Specification-Based Testing 2

Stuart Anderson

Overview

- We consider issues in the generation of test cases in particular defining coverage criteria that reduce the combinatorial complexity of test case generation.
- We then go on to consider model-based black-box testing where we have some model of the system and use that to decide how to exercise the system. Typical examples of models include:
  - Decision trees/graphs
  - Workflows
  - Finite State Machines
  - Grammars
- All of these models provide some kind of abstraction of the systems behaviour we can use this both to explore the systems behaviour and check that it agrees with the abstraction.

 ©2011-14

Reducing the number of test cases

Display Mode	Language	Fonts
full-graphics	English	Minimal
text-only	French	Standard
limited-bandwidth	Spanish	Document-loaded
	Portuguese	

Color	Screen size
Monochrome	Hand-held
Color-map	Laptop
16-bit	Full-size
True-color	

[P&Y p.190: Table 11.3]

Coverage Criterion

- If our tests just took a simple approach to exhaustive testing inputs drawn from Display Mode, Fonts, and Screen Size we would need to consider 27 test cases.
- With large numbers of categories this becomes prohibitive (e.g. n categories each of size k has $k \times n$ possible cases).
- We can reduce this by just requiring that the input set cover all possible m-tuples of each subset of m variables drawn from n.
- For example, in the case above we might require that we just ensure all pairs of (Display Mode, Fonts), (Fonts, Screen Size) and (Display Mode, Screen Size) are covered in the test set.
  This reduces the test set from 27 combinations to 9.

 ©2011-14

Ensuring all Pairs are Covered

Display mode × *Screen size*		*Fonts*
Full-graphics	Hand-held	Minimal
Full-graphics	Laptop	Standard
Full-graphics	Full-size	Document-loaded
Text-only	Hand-held	Standard
Text-only	Laptop	Document-loaded
Text-only	Full-size	Minimal
Limited-bandwidth	Hand-held	Document-loaded
Limited-bandwidth	Laptop	Minimal
Limited-bandwidth	Full-size	Standard

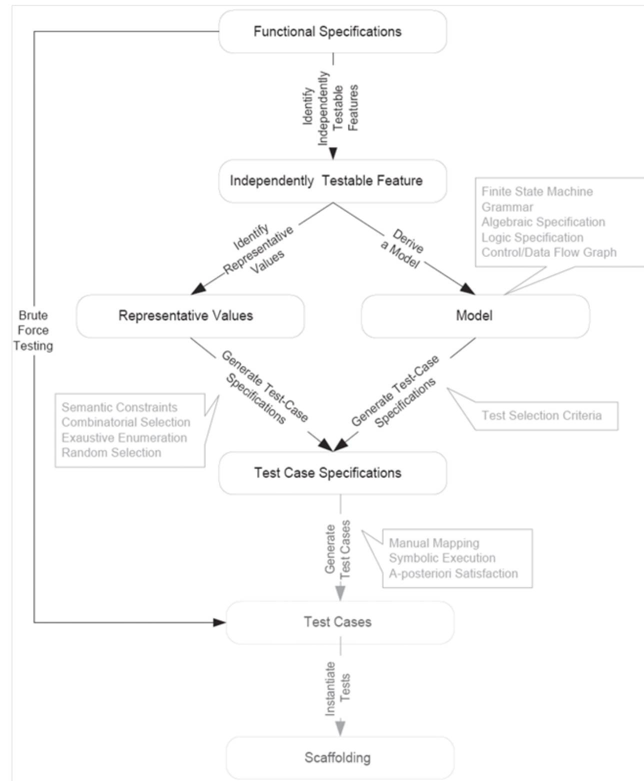
[P&Y p.191: Table 11.4]

 ©2011-14

Remarks

- Generally enumerating all possible combinations is exhaustive but probably infeasible given cost constraints.
- Alternative is to choose some systematic way of reducing the space.
- In this case we chose to find all pairs.
- Other criteria are possible (see the reading).

Model-based Testing



[P&Y p.169: Figure 10.3]

Models

- Models typically provide some abstract representation of the behaviour of the system.
- Typical notations are:
  - Algebraic Specifications
  - Control/Data Flow Graphs
  - Logic-based specification
  - Finite State Machine Specification
  - Grammar-based Specification

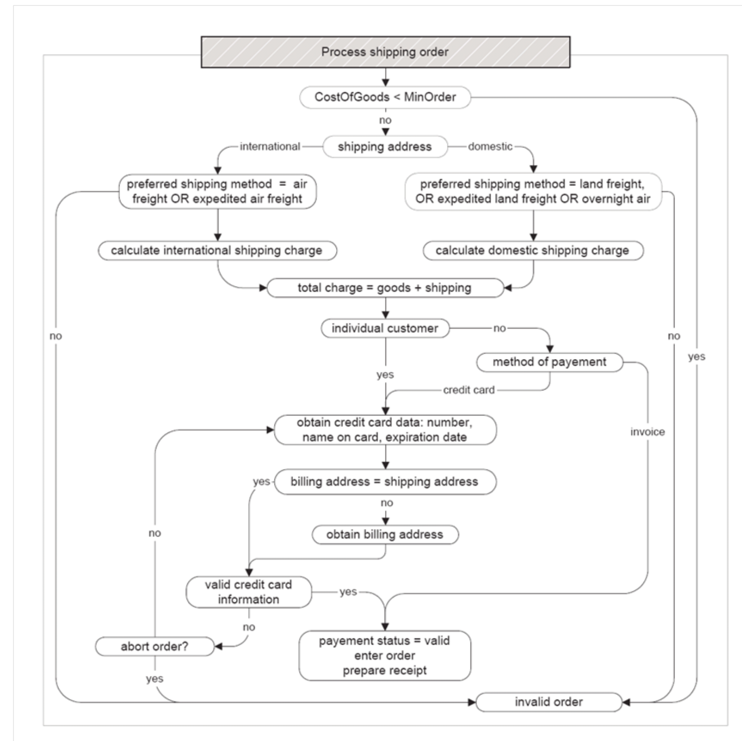
 ©2011-14

Control Flow Graphs

- Often specify the human process the system is intended to support (e.g. UML Activity Diagrams).
- Can be used to represent both *'normal'* and *'erroneous'* behaviours (and recovery behaviour).
- Abstract away from internal representations.
- Focus on interactions with the system

 ©2011-14

Shipping Order Process



[P&Y p.259: Figure 14.7]

Different Adequacy Criteria Are Applicable

- **Node coverage** – ensure that test cases cover all the nodes in the flow graph.
- **Branch coverage** – ensure we branch in both directions at each decision node.
- **Mutations** – we might also consider introducing mutations where the user does not follow the control graph:
  - can provide explanations of *"automation surprises"* (see Rushby paper in readings).
  - Machines are often better at remembering state than humans

 ©2011-14

Coverage Criteria

T-node

Case	Too small	Ship where	Ship method	Cust type	Pay method	Same addr	CC valid
TC-1	No	Int	Air	Bus	CC	No	Yes
TC-2	No	Dom	Air	Ind	CC	–	No (abort)

T-branch

Case	Too small	Ship where	Ship method	Cust type	Pay method	Same addr	CC valid
TC-1	No	Int	Air	Bus	CC	No	Yes
TC-2	No	Dom	Land	–	–	–	–
TC-3	Yes	–	–	–	–	–	–
TC-4	No	Dom	Air	–	–	–	–
TC-5	No	Int	Land	–	–	–	–
TC-6	No	–	–	Edu	Inv	–	–
TC-7	No	–	–	–	CC	Yes	–
TC-8	No	–	–	–	CC	–	No (abort)
TC-9	No	–	–	–	CC	–	No (no abort)

[P&Y p.260: Figures 14.8 & 14.9]

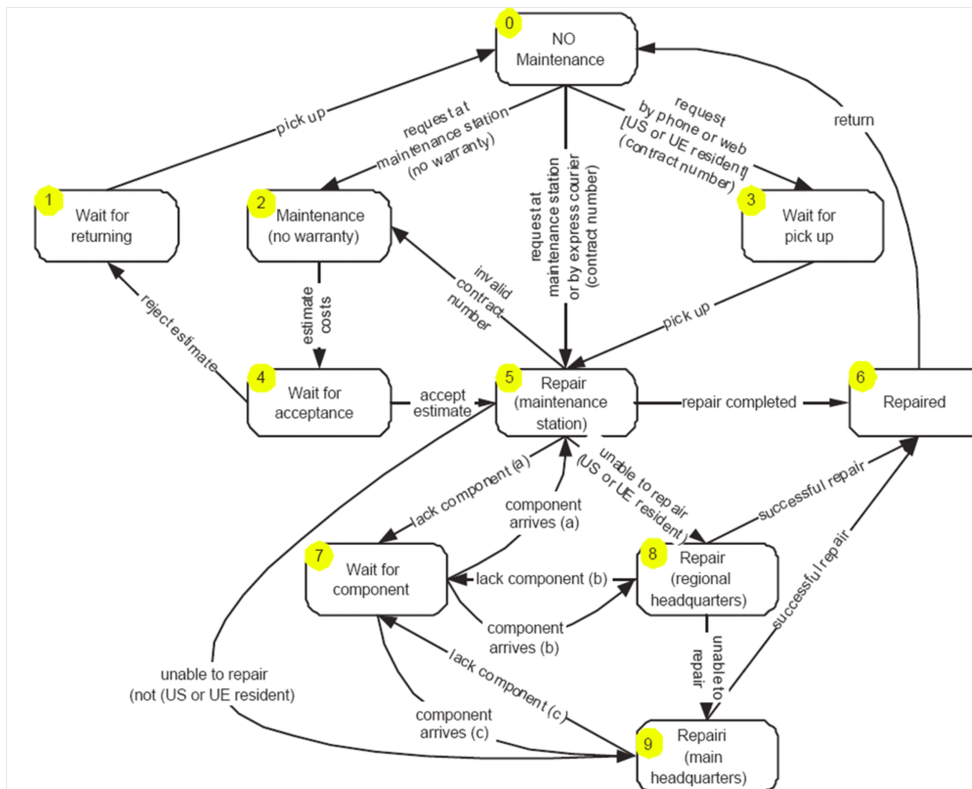
 ©2011-14

Finite State Machines

- Good at describing interactions in systems with a small number of modes.
- Good at describing transducers (via finite state machines).
- Widely used in industry (via Statecharts + associated tools — see Harel reference in the Readings).
- Most systems are *'infinite state'* (or effectively so), but many systems are finite state + parameters – there are a finite set of states that control the way data is moved around.
- Good examples are systems like communication protocols or many classes of control systems (e.g. automated braking, flight control systems).
- Transitions are generally made on inputs (e.g. the discovery of some state of affairs – e.g. that the wheels are locked in a braking system).
- Good for describing interactive systems that rarely reach a final state.

 ©2011-14

Finite State Machines



[P&Y p.248: Figure 14.2]

 ©2011-14

Designing Tests

- Sequence of inputs that drives the system though some sequence of transitions.
- We use coverage criteria to measure how successful we are in exploring the specification.
- The simplest criterion is that we have covered all transitions.

T-Cover

TC-1	0 – 2 – 4 – 1 – 0
TC-2	0 – 5 – 2 – 4 – 5 – 6 – 0
TC-3	0 – 3 – 5 – 9 – 6 – 0
TC-4	0 – 3 – 5 – 7 – 5 – 8 – 7 – 8 – 9 – 7 – 9 – 6 – 0

[P&Y p.249: Table 14.1]

 ©2011-14

Other Coverage Criteria

- Implementations of FSM specification often have more state than the specification (i.e. they may exhibit history sensitivity). Typically because we introduce extra management into the system (e.g. the possibility to undo some number of transitions).

- As a result we often use other coverage criteria that explore the behaviour more thoroughly, e.g.:
  - *Single state path coverage:* collection of paths that cover the states
  - *Single transition path coverage:* collection of paths that cover all transitions.
  - *Boundary interior loop coverage:* criterion on number of times loops are exercised.

 ©2011-14

Other Coverage Criteria

- Errors included by adding an Error state.
- We can consider mutation to discover how the system responds to unexpected inputs.
- We can use probabilistic automata to represent distributions of inputs if we want to do randomised testing.

Grammar-based Testing

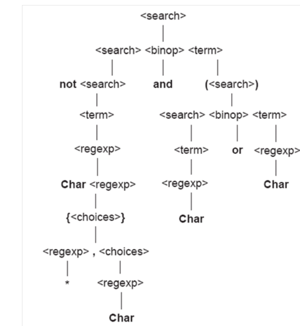
- Grammars are used to describe well-formed inputs to systems.
- We might want to know the system responds correctly to all such inputs.
- We can use grammars to generate sample inputs.
- We can use coverage criteria on a test set to see that all constructs are covered.
- We can use probabilistic CFGs to capture distributions on particular inputs.
- As XML is used increasingly to define transfer formats etc grammar-based testing is becoming increasingly important.
- Grammar-based testing is fairly easy to automate.

 ©2011-14

A Sample Grammar and Test Case

⟨search⟩ ::= ⟨search⟩ ⟨binop⟩ ⟨term⟩ | `not` ⟨search⟩ | ⟨term⟩
⟨binop⟩ ::= `and` | `or`
⟨term⟩ ::= ⟨regexp⟩ | `(` ⟨search⟩ `)`
⟨regexp⟩ ::= *Char* ⟨regexp⟩ | *Char* | `{` ⟨choices⟩ `}` | `*`
⟨choices⟩ ::= ⟨regexp⟩ | ⟨regexp⟩ `,` ⟨choices⟩

hfill [P&Y p.261: Figure 14.11]



[P&Y p.264: Figure 14.14]

Generating Tests

- Coverage criteria are important, e.g.:
  - Every production at least once
  - Boundary conditions on recursive productions – 0, 1, many
- Probabilistic CFGs (Context-Free Grammars) allow us to prioritise heavily used constructs.
- Probabilistic CFGs can be used to capture and abstract real-world data.
- We can easily generate erroneous data using simple mutations in the rules or final sentential forms.
- CFGs can be used to model interaction and low level detail in GUIs.

 ©2011-14

Choice Criteria

- What form does the specification take?
- Experience of the team in different methods.
- Availability and quality of tools
- Cost/benefit analysis on the range of techniques and the available budget (some approaches may require too much infrastructure)

 ©2011-14

Readings

Required Readings

- **Textbook (Pezzè and Young):** Chapter 5, Finite Models
- **Textbook (Pezzè and Young):** Chapter 11, Combinatorial Testing
- **Textbook (Pezzè and Young):** Chapter 14, Model-Based Testing

Suggested Readings

- David Harel, Statecharts: a visual formalism for complex systems, Science of Computer Programming, Volume 8, Issue 3, June 1987, pp. 231-274. http://dx.doi.org/10.1016/0167-6423(87)90035-9
- John Rushby, Using model checking to help discover mode confusions and other automation surprises, Reliability Engineering & System Safety, Volume 75, Issue 2, February 2002, pp. 167-177. http://dx.doi.org/10.1016/S0951-8320(01)00092-8