Model-based testing and verification of dependable systems

Armin Beer

SIEMENS

Overview

- Motivation
- State-of-the-art of testing dependable systems
- Case study: Electronic interlocking system for railways
- Project SoftNet Austria
- Research questions and answers
- Conclusion and outlook

A typical dependable event-based system

ICE accident in Thun / Switzerland
April 28, 2006
“Guardian angel on board”

“Collision of ICE train with 2 shunting locomotives”

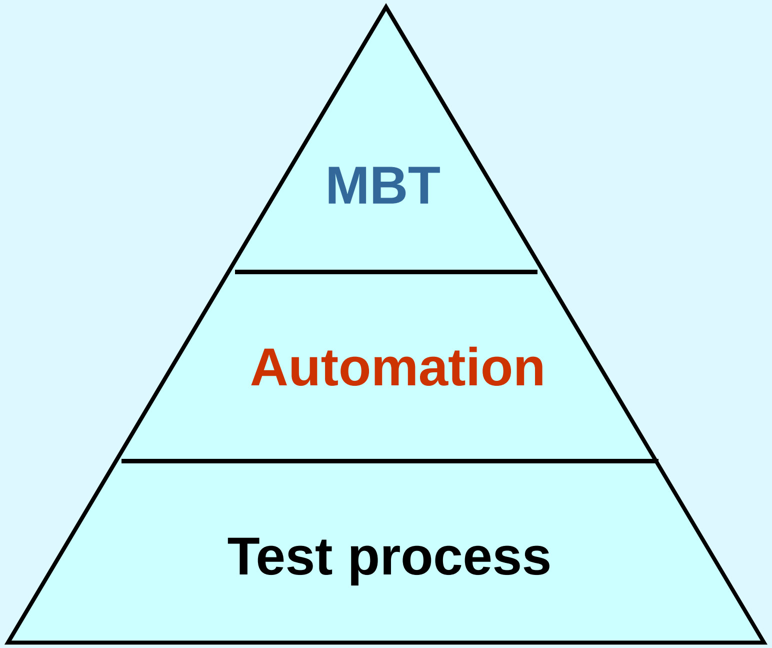
Major damage to property, 8 people slightly injured

“Testing can only prove the presence of bugs, not their absence”
Edsger Dijkstra

Definition of model-based testing (MBT)

Wikipedia (retrieved Nov. 21,2008)

- **Model-based testing** is **software testing** in which **test cases** are derived in whole or in part from a **model** that describes some (usually functional) aspects of the **system under test (SUT)**



Standards relevant for safety-critical systems

SIEMENS

Sources / standards involved …

- **IEC 61508**: Functional safety of electronic safety-related systems
- **EN 50128**: Software for railway control and protection systems
- **DO 178B**: Software considerations in airborne systems and equipment certification
- Traceability matrix regarding testing activities

Examples from various projects:

- **Interlocking systems of transportation systems - case study**
- „GSM on board“ for Airbus
- Certification of modeling tool ASCET

Traceability matrix according to IEC61508 & DO-178B

SIEMENS

Verification group	Technique	IEC61508/ SIL4	DO-178B
Module test and integration	Dynamic analysis and test Functional and black-box test	HR HR	X X
Software safety validation	Functional and black-box test Performance tests Probabilistic tests ...	HR M M	X
Dynamic analysis and test	Error seeding Structural tests ...	R HR	 X
Functional and black-box test	Boundary value analysis Equivalence class test ...	HR HR	X X
Modeling	Data flow diagrams State transition diagrams ...	HR HR	X

State-of-the-art of testing dependable systems

SIEMENS

- Standards do not describe **how** the recommended test techniques could be applied
- The testing standards established by the **ISTQB (International Software Testing Qualification Board)** are a basis for their application in projects
- A mature organization with processes at **CMMI level 3 or 4** can develop systems of high quality

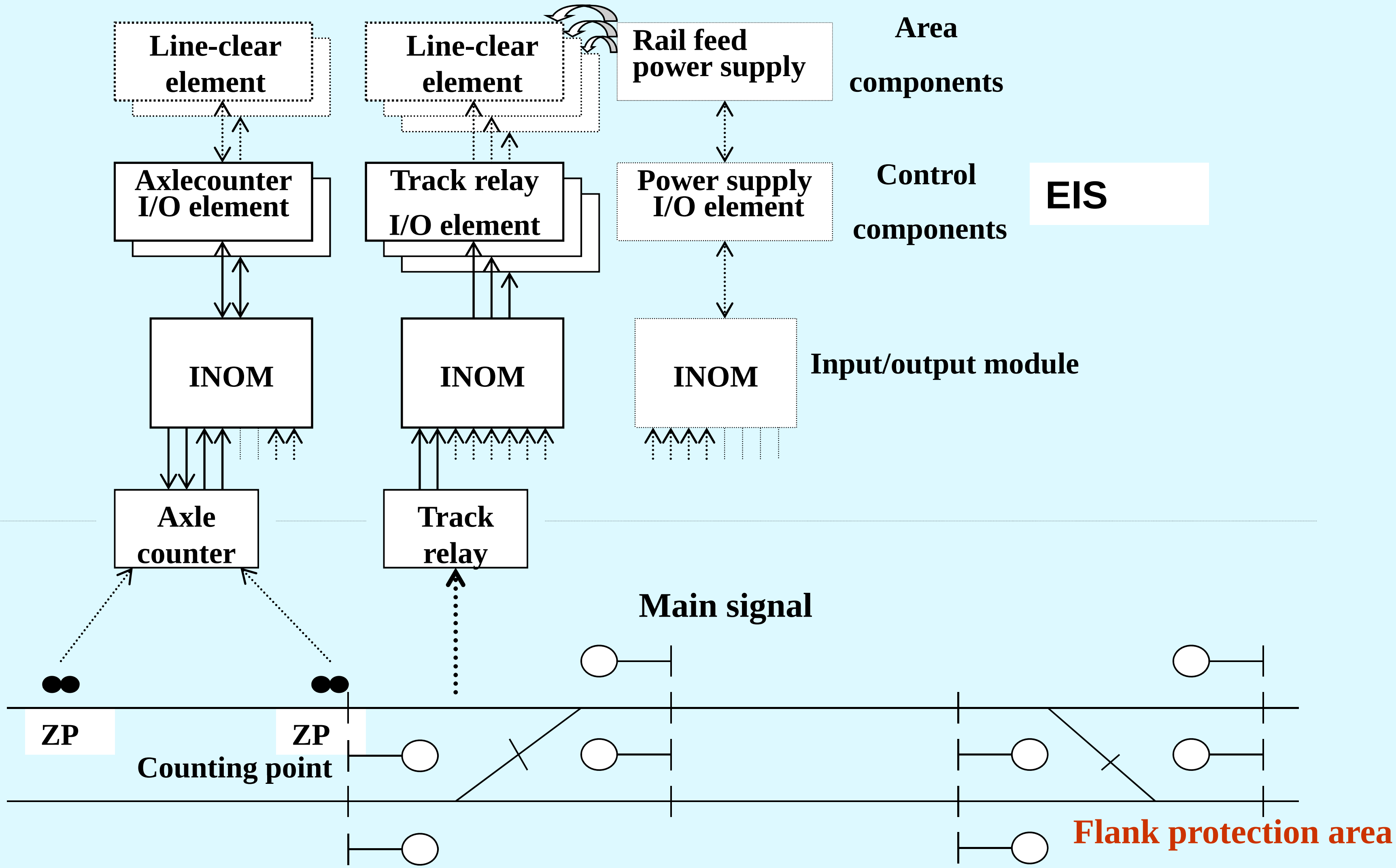
Case study: Electronic interlocking system (EIS)

SIEMENS

- **The project**
- **The testing approach**
- **Results and lessons learned**

The project: Electronic interlocking system for railways

SIEMENS



The project: Test of railway interlocking system

A railway interlocking system **controls and monitors**

- All the signals, track switches (forks),
- **Track vacancy sensors** and
- Other hardware devices in a given area (in most cases a major station and its surroundings)

- The operation of a railway system **obeys very strict rules** in order to prevent accidents

- The software in the railway interlocking system implements these operational rules

The project: Railway control system key properties

SIEMENS

Hardware

- Dedicated hardware (Siemens SMC86/ECC) with **triple redundancy** (2-out-of-3) for automatic hardware fault detection
- All I/O interfaces are redundant with antivalent electric signals
- There is no mass storage

Software

- The software is entirely developed using compilers that are certified for life-critical systems (Pascal and C++)
- **Programming conventions** restrict the usage of dynamic memory allocation, pointers, and floating point computations in a very strict way
- The **tailoring** for a concrete deployment (this could be e.g. Innsbruck central station) is entirely performed by configuration

The testing approach: Testing requirements

- Testing is performed according to the CENELEC standards EN 50126, EN 50128, and EN 50129
- Unit tests are performed with **95% (!) C2 (all paths!) coverage** on the source-code level
- For each of the remaining 5% of execution paths, a duly justification for non-coverage must be written (e.g. demonstrating the impossibility of a scenario).
- Preliminary integration tests are run on the so-called GESIM (simulation emulates the target operating system, I/O devices, etc.)
- Final tests are run on the actual **target hardware in the laboratory** (for each type of I/O port, at least one real world device (e.g. track switch engine, signal) must be used to validate the correctness of the drivers; the remaining ones may be emulated)

Case study: Test preparation and execution

SIEMENS

- The **requirements specifications in human language**, such that automated inference of test cases is not straight-forward
- Even for a life-critical system, exhaustive testing is not feasible due to **combinatory explosion of test cases**
- Test engineers develope test cases based on equivalence classes, boundary sets, and years of domain experience
- Automated test case generation is performed on the deployment-specific level using a tool
- The actual test runs fully automated (controlled by scripts e.g. simulating a moving train, or faults in the outdoor hardware)
- On the system level, there are roughly **3500 test cases**
- A complete, fully automated run of the acceptance test catalogue takes about **10 days**

Research in FFG - Project SoftNet Austria

SoftNet SIEMENS

- Cooperation between partners from **industry and research institutions** (Technical University Graz, SCC Hagenberg, Siemens, Cirquent, Cicero etc.): **http://www.soft-net.at**
- **Topics:**
  - Complexity in software engineering
  - Test management
  - Model-based testing and test-case generation
  - Mutation analysis
  - ...
- **Typical areas of application:**
  - Safety-critical software e.g. embedded systems
  - Systems of high availability e.g. VoIP
  - Critical banking and insurance applications

Research questions

- **RQ1: Which techniques for modeling are applicable in complex systems?**
- **RQ2: Which probabilistic randomized testing technique for software safety validation is applicable?**
- **RQ3: How could error seeding and mutation analysis be used to check the effectiveness of a module test suite?**
- **RQ4: What type of recommender system is needed to support the selection of the appropriate testing technique?**

Answers: RQ1

SIEMENS

RQ1: Which techniques for modeling are applicable in complex systems?

Issues to be solved …

- Usability of modeling techniques
- Different levels of abstraction in behavioral specifications
- Application of n-dimensional equivalence classes
- Test-case generation for different targets

Methods and Tools

- Model-based testing process
- CECIL (Cause-Effect Coverage Incorporating Linear boundaries) - Test Methodology
- IDATG (Integrating Design and Automatic Test-case Generation)
- Model transformation from UML2 to Input/Output Symbolic Transition Systems

Test-case generation methods

SIEMENS

Graph-oriented methods

- Directed graph with cycles
  - a set of edges, which connect two nodes in a defined order, from a start node to a goal-node
  - A finite order of nodes and edges is called a graph

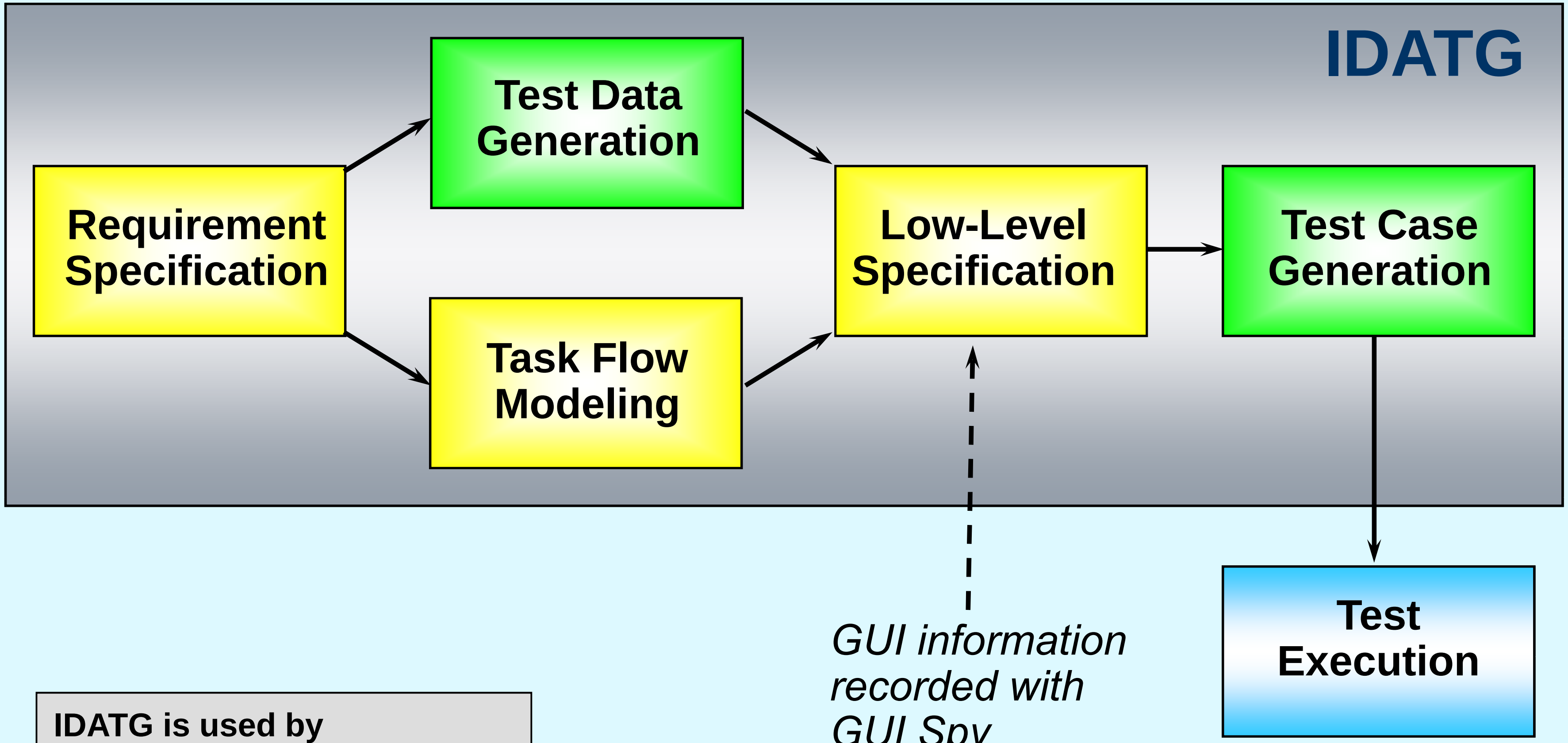
- The coverage of all states and events by test cases is a search in a graph for paths, from a start node to a goal node

Data-oriented methods

- Generation of test data applying the methods of equivalence classes / boundary values, semantic conditions etc.

Model-based testing with IDATG (Integrating Design and Automatic Test-case Generation)

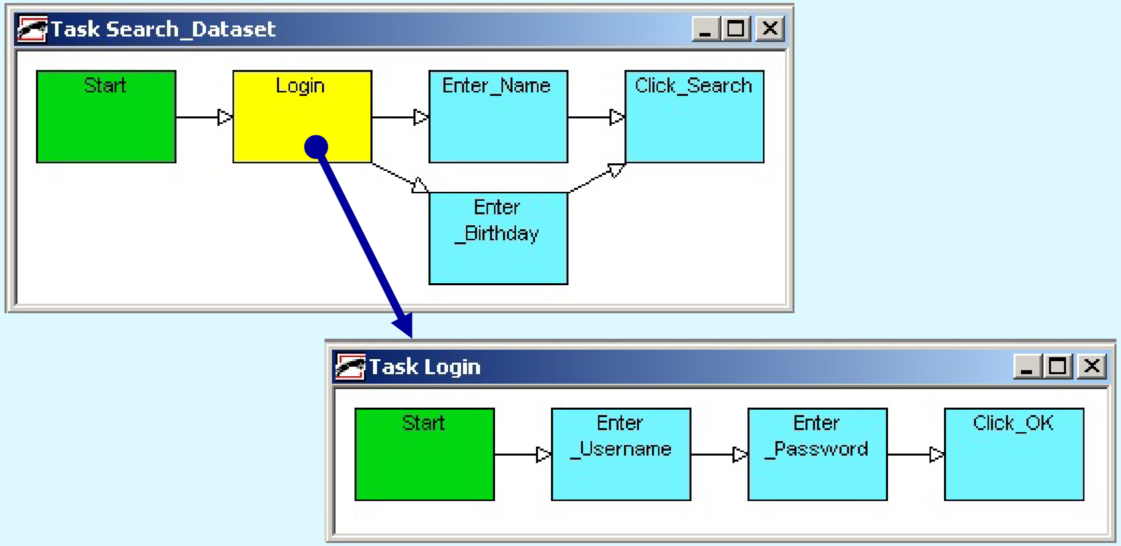
SIEMENS



IDATG is used by

http://idatg.siemens.at

IDATG task flow modeling



- The sequence of test steps for each task can be defined with the **Task Flow Editor**
- **Building Block Concept:** Each step may either represent an atomic step **(blue)** or an entire sub task flow **(yellow)**
- Re-use of building blocks **minimizes** effort for **test maintenance**

Test data generation: the CECIL method

CECIL = Cause-Effect Coverage Incorporating Linear boundaries

Test data design method combining the benefits of:

- Equivalence partitioning
- Boundary value analysis
- Cause/effect analysis

Properties:

- Well suited for complex semantic dependencies
- High error-detection potential
- Difficult to apply manually, but can be mostly automated

Example "Vehicle insurance"

We want to test an application that calculates the annual **insurance premium** for a vehicle (Motorcycle, Car, or Van).

The basic premium depends on the **engine power** (in HP) and the **vehicle type**:

	Motorcycle
< 25 HP	50 €
25 - 49 HP	75 €
>= 50 HP	100 €

	Car	Van
< 60 HP	100 €	200 €
60 - 99 HP	200 €	400 €
>= 100 HP	300 €	600 €

For person groups with a higher accident risk, the premium is 20% higher. These groups are: all persons older than 65 years, men younger than 25, and women younger than 21.

Only persons aged between 21 and 65 are allowed to drive a van. To drive a car or motorcycle, a person must be at least 18.

CECIL Task 1: Problem analysis

SIEMENS

- Identify **input variables**, their types and definition ranges. Represent **enumeration types** as numbers:
  - Vehicle ***Type*** [0=Motorcycle, 1=Car, 2=Van]
  - ***HP*** [0..9999]
  - ***Age*** [0..999]
  - ***Gender*** [0=Male, 1=Female]
- Introduce **effect variables** to express interim results:
  - ***Baseprice*** (depends on *Type* and *HP*)
  - ***Extracharge*** (depends on *Age* and *Gender*)

CECIL Task 2: Define causes and effects

SIEMENS

- Express dependencies as cause/effect pairs:

ID	Cause	Effect
SMALL_BIKE	Type = 0 ∧ HP < 25	Baseprice = 50 €
MEDIUM_BIKE	Type = 0 ∧ HP >= 25 ∧ HP < 50	Baseprice = 75 €
BIG_BIKE	Type = 0 ∧ HP >= 50	Baseprice = 100 €
SMALL_CAR	Type = 1 ∧ HP < 60	Baseprice = 100 €
MEDIUM_CAR	Type = 1 ∧ HP >= 60 ∧ HP < 100	Baseprice = 200 €
BIG_CAR	Type = 1 ∧ HP >= 100	Baseprice = 300 €
SMALL_VAN	Type = 2 ∧ HP < 60	Baseprice = 200 €
MEDIUM_VAN	Type = 2 ∧ HP >= 60 ∧ HP < 100	Baseprice = 400 €
BIG_VAN	Type = 2 ∧ HP >= 100	Baseprice = 600 €
OLD_PERSON	Age > 65	Extracharge = 20%
YOUNG_MALE	Gender = 0 ∧ Age < 25	Extracharge = 20%
YOUNG_FEMALE	Gender = 1 ∧ Age < 21	Extracharge = 20%
NORMAL_MALE	Gender = 0 ∧ Age >= 25 ∧ Age <= 65	Extracharge = 0%
NORMAL_FEMALE	Gender = 1 ∧ Age >= 21 ∧ Age <= 65	Extracharge = 0%
I_TOO_YOUNG	Age < 18	Invalid
I_VAN_TOO_YOUNG	Type = 2 ∧ Age < 21	Invalid
I_VAN_TOO_OLD	Type = 2 ∧ Age > 65	Invalid

Rules for causes and effects

- Cause/effect pairs are either assigned to an **effect variable or** to the invariant ***Invalid***
- All causes affecting the same effect variable must be **mutually exclusive**
- Each possible **input combination** must satisfy exactly one cause for each effect variable or be *Invalid*.
- All causes must be **linear** in nature
- It is not necessary to repeat the **definition ranges** (e.g., *Age > 65* automatically assumes *Age ≤ 999*)

Generation of test data

SIEMENS

- Use a **linear programming** algorithm to find boundary points of the intersection space

  e.g., (*HP, Type, Age, Gender*):

  {(0, 0, 66, 0), (24, 0, 66, 0), (24, 0, 999, 0), (24, 0, 999, 1)}

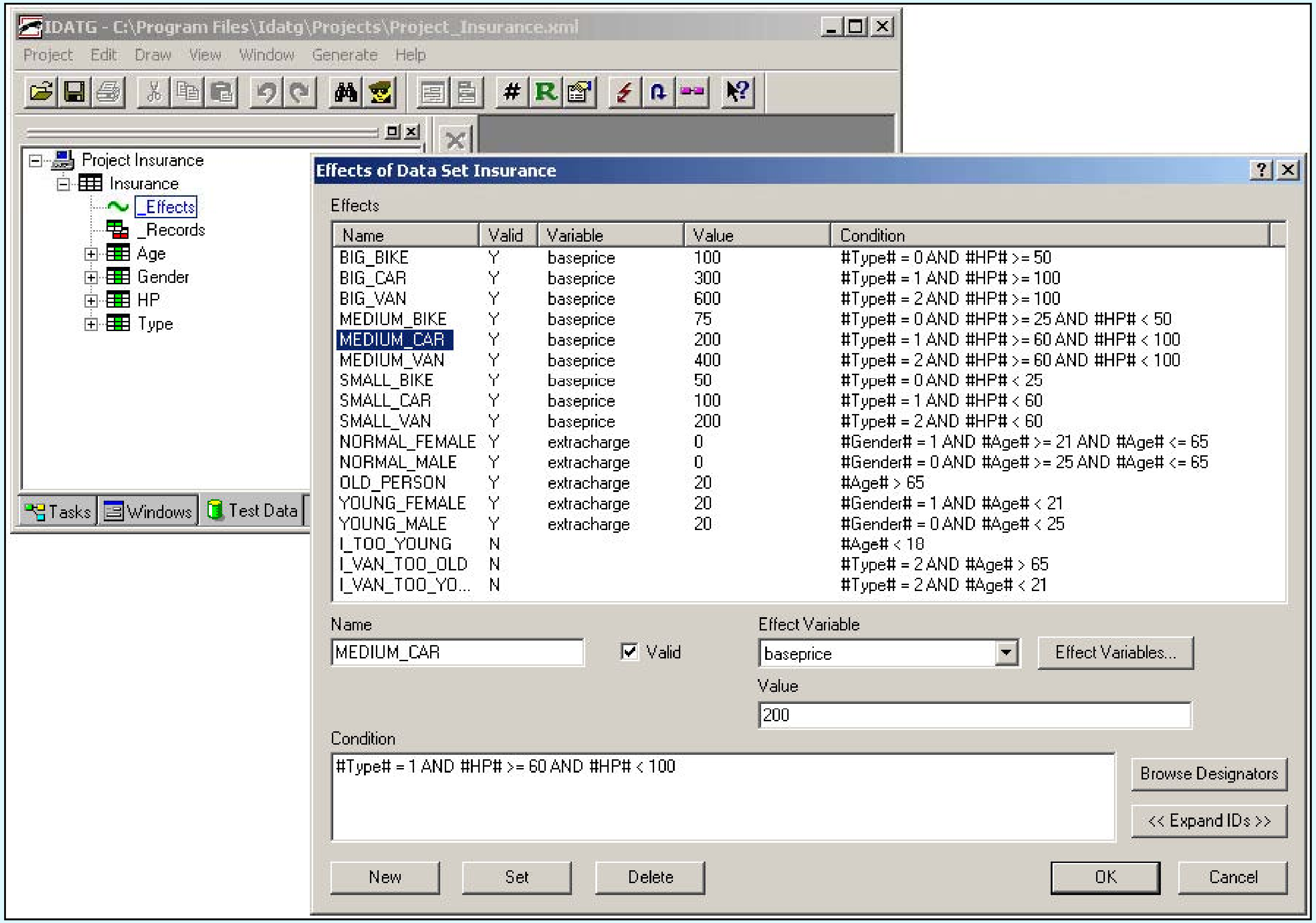
- Calculate the **expected results** using the values of the effect variables

  e.g., *Baseprice = 50€, Extracharge = 20% => Premium = 60€*

- Repeat with other cause combinations until all **causes are covered at least once** or no more combinations are possible

User interface of IDATG tool

SIEMENS



Answers: RQ2

SIEMENS

RQ2: Which probabilistic randomized testing technique for software safety validation is applicable?

Issues to be solved …

- Classical testing may be insufficient because of
  - too few test cases
  - too short test time
- Test case explosion

Methods and Tools

- Statistical Testing is an expansion of Model-Based Testing that deals with state-transition diagrams
- Statistical analysis tools provide the ability to derive indicators on SW quality properties e.g. fault residual, availability, performance

Randomized task flows

SIEMENS

- **Until now,** task flows were only used to express typical user scenarios
- New idea:
  - Generate **long random sequences** of building blocks
  - Consider **semantic conditions** in order to avoid invalid test cases
- Mapping of task flows to a conventional **formal model (EFSM)** is necessary
- **EFSM = Extended Finite State Machine**
  = FSM + variables, conditions, updates

Conversion of task flows into EFSM

SIEMENS

- **Mapping** of steps/blocks to states/transitions
- Task flows presuppose a lot of **implicit information** concerning the behaviour of GUI objects
- This implicit information **must be made explicit** in the form of conditions and updates
- Task flows are not a complete behavioural model => some **details have to be supplied**

Generation strategies

SIEMENS

Random Walk

- Simplest and cheapest method
- Coverage cannot be foreseen
- Search often runs into dead ends

Explicit Search / Model Checking

- Expensive (state space explosion)
- A certain coverage level can be guaranteed
- Intelligent search algorithm finds feasible paths out of dead ends

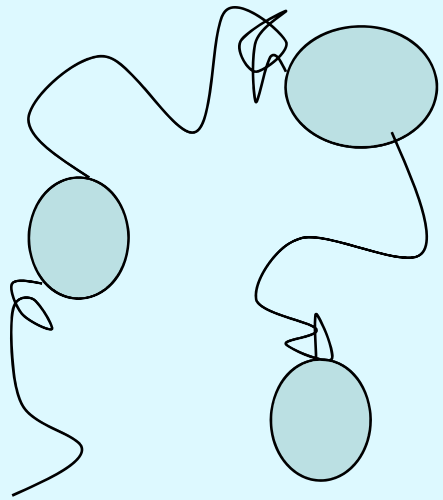
Combination of strategies

SIEMENS

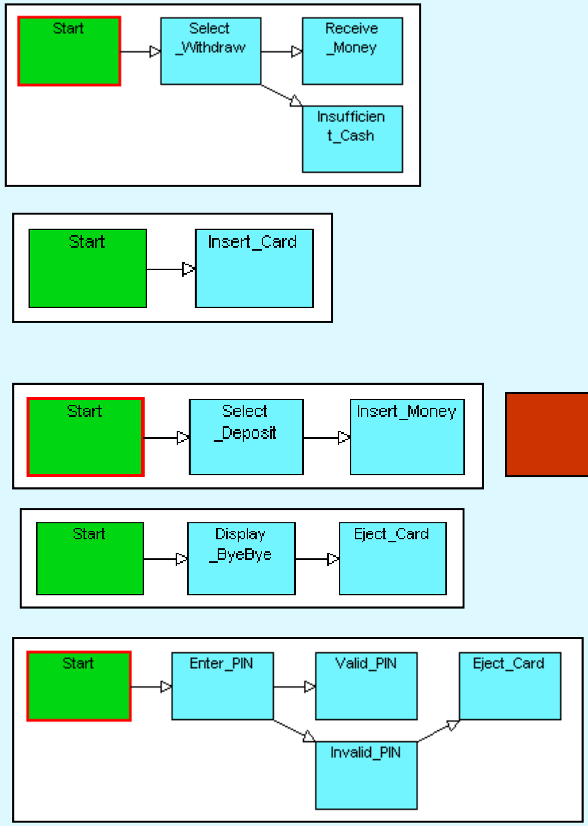
Hybrid Approach

(developed by Dr. Gordon Fraser, TU Graz)

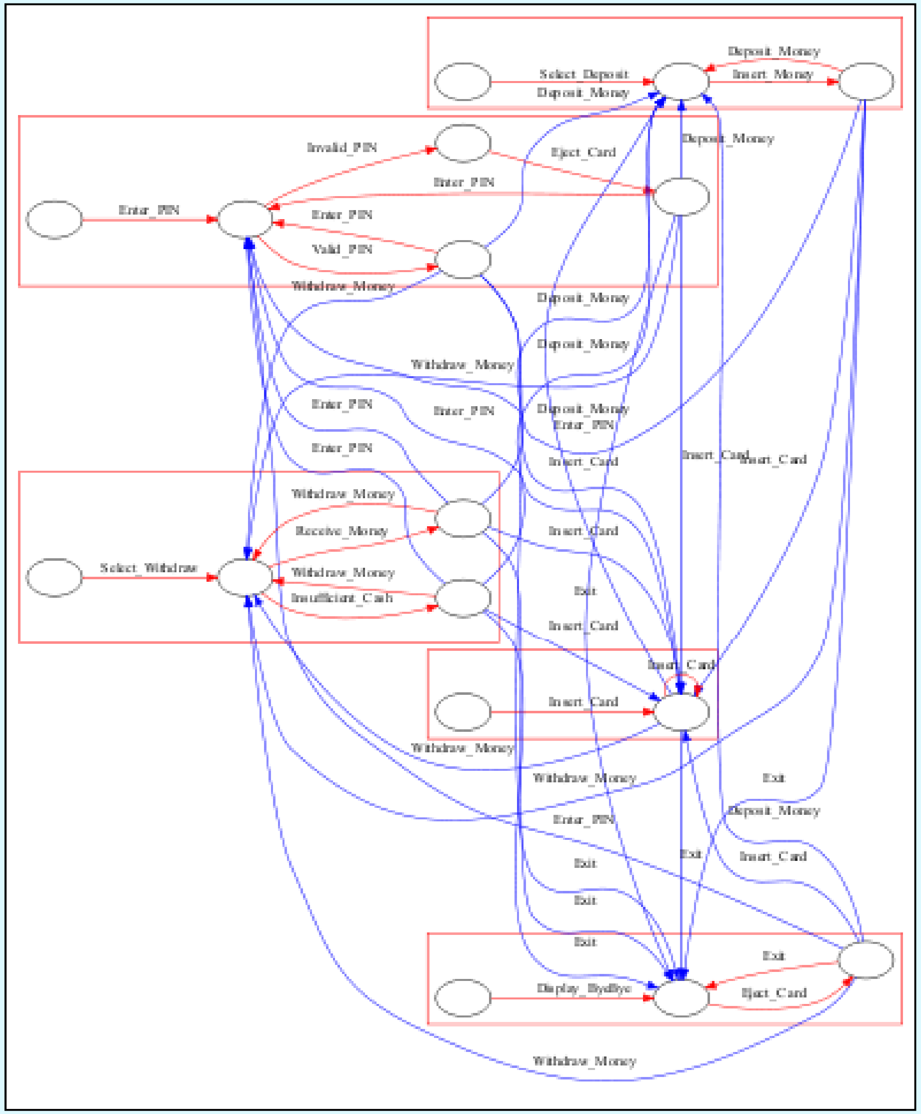
- Use Random Walk until reaching a local minimum („dead end“)
- Use Explicit Search for finding a path out of the minimum
- Repeat until the desired length or coverage criterium has been reached



Example: ATM



Start
Select _Withdraw
Receive _Money
Insufficien t_Cash
Start
Insert_Card
Start
Select _Deposit
Insert_Money
Start
Display _ByeBye
Eject_Card
Start
Enter_PIN
Valid_PIN
Eject_Card
Invalid_PIN



Answers: RQ3

SIEMENS

RQ3: How could error seeding and mutation analysis be used to check the effectiveness of a module test suite?

Issues …

- A program is well tested if all relevant faults are detected and removed
- Coupling Effect (DeMillo et al., 1978): test cases that detect simple faults can also detect more complex faults
- Competent Programmer Hypothesis (Acree et al., 1979): Programs are close to being correct

Methods and tools

- Objective: How well does a test-suite perform at detecting faults?
- Given:JUnit-test suite and program that passes the test-suite
- Create mutant programs
- Run test-suite against each mutant
- Mutation Score: Ratio
  killed mutants / total mutants

http://www.ise.gmu.edu/~ofut/mujava/

Answers: RQ4

RQ4: What type of recommender system is needed to support the selection of the appropriate testing technique?

Issues …

- Interpretation of textual specification and creation of a formal model
- Context-dependent selection of the appropiate test-case design method
- Usabilty aspects in respect to domain experts

Methods and tools

- Charactarisation **schema for testing techniques of Vegas** (S. Vegas: Identifying the Relevant Information for Software Testing Technique Selection; Proceedings of the 2004 International Symposium on Empirical Software Engineering)
- Development of prototype of **recommending system in SoftNet**

Conclusion and outlook

- **Verification and validation of safety-critical systems is the main research topic in SoftNet**
- **Model-based testing and automatic test case generation techniques have to support different levels of abstraction**
- **Statistical or randomized testing is currently implemented in the tool IDATG**
- **Mutation analysis was successfully used in determining the error-detection capability of different test suites**

Next steps:

- **Development of a recommender system for the selection of test-case design techniques**
- **Case studies and evaluation of methods / tools at Siemens**

Literature

SIEMENS

- Beer A., Mohacsi S., “Efficient Test Data Generation for Variables with Complex Dependencies”, IEEE Int. Conference on Software Testing, Verification and Validation (ICST) in Lillehamer, April 2008
- Dura H., Madeira H., “Emulation of Software Faults: A Field Data Study and a Practical Approach,” Published online 6 Nov. 2006.
- Fraser G., Peischl B., Wotawa F., “A Formal Model for IDATG Task Flows“, SNA-TR-2007-P2-03, SoftNet-Report, 2007
- IEEE Computer Society, “IEEE Standard Classification for Software Anomalies”, December 2, 1993
- Mohacsi S., “Practical Experience with Test Case Generation: Higher Product Quality - Reduced Test Costs“, 1.SoftNet-Workshop „Testen und Verifikation“, 7.11.2007 an der TU in Graz, 2007
- Peischl B., “A Survey on Standards and Certification of Safety Critical Software“, SNA-TR-2006-01, SoftNet-Report, 2006
- Robinson-Mallett, Ch., “An Example Application of Statistical Testing Distributed Software“, Fraunhofer Institute for experimental SW engineering, 2007
- Thurnher Ch., “Model Transformation from UML State Machines to Input/Output Symbolic Transition Systems”, master thesis at TU Vienna, 2008

Contact Information

Armin Beer

Phone: +43 (0) 676 5055670

EMail: armin.beer.ext@siemens.com