Quality, Reliability, and (Model-Based) Testing of Embedded Systems

Jan Tretmans

TNO – Embedded Systems Innovation, Eindhoven
Radboud University, Nijmegen
Research cooperation with leading Dutch high-tech multinational industries & SME's

Research cooperation with all Dutch universities with embedded systems research

Research cooperation in EU projects
Embedded Systems

Quality & Testing
Embedded Systems

or: What do Dykes, De Ruyter, and Wafer Scanners have in common
What do Dykes, De Ruyter, and Wafer Scanners have in common
What do Dykes, De Ruyter, and Wafer Scanners have in common
What do Dykes, De Ruyter, and Wafer Scanners have in common
What do Dykes, De Ruyter, and Wafer Scanners have in common
What do Dykes, De Ruyter, and Wafer Scanners have in common
Trend: Software in Embedded Systems

- electronics
- mechanics
- physics/chemistry, etc.

Time:
- 1970
- 2000

Relative effort:
- 100%
Quality of Embedded Systems

Software is brain of system

- software controls, connects, monitors almost any aspect of ES system behaviour
- majority of innovation is in software

Software determines quality and reliability of Embedded System

- often > 50 % of system defects are software bugs
Quality

Exclusive: Pentagon report faults F-35 on software, reliability

BY ANDREA SHALAL-ESA
WASHINGTON | Thu Jan 23, 2014 3:36pm EST

Software glitch hindered Vettel’s qualifying
2014 Australian Grand Prix

March 15, 2014 at 10:00 am by Keith Colantine

Sebastian Vettel’s qualifying performance was impaired by a software problem on his car, Red Bull have confirmed.

Vettel was knocked out in the second phase of qualifying having been 2.4 seconds off new team mate Daniel Ricciardo’s pace.

A faulty sensor was later found to have contributed to Vettel’s problem. Team principal Christian Horner said: “It was unlucky for Seb.”

“His engine software meant he was down on power with extremely poor drivability and we need to understand that, as it compromised his qualifying.”

Despite his difficulties Vettel said the team have “made a big
Challenges for Embedded Systems Testing

1. multi-disciplinarity
2. complexity
3. connectivity
4. component-based development
5. change: variability and evolvability
6. uncertainty
1. Multi-disciplinarity

- Combination of physics/mechanics/electronics … with computer/software
- Requires various expertise
- Testing such combinations requires simulation
1. Multi-disciplinarity

- **Virtualization**
  - models to simulate/emulate physical and environment parts in testing
  - intelligent stub, in-the-loop testing
  - because real system is: expensive, infeasible, dangerous, too slow, too fast, cannot produce error scenarios, …
2. Complexity

Testing effort grows exponentially with system size

Testing cannot keep pace with development

- $x : [0..9]$ → 10 ways that it can go wrong
  - 10 combinations of inputs to check

- $x : [0..9]$ → 100 ways that it can go wrong
  - 100 combinations of inputs to check

- $y : [0..9]$ → 1000 ways that it can go wrong
  - 1000 combinations of inputs to check

→ combinatorial explosion of required testing effort
2. Complexity

Machine with 300 parameters

- $2^{300} = 10^{90}$ different configurations
- \#atoms on earth = $10^{50}$, \#atoms in known universe = $10^{80}$

Completely testing ‘+’ for 32-bit Int

- $2^{32} \times 2^{32} = 10^{19}$ test cases
- 1 nsec / test = 585 years of testing
2. Complexity

Machine with 300 parameters

- \(2^{300} = 10^{90}\) different configurations
- \#atoms on earth = \(10^{50}\), \#atoms in known universe = \(10^{80}\)
3. Connectivity

- Blurring boundaries of systems → everything connected
- Systems-Of-Systems
  - Dynamically connected systems
  - Not under own control
- Software is glue
  - with internal and external world
- Testing:
  - what is SUT?
- Virtualization
  - which systems are available for testing?
  - which systems must be virtualized?
- Dynamics
  - run-time testing and integration
4. Components: Decompose Complexity

Focus of system architecting

Decomposition

Creating

Focus of integration

Composition
4. Components: Integration Challenges

Components come from anywhere
4. Integration of Boeing 787
5. Change: Variability, Evolvability

Different customers want different products
- different platforms, contexts
- product lines
- variability
- mass customization

Combinatorial explosion:
which varieties shall be tested, and when
5. Configurations
5. Change: Variability, Evolvability

Products evolve ever faster

- continuous delivery
deliver fast, deliver often
- test fast, test often
- test automation, model-based testing

Importance of regression testing:
- risk: destroy something which worked previously

changes in systems(-of-systems) after delivery:
run-time testing and health-monitoring

Paradoxes:
- software does not suffer from wear and tear, but software is repaired and patched more often than any other system part that does suffer from it
- late changes are a source of bugs, yet software is presumed to be flexible, so late changes are still often made
6. Uncertainty

• Sometimes you don’t know …..
  – testing a search engine,
    weather forecast, …
  – systems-of-systems,
    big data, …

• Sometimes you don’t want to know …..
  – no details
  – abstraction
  – particular view

Uncertainty of test outcomes & oracles
  – non-determinism, probabilities, constraints
Trends & Challenges

- complexity
- connectivity
- multidisciplinarity
- components
- quality characteristics
- embedded systems
- quality & testing challenges
- change
- variability
- evolvability
- uncertainty
(Model-Based) Testing
Testing

Checking or measuring some quality characteristics of an executing software object by performing experiments in a controlled way w.r.t. a specification
Quality: There is more than Testing

- Quality control
- Testing
- Model checking
- Requirement management
- Qualification
- CMM
- Certification
- Walk-through
- Static analysis
- Debugging
- Reviewing
- Verification
- Software process

QUALITY
Model-Based Testing

MBT

next step in test automation

+ test generation
+ result analysis
MBT: Example Models
1. Manual testing

System Under Test

pass fail
2: Scripted Testing

1. Manual testing
2. Scripted testing

![Diagram](Image)
3: Keyword-Driven Testing

1. Manual testing
2. Scripted testing
3. Keyword-driven testing
4: Model-Based Testing

1. Manual testing
2. Scripted testing
3. Keyword-driven testing
4. Model-based testing
MBT: next step in test automation

- Automatic test generation
  + test execution + result analysis
- More, longer, and diversified test cases
  more variation in test flow and in test data
- Model is precise and consistent test basis
  unambiguous analysis of test results
- Test maintenance by maintaining models
  improved regression testing
- Expressing test coverage
  model coverage
  customer profile coverage
Model-Based Testing
with Labelled Transition Systems
and ioco
MBT: Labelled Transitions Systems

- SUT ioco model
- sound \downarrow \uparrow exhaustive
- SUT passes tests

LTS test generation

- set of LTS tests

LTS test execution

- SUT behaving as input-enabled LTS
- input/output conformance ioco

SUT passes tests
Conformance: $ioco$

$$i\ ioco\ s\ \overset{\text{def}}{=}\ \forall\ \sigma\in\text{Straces}\,(s) :\ out\ (i\ \text{after}\ \sigma)\ \subseteq\ out\ (s\ \text{after}\ \sigma)$$

$s$ is a Labelled Transition System

$i$ is (assumed to be) an input-enabled LTS

$$p\xrightarrow{\delta} p\ =\ \forall\ !x\in L_U\cup\{\tau\}.\ p\xrightarrow{!x}$$

$$\text{Straces}\,(s)\ =\ \{\ \sigma\in (L\cup\{\delta\})^*\ |\ s\xrightarrow{\sigma}\ \}$$

$$p\ \text{after}\ \sigma\ =\ \{\ p'\ |\ p\xrightarrow{\sigma} p'\ \}$$

$$\text{out}\,(P)\ =\ \{\ !x\in L_U| p\xrightarrow{!x},\ p\in P\ \}\ \cup\ \{\ \delta| p\xrightarrow{\delta} p,\ p\in P\ \}$$
Conformance: \( ioco \)

\[
i \ ioco \ s \iff \forall \sigma \in Straces(s): \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma)
\]

Intuition:

\( i \ ioco \)-conforms to \( s \), iff

- if \( i \) produces output \( x \) after trace \( \sigma \),
  then \( s \) can produce \( x \) after \( \sigma \)

- if \( i \) cannot produce any output after trace \( \sigma \),
  then \( s \) cannot produce any output after \( \sigma \) (quiescence \( \delta \))
Example: \textbf{iocco}

\begin{itemize}
  \item ?quart
  \item ?dime
  \item ?dime ?quart
  \item ?dime ?dime ?quart
  \item ?dime ?dime
  \item ?dime ?dime ?quart
  \item ?dime ?dime
  \item ?dime
  \item ?dime
  \item ?dime ?dime
  \item ?dime
  \item ?dime
\end{itemize}

\textit{specification model}

\textbf{non-determinism uncertainty under-specification}
Example: *ioco* Test Generation

**specification model**

```
s
  ?dime
  !tea
  !coffee

i ioco s
```

**generated test case**

```
t
  !dime
  ?tea
  ?coffee
  ?choc

  δ
  pass
  pass
  fail
  fail

i
  ?dime
  !tea
  !choc
```

**implementation**

```
i ioco s = \( \forall \sigma \in \text{Straces}(s) : \text{out}(i \text{ after } \sigma) \subseteq \text{out}(s \text{ after } \sigma) \)
```
MBT with *ioco* is Sound and Exhaustive

Test assumption:

\[ \forall SUT \in \text{IMP} . \exists m_{SUT} \in \text{IOTS} . \]
\[ \forall t \in \text{TESTS} . \]
\[ \text{sut passes } t \iff m_{SUT} \text{ passes } t \]

Prove soundness and exhaustiveness:

\[ \forall m \in \text{IOTS} . \]
\[ ( \forall t \in \text{gen}(s) . m \text{ passes } t ) \]
\[ \iff m \text{ ioco } s \]

SUT comforms to s

exhaustive  
\[ \uparrow \quad \downarrow \]
sound

SUT passes gen(s)
Model-Based Testing

Tools
MBT : A Tool

model-based test generation

system model

SUT

test execution

pass fail
MBT: A Tool

off-line MBT

model-based test generation

system model

requirements

ideas

test cases

test execution

SUT

test harness

verdict test result analysis

pass fail

MBT
MBT: A Tool

- System model
- Test generation + execution
- SUT
- Test harness
- Verdict
- Test result
- Analysis

On-the-fly MBT
Requirements
Ideas
Pass fail
MBT : Many Tools

- AETG
- Agatha
- Agedis
- Autolink
- Axini Test Manager
- Conformiq
- Cooper
- Cover
- DTM
- fMBT
- $\forall$st
- Gotcha
- Graphwalker
- JTorX
- MaTeLo
- MBTsuite
- M-Frame
- MISTA
- NModel
- OSMO
- ParTeG
- Phact/The Kit
- PyModel
- QuickCheck
- Reactis
- Recover
- RT-Tester
- SaMsTaG
- Smartesting CertifyIt
- Spec Explorer
- StateMate
- STG
- Temppo
- TestGen (Stirling)
- TestGen (INT)
- TestComposer
- TestOptimal
- TGV
- Tigris
- TorX
- TorXakis
- T-Vec
- Uppaal-Cover
- Uppaal-Tron
- Tveda
- ..........
MBT : Many Tools

- AETG
- Agatha
- Agedis
- Autolink
- Axini Test Manager
- Conformiq
- Cooper
- Cover
- DTM
- G∀st
- Gotcha
- Graphwalker
- JTorX
- MaTeLo
- MBT suite
- M-Frame
- MISTA
- NModel
- OSMO
- ParTeG
- Phact/The Kit
- QuickCheck
- Reactis
- Recover
- RT-Tester
- SaMsTaG
- Smartesting CertifyIt
- Spec Explorer
- Statemate
- STG
- Temppo
- TestGen (Stirling)
- TestGen (INT)
- TestComposer
- TestOptimal
- TGV
- Tigris
- TorX
- TorXakis
- T-Vec
- Uppaal-Cover
- Uppaal-Tron
- Tveda
- ............

MBT : Many Tools
Model-Based Testing

Applications
Electronic Passport

New Passport

- Machine Readable Passport (MRP, E-passport)
- with chip (JavaCard), contact-less
- storage of picture, fingerprints, iris scan, .......
- access to this data protected by encryption and a new protocol
- few years ago released in EU

Our job: testing of e-passports

- emphasis on access protocol
  == exchange of request-respons messages between passport and reader (terminal)
MBT for E-Passports: Model
model-based test tool

java drivers adapter

test runs

state-based model

pass fail

english specifications

e-passport & wireless reader

SUT

TTCN

Test cases model

Test generation

Test execution

model-based test tool
MBT for E-Passports : Test Runs
MBT for E-Passports: Results

- Tested:
  - Basic Access Control (BAC)
  - Extended Access Control (EAC)
  - Active Authentication (AA)
  - Data Reading

- Tests up to about 2,000,000 test events
  - complemented with manual tests

- No error found ......
• **Microsoft**
  - EU, US anti-trust case: make Windows more open for competitors
  - producing documentation for Windows protocols
  - check documentation w.r.t. real product by model-based testing

- 75 protocols, 50 fte, 10,000 reqs, average 1000 LoM/model
- MBT tool: Spec Explorer
- Productivity gain: 42 % and 34 % w.r.t. traditional testing
MBT : Who’s Done It

• MBT User Conference
  – telecom, automotive, embedded, not so much administrative yet
  – many companies are experimenting, pilot projects
  – not off-the-shelf
  – connection to other development tools is important issue
  – flexibility, maintainability is big plus
  – sometimes combined with scrum, agile
  – not everything with MBT

+ gain (time, cost, coverage, ...) : 10 - 40 %
MBT: Next Step Test Automation

**State of Research**

- Model-Based Testing
- Keyword-Driven

**State of Practice**

- Scripted
- Manual Testing
MBT: State of the Art

• promising, emerging
• a number of successful applications
• many companies are experimenting

MBT: State of Practice

Reasons

• technical
• tools
• organizational
• maturity of testing
• educational
• . . . . .

MBT: State for the Future

(for High-tech Embedded Systems)

• ?
MBT Technical Issue: State + Data

model
with large data

? 2 MB XML file

[ not correct ]

! 2 MB with a Error Messages

[ correct ]

! 4 MB processed XML file
MBT: Next Generation Challenges

- abstraction
- concurrency
- parallelism
- size
- usage
- profiles for testing
- scalability
- link to MBSD
- model composition
- state + complex data
- heterogeneous components
- model selection criteria
- test selection criteria
- multiple paradigms integration
- uncertainty nondeterminism
- quality constraints for continuous testing
- model driven development
MBT: Next Generation Challenges

- State of the Art
- MBT Tools
- abstraction
- concurrency
- parallelism
- usage profiles for testing
- scalability
- link to MBSD
- state + complex data
- model composition
- test selection criteria
- multiple paradigms integration
- uncertainty nondeterminism

Next Generation Challenges

- Next Generation Challenges
Next Generation MBT: TorXakis

- abstraction
- concurrency parallelism
- state + complex data
- model composition
- test selection criteria
- multiple paradigms integration
- uncertainty nondeterminism
- usage profiles for testing
- scalability
- link to MBSD
- model
- SUT
- TorXakis

Next Generation MBT:

- TorXakis
- scalability
- usage profiles for testing
- model composition
- test selection criteria
- multiple paradigms integration
- uncertainty nondeterminism
- concurrency parallelism
- state + complex data

Modeling of state and complex data with multiple paradigms integration for test selection criteria and uncertainty nondeterminism.
TorXakis: On-the-Fly MBT

on-the-fly MBT

TorXakis

system model

SUT

pass fail
TorXakis: Overview

Models
- state-based control flow and complex data
- support for parallel, concurrent systems
- composing complex models from simple models
- non-determinism, uncertainty
- abstraction, under-specification

But ....
- research prototype
- poor usability

Tool
- on-line MBT tool

Current Research
- scalability
- test selection

Under the hood
- powerful constraint/SMT solvers (Z3, CVC4)
- well-defined semantics and algorithms
- ioco testing theory for symbolic transition systems
- algebraic data-type definitions
TorXakis Model

```plaintext
FUNCTION gcd ( a, b :: Int ) :: Int
    IF a == b THEN a
    ELSE IF a > b THEN gcd ( a - b, b )
    ELSE gcd ( a, b - a )
ENDDEF

PROCEDURE processor [ Start :: JobData; Finish :: JobOut ] ( procnm :: Int )
    Start ? job :: JobData
    Finish ! JobOut { jobId(job) , procnm , gcd ( x(job), y(job) ) }
ENDDEF

PROCEDURE processors [ Start :: JobData; Finish :: JobOut ] ( procnm :: Int )
    processor [ Start, Finish ] ( procnm )
    ||
    ||    [[ procnm > 1 ]] => processor [ Start, Finish ] ( procnm-1 )
ENDDEF

PROCEDURE dispatcher [ Job, Dispatch :: JobData ] ()
    Job ? job :: JobData [[ isValidJob(job) ]]  
    Dispatch ! job
ENDDEF
```
Model-Based Testing

TorXakis Example:

Dispatcher-Processing System
Example: Dispatcher-Processing System

- Input jobs

- Dispatcher

- Processor 1
- Processor 2
- Processor 3
- Processor 4

- Processed jobs
Example: Dispatcher-Processing System
Example: Dispatcher-Processing System

Start job

Finish job

processor

Idle

Start

Processing

Finish

state transition system

DisPro01-processor.txs
Example: Dispatcher-Processing System

DisPro02-dispatch.txs
Example: Two Parallel Processors
Example: Two Parallel Processors

processor 1

processor 2

parallelism

Example: Two Parallel Processors
Example: Four Parallel Processors
Example: Four Parallel Processors

parallelism
Example: Dispatcher-Processing System

processors ::= processor(1) ||| processor(2) ||| processor(3) ||| processor(4)
Example: Dispatcher-Processing System

```
dispatch_procs ::= dispatcher |[ Start ]| processors
```

`DisPro03-procs.txs`
Example: Dispatcher-Processing System

dispatch_procs ::= HIDE [ Start ] IN dispatcher |[ Start ]| processors NI

DisPro04-hide.txs

Job

dispatcher

Start

Finish

Finish

Finish

Finish

Finish

Start

dispatcher

Job

abstraction

Start
Example: Dispatcher Processing System
Example: Dispatcher-Processing System

Inputs: Job1, Job2, Job3:

uncertainty
no unique expected result
Example: Dispatcher-Processing System

Inputs: Job1, Job2, Job3:

uncertainty
non-determinism
Example: Dispatcher-Processing System

```
FUNCDEF gcd ( a, b :: Int ) :: Int ::= 
   IF a == b THEN a 
   ELSE IF a > b THEN gcd ( a - b, b ) 
                    ELSE gcd ( a, b - a ) 
   FI 
FI

TYPEDEF JobData ::= JobData 
   { jobld :: Int 
     ; jobDescr :: String 
     ; x, y :: Int 
   }

state + data

Start job

processor

Finish job

FUNCDEF isValidJob ( jobdata :: JobData ) :: Bool ::= 
   jobdata.jobld > 0 
   ∧ strinre ( jobdata.jobDescr, REGEX('[A-Z][0-9]{2}[a-z]+') ) 
   ∧ . . . . .

Start

? job :: JobData 
[[ isValidJob(job) ]] 

Finish

! gcd ( job.x, job.y )
```
More Complex Data

Test data generation from XSD (XML) descriptions with constraints
Demo: Dispatcher-Processing System
Model-Based testing

Discussion
MBT by TNO-ESI

• Large, complex, systems-of-systems
• heterogeneous, evolving systems
• large, complex, connected
• complex data with combinatorial explosion
• customer variations
• parallelism, uncertainty
• compositionality
• specifications not always available
• Domain Specific Language
• virtual or simulated SUT
MBT for High-Tech Systems: TorXakis

MBT: next step in test automation

detecting more bugs faster and cheaper

- Automatic test generation
  + test execution + result analysis
- More, longer, and diversified test cases
  more variation in test flow and in test data
- Model is precise and consistent test basis
  unambiguous analysis of test results
- Test maintenance by maintaining models
  improved regression testing
- Expressing test coverage
  model coverage
customer profile coverage
Model-Based Testing
Discussion
Test Selection
Test Selection

• Exhaustiveness never achieved in practice

• Test selection = select subset of exhaustive test suite, to achieve confidence in quality of tested product
  – select best test cases capable of detecting failures
  – measure to what extent testing was exhaustive: coverage

• Optimization problem

  best possible testing ↔ within cost/time constraints
Testing and Quality

Cost

Quality-assurance costs

Remaining-defects costs

# Test cases
THE CARTOON TESTER ON HIS WAY TO EXPOQA
A CONVERSATION WITH A FELLOW PASSENGER TURNS TO TESTING EVEN BEFORE THE PLANE HAS SET OFF!

ARE YOU ON A BUSINESS TRIP?

SORT OF. I'M GOING TO A S/W TESTING CONFERENCE. I'M REALLY LOOKING FORWARD TO IT.

AH! SO YOU CAN TELL ME HOW FULLY TESTED THIS PLANE WAS. YOU SEE, I'M A BIT SCARED OF FLYING.

NOTHING IS EVER FULLY TESTED.

OH, REALLY? WHAT DO YOU MEAN?

WELL, IT'S IMPOSSIBLE TO IMAGINE EVERY TEST SCENARIO, NEVER MIND TESTING THEM ALL.

BUT THAT CAN'T BE TRUE.

OK, DO YOU REALLY THINK THIS PLANE WAS TESTED WITH TODAY'S EXACT WEATHER CONDITIONS WITH THE WEIGHT OF PASSENGERS AND LUGGAGE? I DOUBT IT. AND THAT'S NOT ALL...

STOP THIS PLANE! I WANT TO GET OUT!

DID I SAY SOMETHING WRONG?

Andy Glover cartoontester.blogspot.com Copyright 2010
Test Selection: Approaches

1. random

2. domain / application specific: test purposes, test goals, …

3. model / code based: coverage
   - usually structure based

\[ test: \ a! \ x? \]

100%

50%

transition coverage
Test Selection

Extra (domain) information required

• which test cases have high value?
• which errors are likely?
• which errors have high impact?
• what is the user / customer doing?
Model-Based Testing

Discussion

*How to Get these D... Models*
Models

- Everybody wants models
- Doing nice things with models
  - Model-based testing, model checking, simulation, ..... 
- How to get these models?
  - in particular for: legacy, third-party, out-sourced, off-the-shelf, ..... components
- Does the model correspond with the real system?
Testing: Model-Based Testing

model-based test generation

model

system

test execution

pass fail

Testing Diagram:

- Model-based test generation
- Model
- System
- Test execution
- Pass
- Fail
Test-Based Modeling

Model Learner

test execution

model

system

pass fail
Test-Based Modeling: Research

Automatically learning a model of the behavior of a system from observations made with testing

- test-based modeling
- automata learning
- black-box reverse engineering
- observation-based modeling
- behavior capture and test
- grammatical inference
Learned Model of OCE Printer Module
MBT for High-Tech Embedded Systems

MBT:

• Is it the promising future of software testing?

• Can we do without it?

• If not MBT, what then?