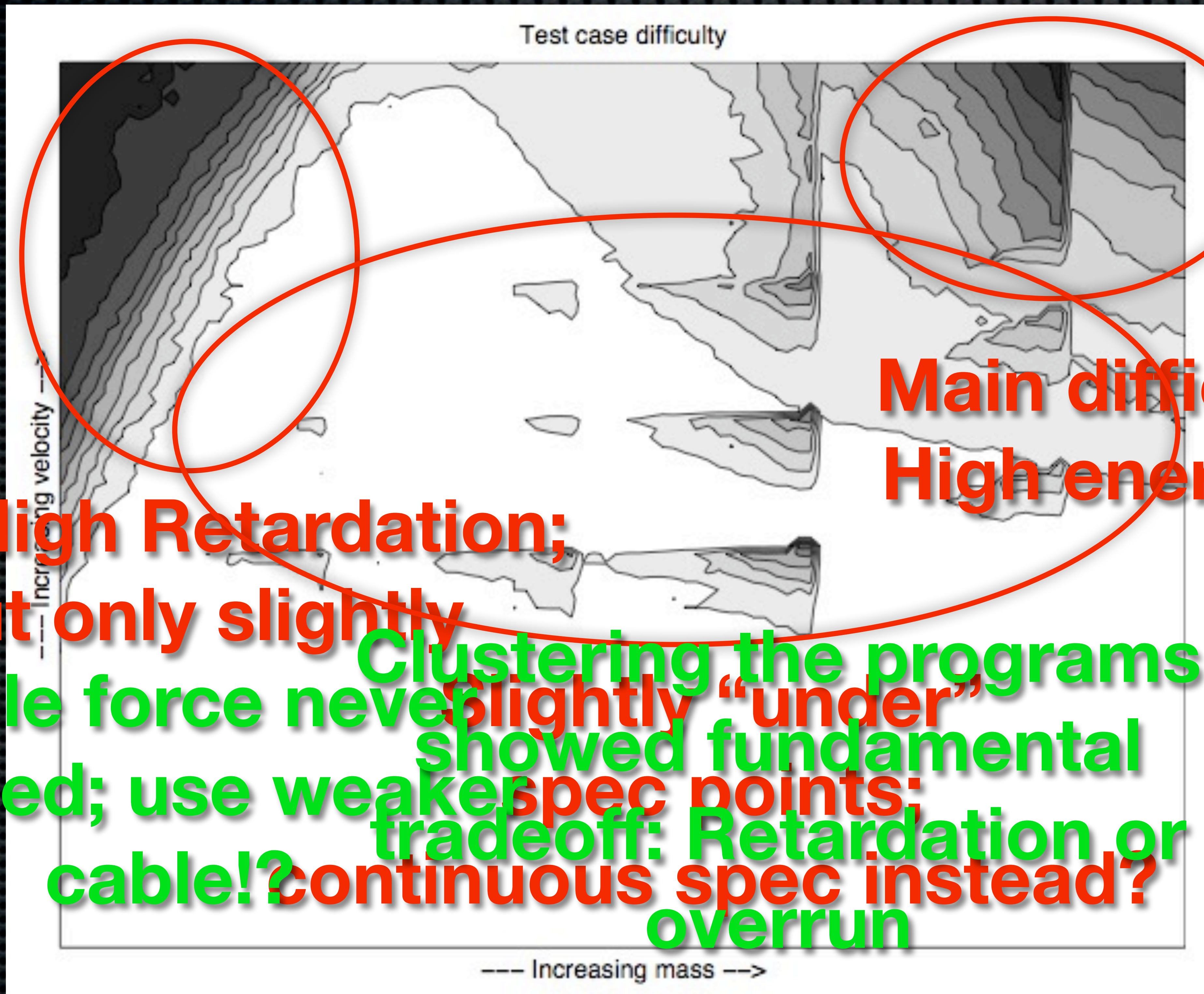
Old



- Aircraft braking system
- Genetic Programming of Control software
- Create many programs, diagram of where they fail
- Help Engineers visualize problems early!

Analysis of Failures of GP programs

Old



Main difficulty: High energies!

Too High Retardation; but only slightly

Slightly “under” spec points; continuous spec instead?

Cable force never violated; use weaker cable!?

Clustering the programs showed fundamental tradeoff: Retardation or overrun

Many Limitations

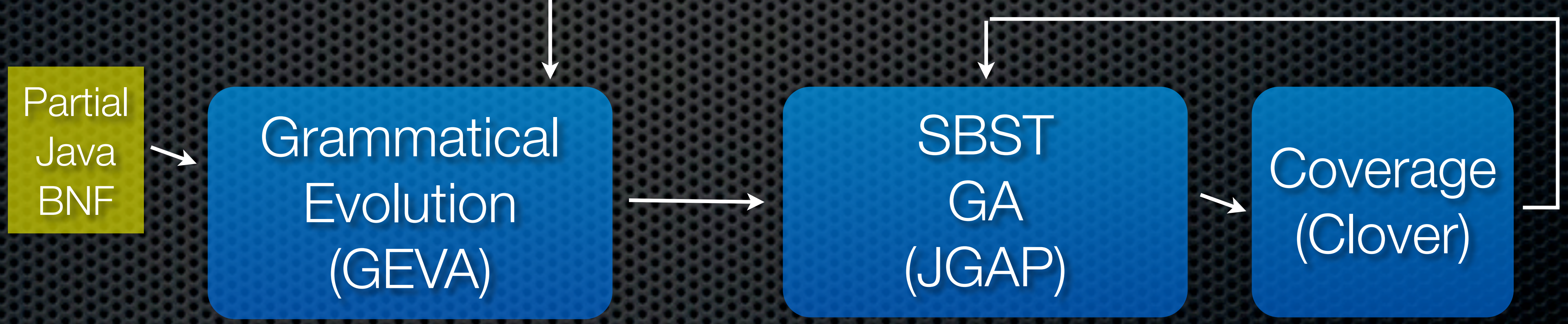
Old

- Small target application
- Few requirements
- Low-dimensional input space
- Existing simulator; typically not available in early phases
- Fundamental assumption: SB AutoProgramming fail in similar ways to human programmers
  - What is your experience?
  - Does it really need to?

Factorial Experiment on SBST Scalability

Hot

	LOC	CC	...
Run 1	200	5	
Run 2	50	10	
...	...	...	



Compare to RT

SWELL =
Swedish VV ExceLLence

National Research School in
Software Verification & Validation

swell.se

More information:

http://www.cse.chalmers.se/~feldt