Model-Based Testing

Alexander Pretschner
TU Kaiserslautern and Fraunhofer IESE

Saarbrücken, 31/05/2010

Motivation

- The oracle problem
  - Automatically deriving tests that include fine-granular expected output information: more than robustness testing
  - Specifications (expected output) tend to be bad
- Common "methodologies" for deriving test cases are, because of their level of abstraction, not too helpful
  - "Build partitions"—but that's the nature of the beast
- Process of deriving tests not reproducible and not systematic; bound to the ingenuity of single engineers

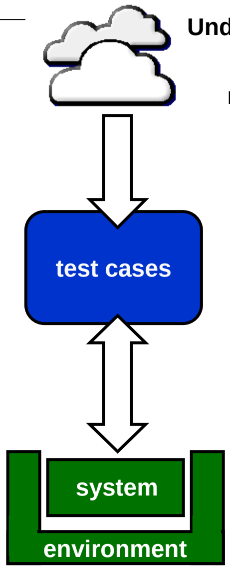
Overview

- Motivation
- Models and Abstraction
- Scenarios
- Selection Criteria
- Generation Technology
- Cost Effectiveness and Evidence
- Summary

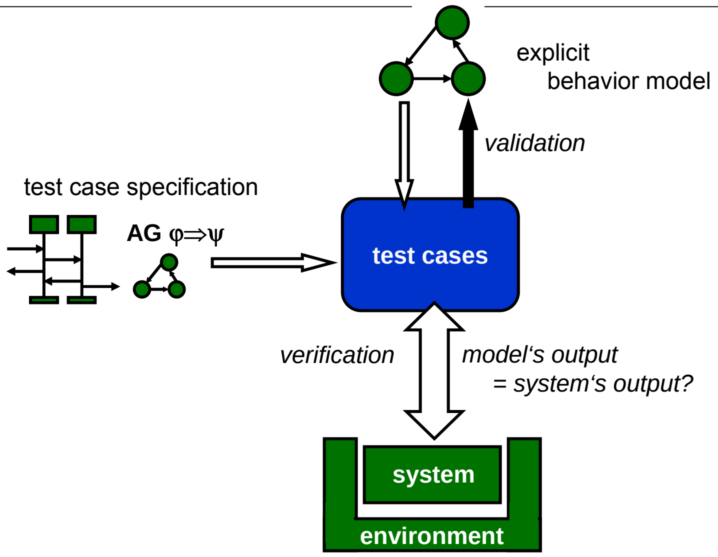
Goal of Today's Class

- Understand the ideas of model-based testing
- Understand where you have to think about its deployment
- Know what it can do and what it can't
- Know where and not automation is likely to be possible
- Be able to, in principle, conceive a set-up for model-based testing in your context
  - Decide on abstraction, build model, decide on test selection criteria, perform test case generation, execute generate tests, judge what you did
  - Clearly, that's domain-specific

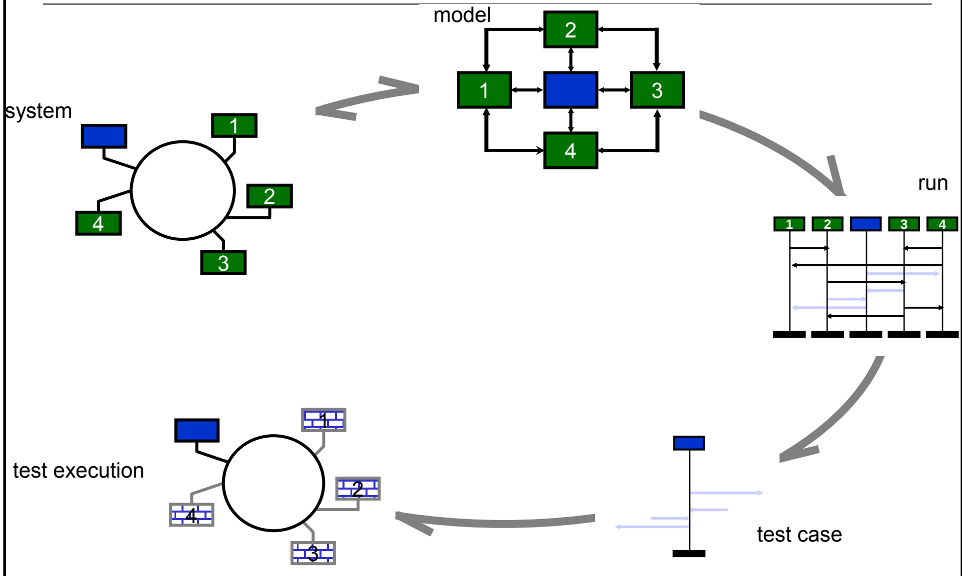
Testing



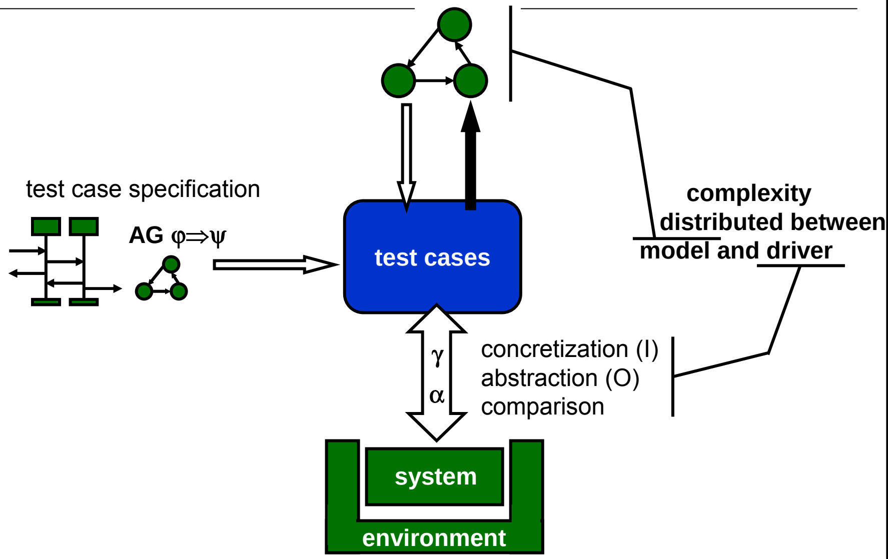
Model-Based Testing



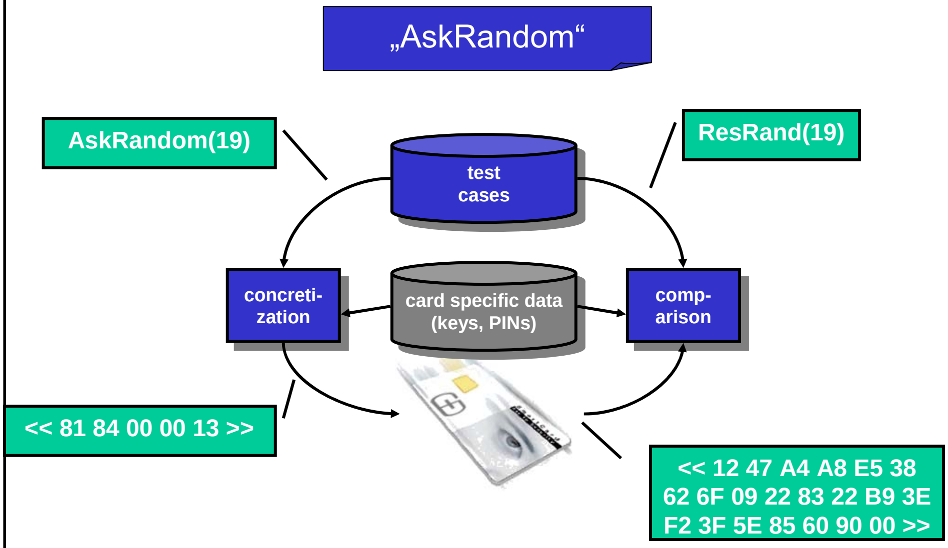
Test Generation and Execution



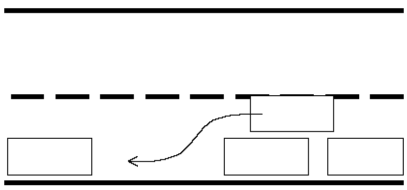
Levels of Abstraction



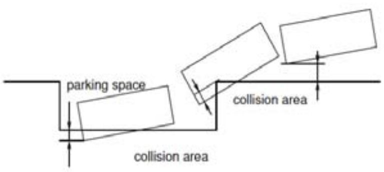
Levels of Abstraction: Example



Example II: Autonomous Parking



Functionality

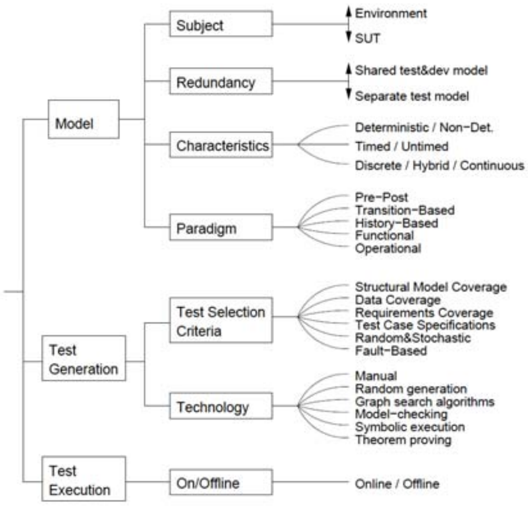


Abstract Functionality:

Don't enter collision area

Taken from Buehler, Wegener: Evolutionary Functional Testing of an Automated Parking System, CCCT'03

Flavors of Model-Based Testing



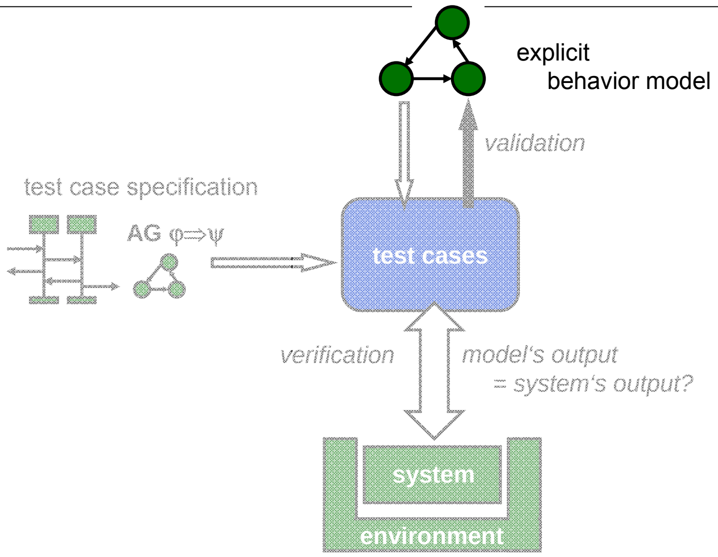
Utting, Pretschner, Legeard: A taxonomy of MBT, technical report 04/2006, University of Waikato, May 2006

Difficult Questions

- What is modeled? How are models validated?
- What is tested, and how is this specified?
- How are test cases computed and executed?
- Do explicit behavior models yield better and cheaper products?
  - Or is it better to just define test cases?
  - E.g., test cases in XP serve as specification
- Aren’t reviews or inspections more efficient and effective?

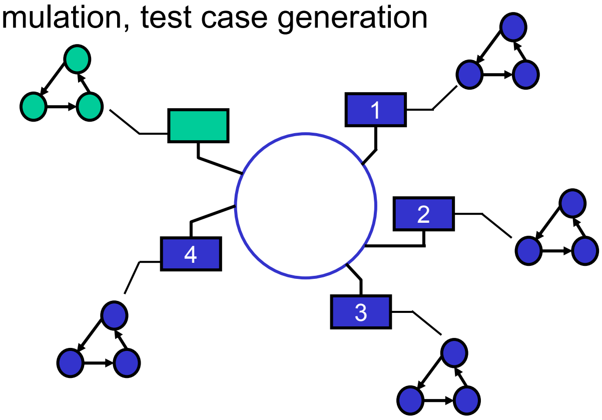
Overview

- Models
- Scenarios
- Selection Criteria
- Generation Technology
- Cost Effectiveness and Evidence
- Summary

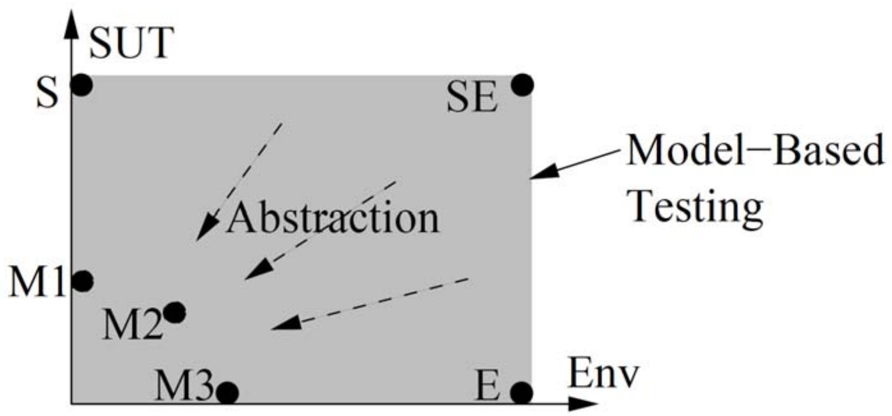


Implementation and Environment

- Models of (partial) environment often necessary
  - SW almost always based on assumptions (⇒ integration/system tests)
  - Simulation, test case generation



Abstraction: Models of SUT and Environment



Utting, Pretschner, Legeard: A taxonomy of MBT, technical report 04/2006, University of Waikato, May 2006

Purpose of Abstractions

- Insights into a system
- Specification
- Encapsulated access to parts of a system
- Communication among developers
- Code generation
- Test case generation
- …

One: Models encapsulate Details

- Like "abstractions" in programming languages: subroutines, exceptions, garbage collection, Swing
  - No or "irrelevant" loss of information
    - "macro expansion"
    - Example: MDA for communication infrastructure
  - Separation of concerns, orthogonality
- Matlab-Simulink-like
  - Block diagrams: architecture and behavior
  - 1:1 representation of a differential equation
  - Encapsulation of concrete computation
- Helpful for MBT but not sufficient if validation of model is done by simulation only
  - Is it easier to test a Java program than to test the corresponding bytecode?

Two: Models omit Details

- Simplification with “relevant” loss of information
- Intellectual mastery; “refinement”
- “Complexity essential, not accidental” [Brooks’87]
- Functionality, Data, Scheduling, Communication, Performance

Abstractions I

- Function
  - Restriction to a particular function(ality)
  - Detection of feature interactions?
- Data
  - No loss of information: binary numbers → integers
  - Loss of information: equivalence classes → 1 symbol
- Communication
  - ISO/OSI stack:
    complex interaction at bottom → 1 (inter-)action above
  - Corba, J2EE

Abstractions II

- Time (more general: QoS)
  - Ignore physical time; nondeterministic timeouts
  - Granularity of time
- Permutations of sequences of signals (underspecification in the model)
- Implies natural restrictions w.r.t. tests

Levels of Abstraction

- Model as precise as SUT—directly validate SUT!
- Reuse of model components?
  - Validate integrated model
- Reuse of environment models?
  - Directly test SUT
- Parametrization of the model?
  - Informal inductive argument
- One model as reference implementation?
  - Conformance tests—why not directly use test cases?

Behavior Models

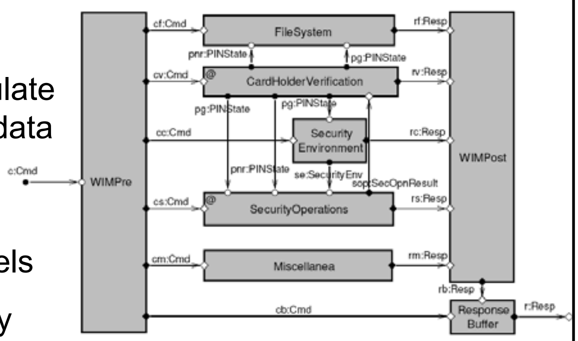
- Executability helps with validation
  - Prototypes
  - Some disagree: carrying out proofs is much better for validation
- Behavior models need not be executable
  - E.g., specification of a sorted array
  - Quantifiers very powerful modeling abstractions
- Many specification styles; many boil down to pre and postconditions
  - “declarative” rather than “operational”
- Doesn’t impact our analysis of model-based testing

So what?

- Encapsulation helpful if model is to be reviewed (not simulated/tested)
- But models for test case generation must be written down
  - Appropriate languages
  - SUT and environment
- Models “better” since “simpler”
  - But complexity essential, not accidental
  - Missing information must be given by a human
- Simplifying models for test case generation rather than for code generation!

Example – Part I

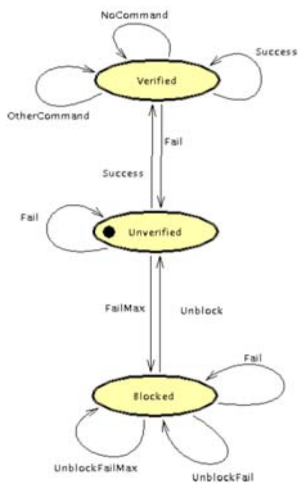
- Chip card
- Components encapsulate behavior and private data state
- Communication exclusively via channels
- Structure motivated by functional decomposition



Philipps et al., Model-based Test Case Generation for Smart Cards, Proc. FMICS'03

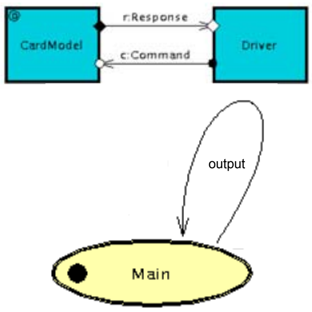
Example – Part I

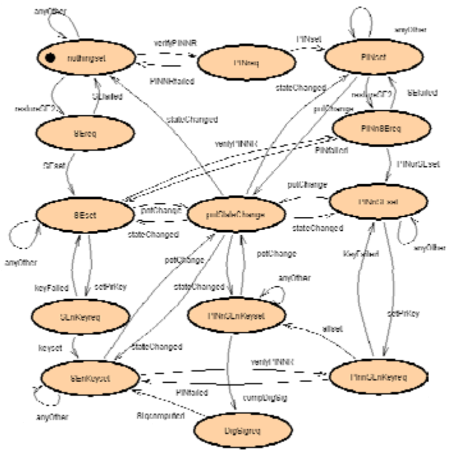
- Behavior of one CardHolderVerification component
- Wrong PIN increases PIN counter
- Max PIN counter → card blocked
- Extended Finite State Machine Transitions $i?X \wedge \gamma \wedge o!Y \wedge \alpha$



Example – Part I

- Environment models
  - Restrict possible input

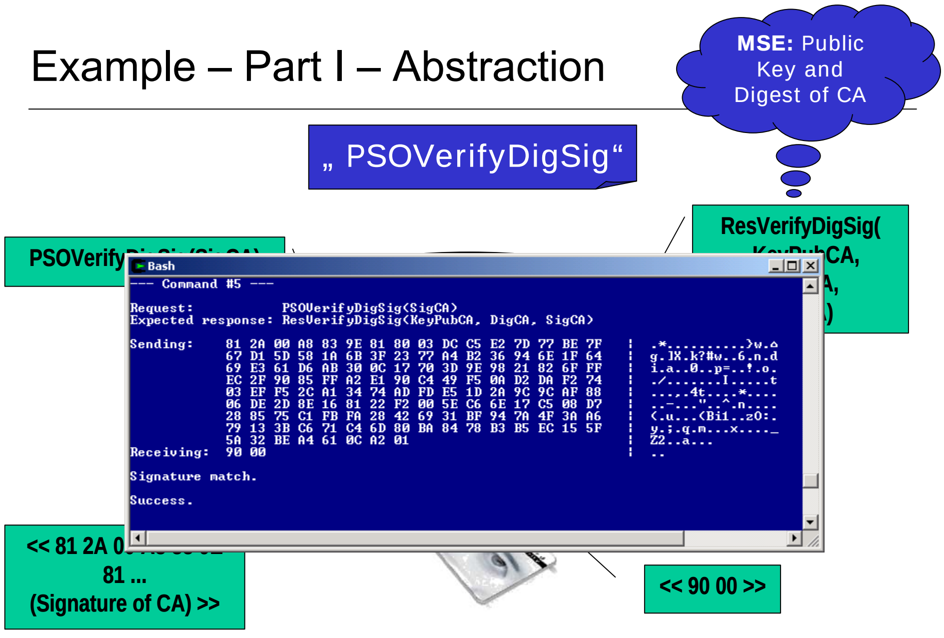




Example – Part I – Abstraction

- Function: rudimentary file system
- Random numbers: “rnd”
- No actual computation of crypto operations
  - Driver
- Abstract commands
  - No testing at the level of corrupt APDUs
  - Done separately
- No hardware-based attacks

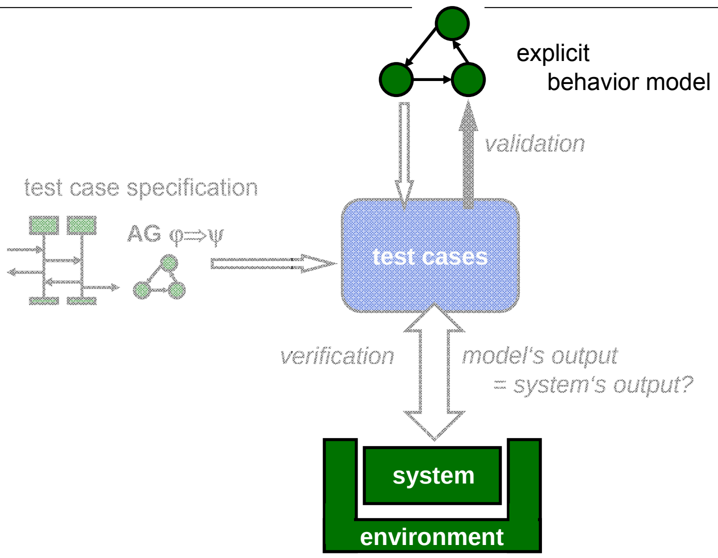
Example – Part I – Abstraction



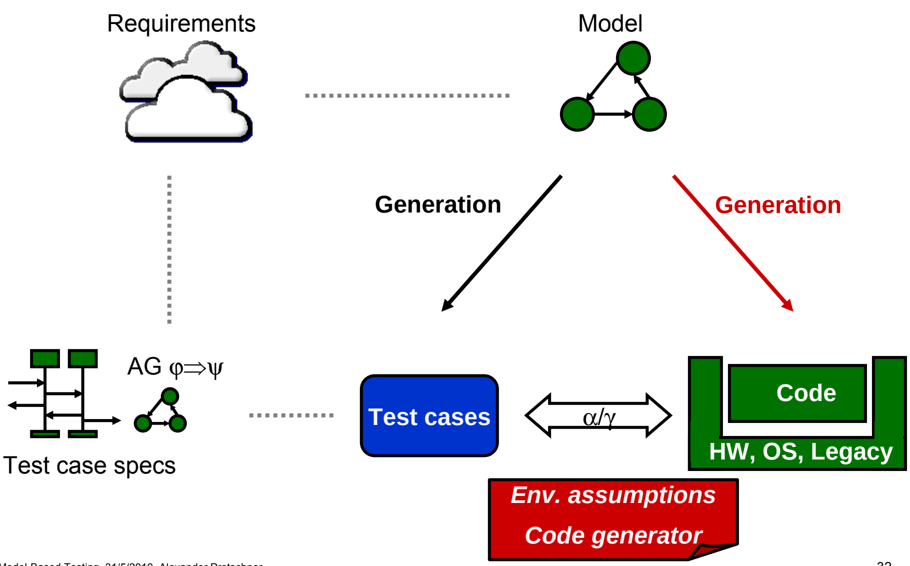
Slide: Jan Philipps

Overview

- ▶ Models
- ▶ Scenarios
- ▶ Selection Criteria
- ▶ Generation Technology
- ▶ Cost Effectiveness and Evidence
- ▶ Summary



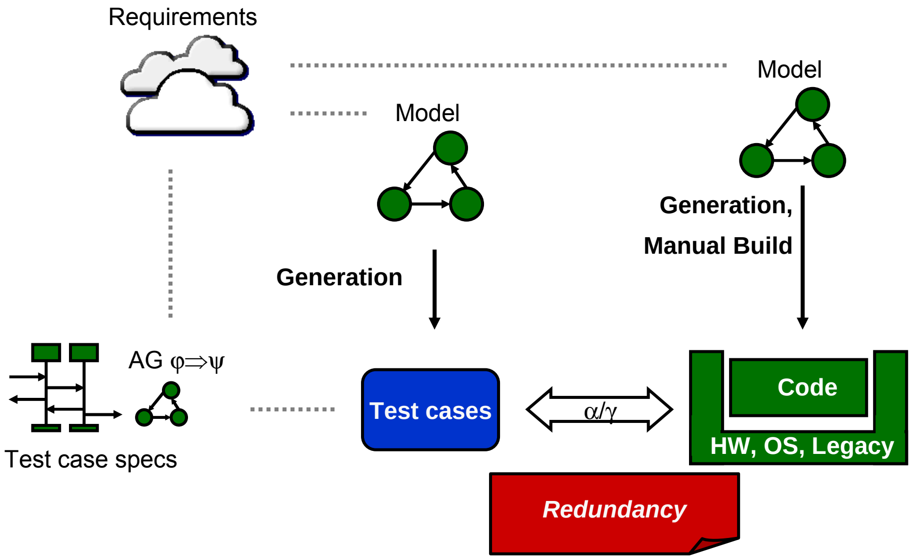
Scenario I: Tests and Code generated from 1 Model



Discussion: One Model for Both

- Generation: no redundancy → no verification
  - "exceptions" don't occur—model is valid, generator as well (or is it?)
- Tests for
  - Code generators (simulation and production)—MDD
  - Assumptions on the environment
  - Possibly performance/stress
  - Exceptions
- Models valid → that's alright!
  - Different flavor of MBT
  - No "double check" model ⇔ implementation
- Abstraction levels
- Test and development models
- Model as basis for manual implementation

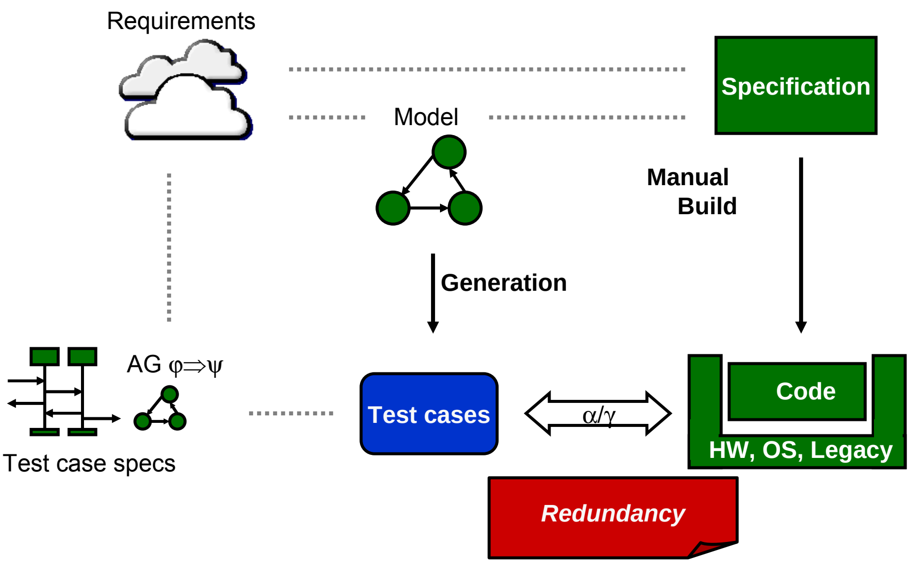
Scenario II: Two Models



Discussion: Two Models

- Expensive
- Redundancy
- Different levels of abstraction
- Both tests and code profit from the (alleged) advantages of model-based development
- Precise specifications
  - Car manufacturers and suppliers
  - Behavior models lead to better specifications
  - Model alone no (good) specification

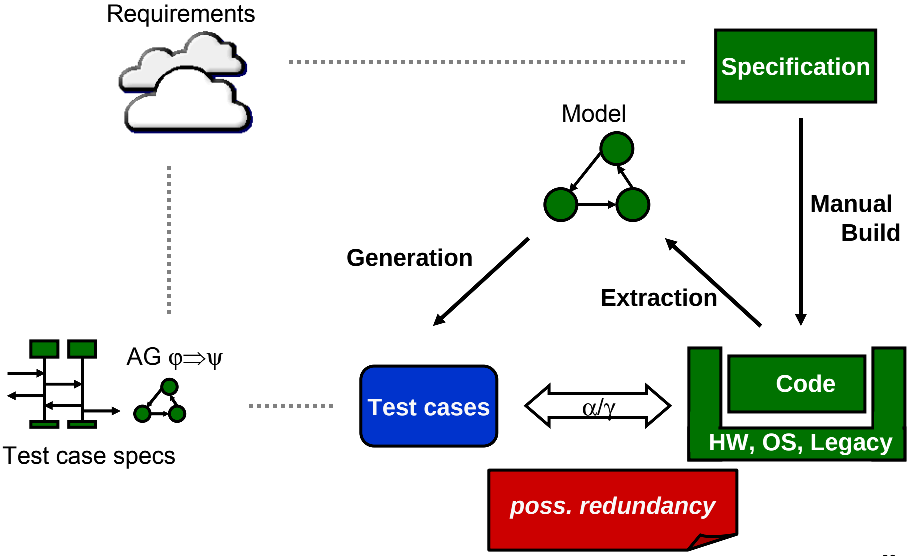
Scenario III: Model only for TC Generation



Discussion: model for test only

- Redundancy
- Expensive; concentration on critical parts possible (?)
- Interleaving code/model with changing requirements
- Specification doesn't profit from benefits of model-based development
- Assessment of new model-based testing technology
- "Conformance" tests: suppliers must show adherence to model
- Scenario of our running chip card example

Scenario IV: Model Extraction from Code



Discussion: Model Extraction

- Abstractions always bound to purpose and domain: automation?
- Automatic generation: redundancy?
- Interleaving code/model?
- Ex-post development of tests
- Assessment of new generation technology with manual extraction
- Tests for “exception/no exception” possible

Continuous Testing

- Assume execution and analysis of tests come at no cost
  - Generation of tests in the background
  - Execution of tests in the background
  - Abstraction level possibly exceptions/no exceptions
- Maturity of software
  - Too many detected errors → tedious analysis
- Embedded systems
  - Execution takes time
  - Simulators
  - Business information systems are different

Summary I

- 1 model for both
  - No redundancy, no double check
  - "Test models" different from "development models"
  - Cf. argument on using abstract models
- 2 distinct models
  - Redundancy
  - Expensive
  - Different levels of abstraction possible

Summary II

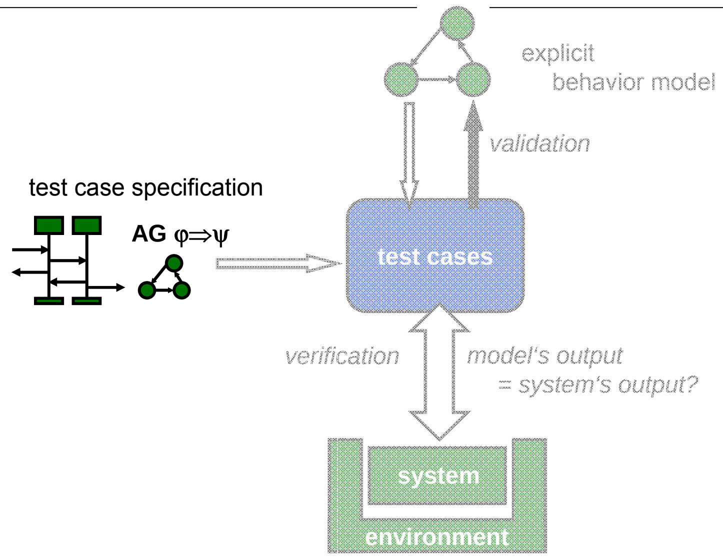
- 1 model for tests
  - Redundancy
  - Changing requirements: interleaving model and code development?
  - OEM builds model, suppliers have to conform to it
- 1 model from code
  - Redundancy?
  - Ex-post development of test cases only
- [Pretschner'05]

And in the real world?

- Model-based testing in the hardware industry
  - Need for redundancy is acknowledged
  - Reluctance in the SW industry!
- Stochastic testing: reliability engineering
- Continuous systems in Matlab: test code generators
- Models primarily built for test case generation: stage of case studies
- For SW, I haven’t encountered the situation where two distinct models are built ($$$)
- Generate tests to validate models is rather common

Overview

- Models
- Scenarios
- Selection Criteria
- Generation Technology
- Cost Effectiveness and Evidence
- Summary

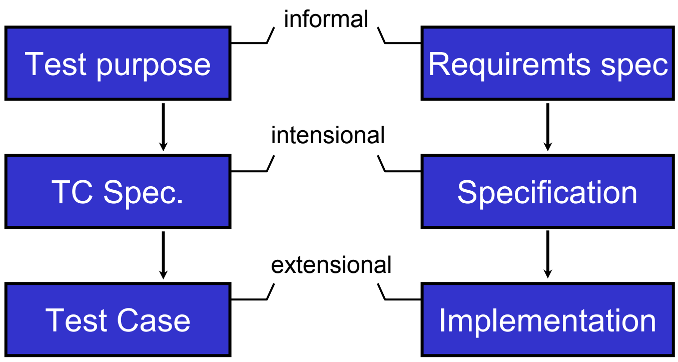


Test Purpose and Test Case Specification

- Familiar problem …
  - Irrelevant if model-based or not
- Test cases: selected “relevant” traces
- What‘s “relevant“? What‘s “good“?
- Test purpose informal, TC spec formal

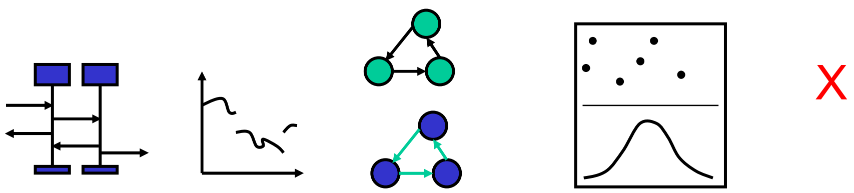
Test purpose, TC specification, test case

- TC spec. formalizes test purpose and renders it operational
  - E.g., an invariant cannot directly be tested



Selection Criteria

functional ad-hoc structural stochastic fault-based



Summary

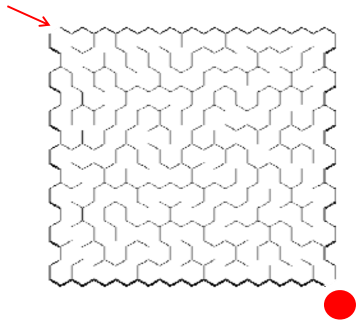
- Functional criteria
  - Specific to domain or application; requirements
  - Methodological support
- Structural criteria
  - Independent of domain
  - Data flow, control flow, data
  - Automatic generation of TC specs and test cases
  - Measurable
  - Ability to reveal faults unclear
  - Models of SUT and environment

Summary II

- Stochastic criteria
  - Uniform distributions: “purely at random”
  - User profiles
  - In general, not “worse” than structural criteria
- People tend to agree that there’s not one single good criterion!

Test Case Generation

- ► Search problem
- ► Techniques
  - ► Dedicated algorithms for dedicated criteria
  - ► (Bounded) model checking
  - ► Deductive theorem proving
  - ► Symbolic execution
  - ► [Lucio'05]



Search Problem

- ► Enumerate traces and select w.r.t. TC specification
- ► Respect constraints during enumeration
  - ► Functional criteria
- ► General problem: find traces that cover edges/nodes/special data values in the control flow and data flow graphs
  - ► Structural criteria
  - ► Directed/heuristic search
- ► Often, it is a good idea not to visit states twice
  - ► State storage
- ► Minimization of test suites not covered today

Overview

- Models
- Scenarios
- Selection Criteria
- Generation Technology
- Cost Effectiveness and Evidence
- Summary

Assumptions

- Effectiveness and cost effectiveness
  - Models help with getting requirements/specs straight
  - Test suite vs. model: creation and maintenance
- Existence of adequate level of abstraction
  - Abstraction and precision
  - Easy model validation and maintenance
  - Distribution of complexity
- Reuse
  - Simpler changes in the model (plus push button)
  - Adaptor and environment models/TC specifications

Evidence: (Cost) Effectiveness

- “Model-Based Testing does find errors”
- Different/more errors in SUT?
  - Farchi et al. ’02, Pretschner et al. ’05
  - Except for last study: no precise description of reference
  - Ongoing dispute on comparison with reviews
- Errors in model or specs
- Cost Effectiveness
  - Farchi et al. ’02, Bernard et al.’04, Sinha et al. ’06
  - “building tests took less time”
- In sum: hard to admit, but very little evidence!
  - But: neither empirical evidence about benefits of OO software

Coverage?

- Unsettled discussion on benefits of structural criteria
  - Inconclusive studies on both control and data flow
  - Not surprisingly, using such a criterion “leads to failures that would have gone undetected”
  - DO-178B recommends MC/DC for level A software
- Unclear if things change when used on specifications
- People agree: structural tests complement functional tests

Empirical Evidence

- Compare any “new” approach to random tests and “traditionally developed tests”
- Homogeneous systems?
  - Domain
  - Stage of development
  - Programming language
  - Skills of programmers
  - Complexity
- As always: generalization?!

(Personal) Summary and Gut Feel

- Don’t rely on structural criteria only!
  - Large state spaces, big problems, anyway!
- Abstract models for testing for exceptions might be cost-effective
  - Run tests in the background
- Continuous testing if at no cost
- Model-Based Testing does find additional failures
  - But it’s not entirely clear if these wouldn’t also have been found as a result of carefully studying the specs
- Model in itself definitely helps (XP: tests are spec/model)
- Not necessarily automated generation
- Plenty of other low-level problems in the real world

Overview

- Models
- Scenarios
- Selection Criteria
- Generation Technology
- Cost Effectiveness and Evidence
- Summary

Summary

- Model of SUT and environment at different levels of abstraction
  - Abstraction compulsory
  - Oracle
- Possibly automated test generation with environment model (statistical testing; structual criteria on encoded scenarios) and structure of model of the SUT
  - But we still need to tell the machine what a good test consists of!
- Different scenarios
- Different generation technologies
- As usual, little evidence …

My Personal Bottom Line

- Go for it! I do eat my own cooking!
- Don't use it to write a script; model a stack?
- Use of models beyond testing important
  - Specifications, contracts for suppliers/OEM
  - Cost-effectiveness unlikely if nobody uses models anyway
- Different levels of abstraction are acceptable
- Not so sure about automation
- Enforcement of test rationales can help tremendously
- Use knowledge on earlier failures; user profiles

Literature

- M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, A. Pretschner, "Model-Based Testing of Reactive Systems", Springer Verlag, 2005
  - S. Sandberg, "Homing and Synchronizing Sequences", chapter 1 in [Broy et al.'05]
  - M. Krichen, "State Identification", chapter 2 in [Broy et al.'05]
  - H. Björklund, "State Verification, chapter 3 in [Broy et al.'05]
  - A. Gargantini, "Conformance Testing", chapter 4 in [Broy et al.'05]
  - A. Pretschner, J. Philipps, "Methodological Issues in Model-Based Testing", chapter 10 in [Broy et al.'05]
  - L. Lucio and M. Samer, "Technology of Test-Case Generation", chapter 12 in [Broy et al.'05]
- A. Pretschner, W. Prenninger, S. Wagner, C. Kühnel, M. Baumgartner, B. Sostawa, R. Zölch, T. Stauner: One Evaluation of Model-Based Testing and its Automation, Proc. ICSE 2005, pp. 392—401, 2005
- E. Farchi, A. Hartman, S. S. Pinter, Using a model-based test generator to test for standard conformance, IBM Systems Journal 41 (1):89-110, 2002
- E. Bernard, B. Legeard, X. Luck, F. Peureux, Generation of test sequences from formal specifications: GSM 11.11 standard case-study, SW Practice and Experience 34 (10):915 – 948, 2004
- A. Sinha, C. Williams, P. Santhanam, A measurement framework for evaluating model-based test generation tools, IBM Systems Journal 45(3):501-514, 2006