


Faults & Failures in Novice GOAL Programs

Michael Winikoff

University of Otago, New Zealand

"Novice Programmers' Faults & Failures in GOAL Programs:
Empirical Observations and Lessons"





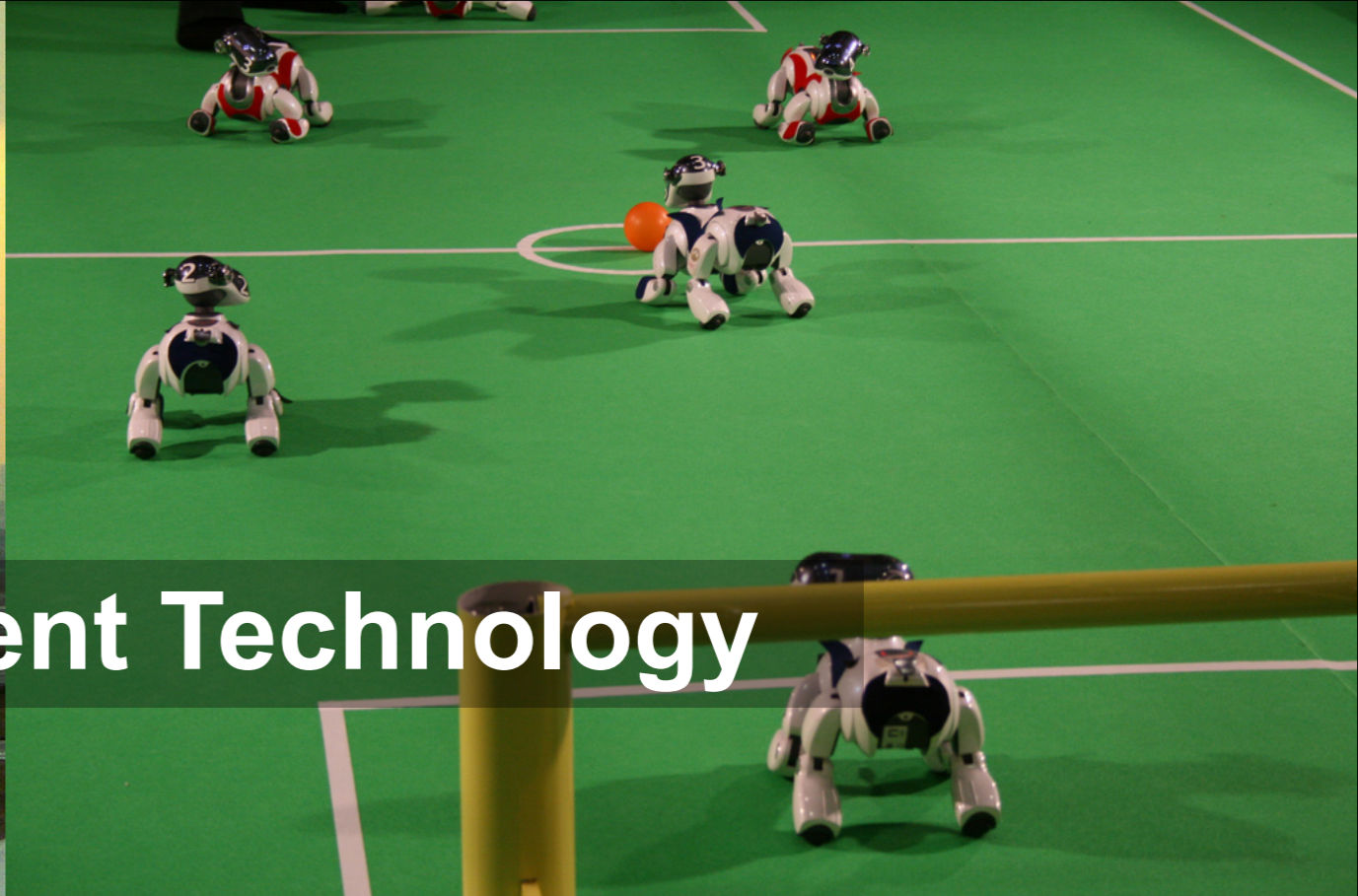
“In contrast to remote control, this sophisticated set of computer programs acts as an agent of the operations team on board the remote spacecraft. Rather than have humans do the detailed planning necessary to carry out desired tasks, remote agent will formulate its own plans, using high level goals provided by the operations team. Remote agent devises its plan by combining those goals with its detailed knowledge of both the condition of the spacecraft and how to control it.” <http://nmp.nasa.gov/ds1/tech/autora.html>

Deep Space 1 is an *agent*

- **Situated** (“placed on board”)
- **Autonomous** (“In contrast to remote control”)
- **Proactive** (“... formulate its own plans, using high level goals”)
- **Reactive** (“If problems develop, remote agent in many cases will be able to fix them or work around them. ...”)
- **Social** (“... If it cannot, it can request help from its sentient terrestrial collaborators”)

Also Multi-agent systems (MAS) and agent societies.

"Human-inspired computing"



Some Applications of Agent Technology



Applicable where there is natural distribution, and where resilience and flexibility are required

Human-inspired PLs

- Based on models of human decision making and planning
- Belief Desire Intention (BDI) model influential
- **Plans and Goals**
- Need to provide means for persistently achieving goals while responding to changes
- Examples: JACK, Jadex, Jason, 3APL, 2APL, Brahms, GOAL ...

Research Question

What are the types of **faults** that *novice* programmers create when using Agent-Oriented Programming Languages (AOPLs) and how do they manifest as **failures**?

- **Fault**: mistake in the program
- **Failure**: run-time manifestation of an error

Motivation

- Potential implications to language design, tool design, teaching programming
- Novices? If agent programming is to take off, need to teach AOPL to lots of agent-novices ...

Contributions

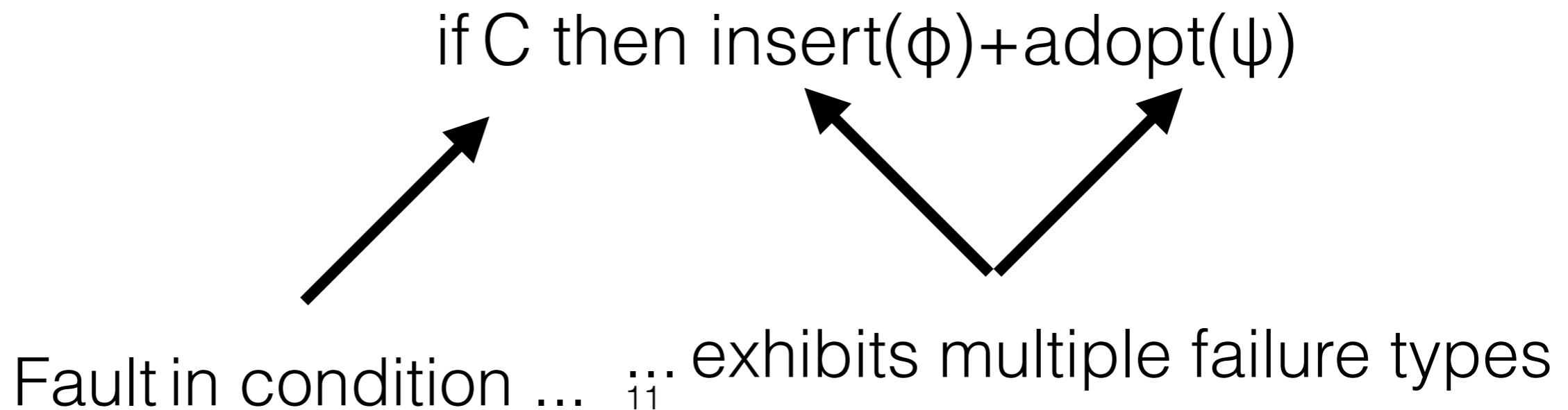
- Taxonomies for faults and for failures
- Empirical data on fault and failure occurrences
- Implications for debugging tools, language design, and teaching

Taxonomies

- Bottom-up vs. Top-down?
- Principles for systematic derivation:
 - Fault: (syntactic) language features
 - Failure: language semantics

Taxonomies

- Bottom-up vs. Top-down?
- Principles for systematic derivation:
 - Fault: (syntactic) language features
 - Failure: language semantics
- Fault and Failure locations expected to correlate, but ... e.g. incorrect action selected due to error in domain knowledge



Fault Taxonomy

- Consider rules: whole rule, condition, action, order
- Consider other parts of program: domain knowledge, initial beliefs/goals, action definition
- Typos and other error
- Augment with observations ...

- **.goal = domain knowledge (Prolog rules) + initial beliefs + initial goals + action definitions (pre/post conditions) + rules ...**
- **Rules: percept and main module**
- **Rule: if condition then action(s) [also forall-do]**

Example GOAL Rule

if bel(in(Room), color(Block,Color)),
not(goal(deliver(ABlock))) **then**
adopt(deliver(Block))

Failure Taxonomy

- **P**ercepts, **A**ctions, **G**oals

Can ...

- Fail to do what should be done
- Do what shouldn't be done

1. Clear percepts

**2. Update percepts
(execute event module)**

**3. Select rule in main
module and execute it**

4. Drop believed goals

Taxonomies

Fault Taxonomy

- a: missing rule
- b: additional (wrong) rule
- c: condition on rule wrong - specific variants:
 - cs: condition too strong (EO)
 - cw: condition too weak (EO)
- d: action(s) of rule wrong (but legal)
- e: rule includes two user-defined actions
- f: rule order wrong (EO)
- g: action definition wrong
- i: using “if then” instead of “forall do” (EO)
- j: missing action in a rule (special case of “d”, EO)
- k: fault in domain knowledge
- n: fault in initial beliefs/goals
- t: typo (e.g. `atblock` instead of `atBlock`)
- o: other fault not classified above

Failure Taxonomy

- P1: failure to deal with percept
- P2: other incorrect percept processing
- G1: failure to add goal that should be added
- G2: failure to drop goal that should be dropped
- G3: adding a goal that shouldn't be added
- G4: incorrectly adding a second goal of the same type (special case of G3, EO)
- G5: dropping a goal that shouldn't be dropped
- A1: selecting wrong (user-defined) action
- A2: beliefs not updated correctly when action performed
- A3: action selected when should be doing nothing (waiting for environment, EO)
- A4: action interface mismatch (EO)
- O: other failure not classified above

Figure 2: Fault Taxonomy (left) and Failure Taxonomy (right). “EO” denotes Empirical Observations.

Taxonomies

Fault Taxonomy

a:	missing rule
b:	additional (wrong) rule
c:	condition on rule wrong - specific variants: cs: condition too strong (EO) cw: condition too weak (EO)
d:	action(s) of rule wrong (but legal)
e:	rule includes two user-defined actions
f:	rule order wrong (EO)
g:	action definition wrong
i:	using “if then” instead of “forall do” (EO)
j:	missing action in a rule (special case of “d”, EO)
k:	fault in domain knowledge
n:	fault in initial beliefs/goals
t:	typo (e.g. atblock instead of atBlock)
o:	other fault not classified above

Failu

P1:

P2:

G1:

G2:

G3:

G4:

G5:

A1:

A2:

A3:

A4:

O:

Taxonomies

Failure Taxonomy

P1: failure to deal with percept

P2: other incorrect percept processing

G1: failure to add goal that should be added

G2: failure to drop goal that should be dropped

G3: adding a goal that shouldn't be added

G4: incorrectly adding a second goal of the same type
(special case of G3, EO)

G5: dropping a goal that shouldn't be dropped

A1: selecting wrong (user-defined) action

A2: beliefs not updated correctly when action performed

A3: action selected when should be doing nothing
(waiting for environment, EO)

A4: action interface mismatch (EO)

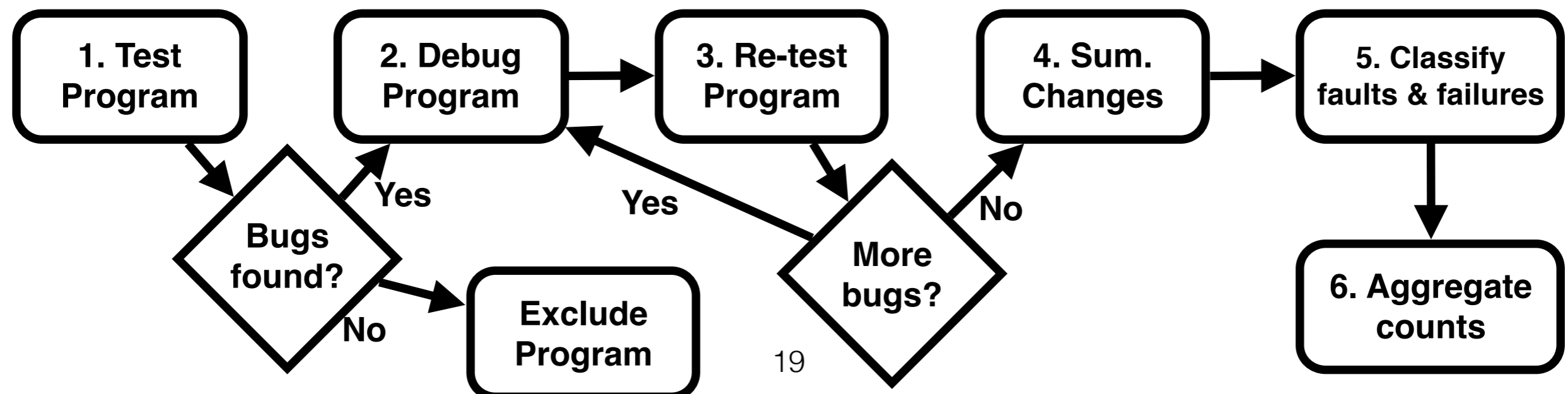
O: other failure not classified above

Contributions

- Taxonomies for faults and for failures
- Empirical data on fault and failure occurrences
- Implications for debugging tools, language design, and teaching

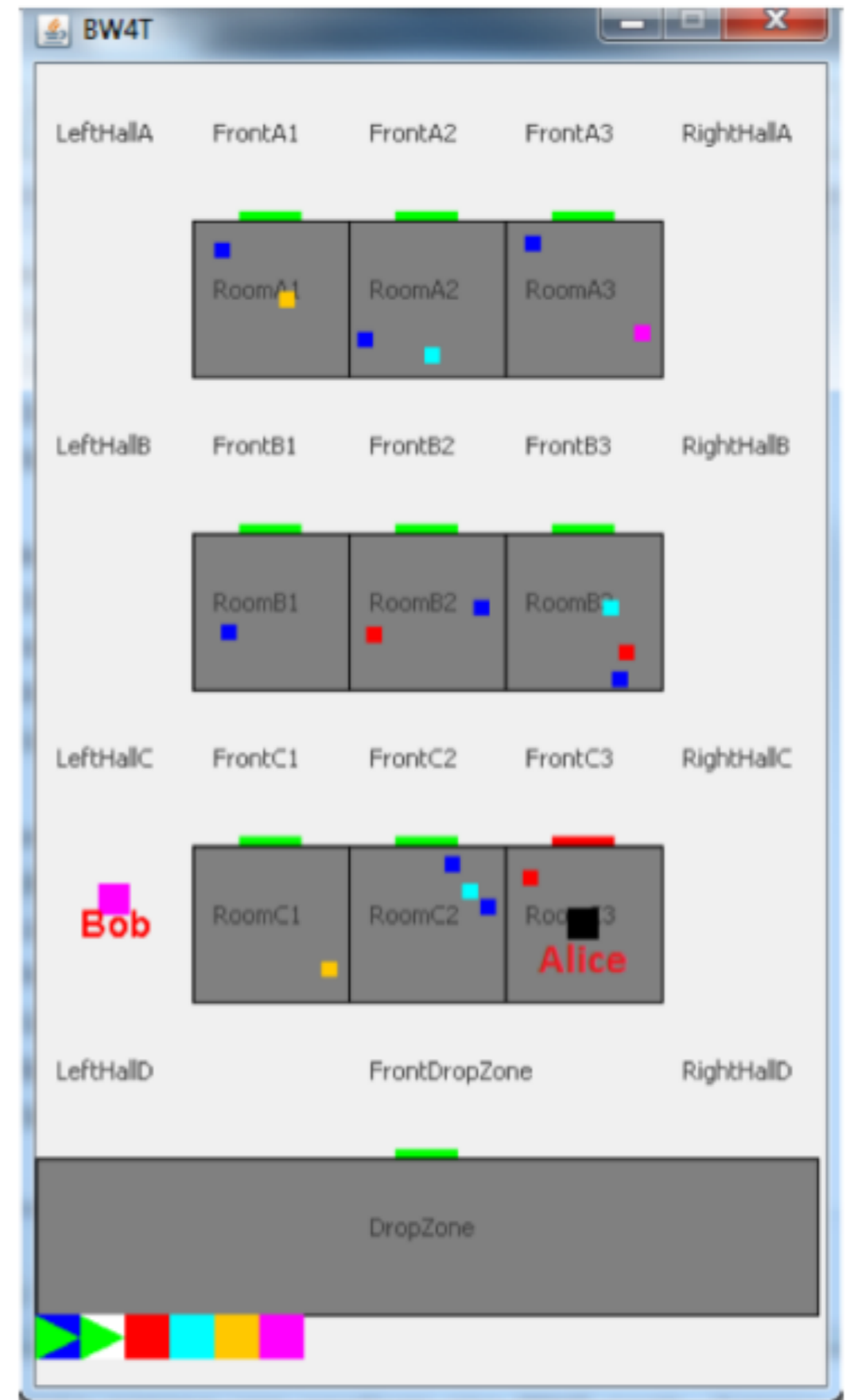
Methodology

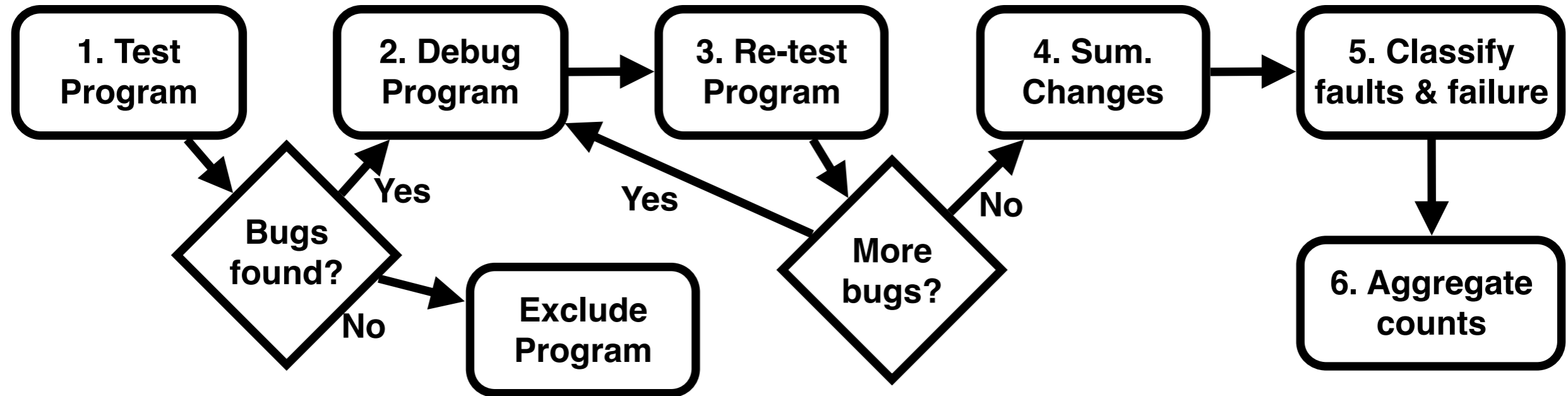
- Obtained 55 student-written assignments (single agent BW4T), but 4 didn't run so excluded
- Students were provided with skeletal program (one action definition, 2 rules to explore, some percept processing rules)
- Programs ranged from 172 to 378 lines (mean 225.5, median 220)



BW4T

- Blocks World for Teams
- Aim: deliver coloured blocks in desired order
- Actions: `goTo(Locn)`, `goToBlock(BlockID)`, `pickUp`, `putDown`
- Percepts: `in(Room)`, `color(BlockID, Color)`, `holding(BlockID)`, ...





1: tested with >7300 tests; 41 buggy, 10 bug-free

2: In debugging, considered alternatives ...

Excluded 5 very buggy programs (>10 changes)

5: "What failure ... if this was the only fault?"

6: Counted how many programs had >0 occurrences

Example

- **Behaviour:** goTo(RoomA1), goToBlock(44), pickUp, goTo(DropZone), putDown, goTo(RoomA1), goToBlock(44), goToBlock(44), ...
- **Culprit:** if bel(in(Room), nextColorInSeq(Color), color(Block,Color), not(holding(_)), pos(Block,Room)) then adopt(atBlock(Block))
- **Fix:** add not(gone(Block))
- **Classification:** Error: too weak condition (cw), Fault: adding goal incorrectly (G3)

Results: Faults

Fault	Count	Fault	Count
rule order (f)	19	other (o)	4
weak cond (cw)	15	extra rule (b)	3
2 user actions (e)	9	wrong action (d)	2
miss act (j)	8	ifthen/forall (i)	2
strong cond (cs)	7	typo (t)	2
actiondef (g)	6	domain (k)	(1)
misrule (a)	5	initial bel/goal (n)	0
othercond (c)	5		

Results: Faults

Fault	Count	Fault	Count
rule order (f)	19	other (o)	3
weak cond (cw)	15	extra rule (b)	3
2 user actions (e)	9	wrong action (d)	2
miss act (j)	8	ifthen/forall (i)	2
strong cond (cs)	7	typo (t)	2
actiondef (g)	6	domain (k)	0(*)
misrule (a)	5	initial bel/goal (n)	0
othercond (c)	5		

Almost all fault types seen ... but only 8 in >10% and only 4 in >20% of programs

Results: Faults

Fault	Count	Fault	Count
rule order (f)	19	other (o)	3
weak cond (cw)	15	extra rule (b)	3
2 user actions (e)	9	wrong action (d)	2
miss act (j)	8	ifthen/forall (i)	2
strong cond (cs)	7	typo (t)	2
actiondef (g)	6	domain (k)	0(*)
misrule (a)	5	initial bel/goal (n)	0
othercond (c)	5		

Most common issue: rule order

Results: Faults

Fault	Count	Fault	Count
rule order (f)	19	other (o)	3
weak cond (cw)	15	extra rule (b)	3
2 user actions (e)	9	wrong action (d)	2
miss act (j)	8	ifthen/forall (i)	2
strong cond (cs)	7	typo (t)	2
actiondef (g)	6	domain (k)	0(*)
missrule (a)	5	initial bel/goal (n)	0
othercond (c)	5		

Condition faults common - if merge cw and cs, then 20 programs

Results: Faults

Fault	Count	Fault	Count
rule order (f)	19	other (o)	3
weak cond (cw)	15	extra rule (b)	3
2 user actions (e)	9	wrong action (d)	2
miss act (j)	8	ifthen/forall (i)	2
strong cond (cs)	7	typo (t)	2
actiondef (g)	6	domain (k)	0(*)
misrule (a)	5	initial bel/goal (n)	0
othercond (c)	5		

Illegal(*) GOAL usage (two user defined actions) quite common! (e)

(*) This is no longer illegal: the user is responsible for ensuring that two actions can be done simultaneously

Results: Faults

Fault	Count	Fault	Count
rule order (f)	19	other (o)	3
weak cond (cw)	15	extra rule (b)	3
2 user actions (e)	9	wrong action (d)	2
miss act (j)	8	ifthen/forall (i)	2
strong cond (cs)	7	typo (t)	2
actiondef (g)	6	domain (k)	0(*)
misrule (a)	5	initial bel/goal (n)	0

Most faults observed relate to rules (g, n, k exceptions)
action def issues relate to async environment ...

Results: Faults

Fault	Count	Fault	Count
rule order (f)	19	other (o)	3
weak cond (cw)	15	extra rule (b)	3
2 user actions (e)	9	wrong action (d)	2
miss act (j)	8	ifthen/forall (i)	2
strong cond (cs)	7	typo (t)	2
actiondef (g)	6	domain (k)	0(*)
misrule (a)	5	initial bel/goal (n)	0
othercond (c)	5		

Typos (t) rare, only a few "other" (o)

Results: Failures

Failure	Count
P1: fail to deal with percept	12
P2: other percept	10
G1: fail to add goal	5
G2: fail to drop goal	4
G3: adding a goal wrongly	7
G4: add duplicate goal	7
G5: dropping goal wrongly	1
A1: wrong action	29
A2: incorrect belief update	10
A3: should've done nothing	2
A4: action-interface mismatch	1
O: Other	0

- 17 of 36 programs had a **P**ercept processing error (47%)
- 17 had **G**oal error
- 31 (86%) had **A**ction error

Implications

- **Language design:** (1) Percept processing is a source of faults - find simpler way to specify percept processing? (2) Extending GOAL to allow multiple sequential user-defined actions ...
- **Teaching:** Don't use explicit drop(goal); use conditions so single rule applicable
- **Tool design:** condition checking, debugging percept processing

Validity

- Internal: only one problem (BW4T), only single agent, looked at final submission (easier bugs already removed - but this is good)
- External: GOAL only ...

Future Work

- More programs, and not just BW4T
- Different AOPLs
- Applying lessons learned

Conclusions

- Derived taxonomies for faults and failures
- Empirical investigation of occurrences
- Implications to language design, tool design, teaching
- *Thanks to Sharmila, and Delft colleagues (Koen and Maaïke)!*