Conformiq QTRONIC

The Leading Model Driven Testing Tool

Kimmo Nupponen
Lead Developer
Conformiq Software Ltd

The Problem

- How to **design** and **create** test cases
  - for automatic execution...
  - ... e.g. in **TCL**, **Java**, **TTCN-3**, **C++**, *or...*
  - for manual testers (test plans)
- Manual test case design takes time...
- ...and creates risks!

Our Solution

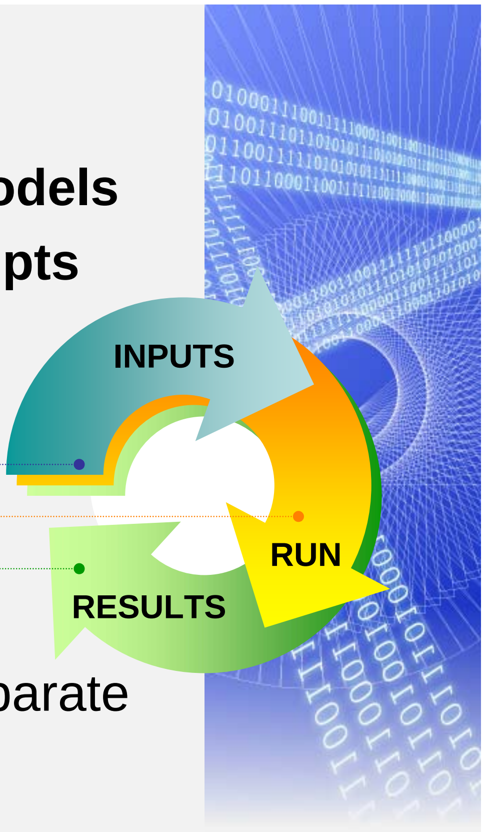
- Our product derives test cases automatically from **functional models**...
- generating also test **data**, **time**, and **expected results** (**test oracles**)...
- using well-established heuristics like model-level **branch coverage** or **boundary value analysis**...
- and state-of-the-art algorithms including **symbolic state space analysis**, **constraint solving** and **combinatorial optimization**

QTRONIC™

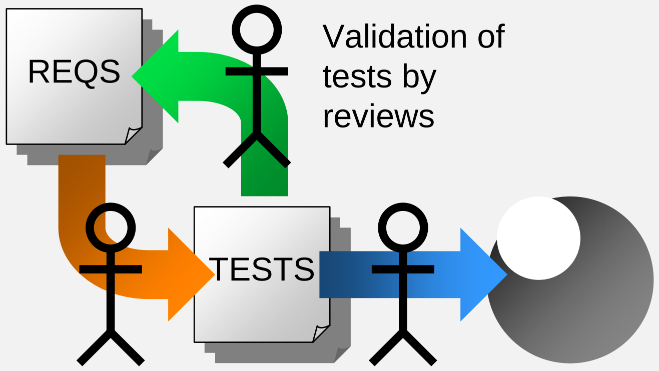
- Our commercial tool for model **driven** testing, i.e. test automation that is really fully driven by **system models**
- Philosophy:
  - User must be able to focus on the **correct behavior of the system**, not on **how it should be tested**
  - Use the reference model **as is**
  - Give **full modeling power** to the user with classes, time, data, concurrency…

Model Driven Testing

- Automatic testing **driven by models**
- **No manually created test scripts needed**
- Provides automatic
  - ✓ **Test input generation**
  - ✓ **Test execution**
  - ✓ **Test result evaluation**
- **Eliminates costs and risks** of developing and maintaining separate test scripts



Traditional Test Design

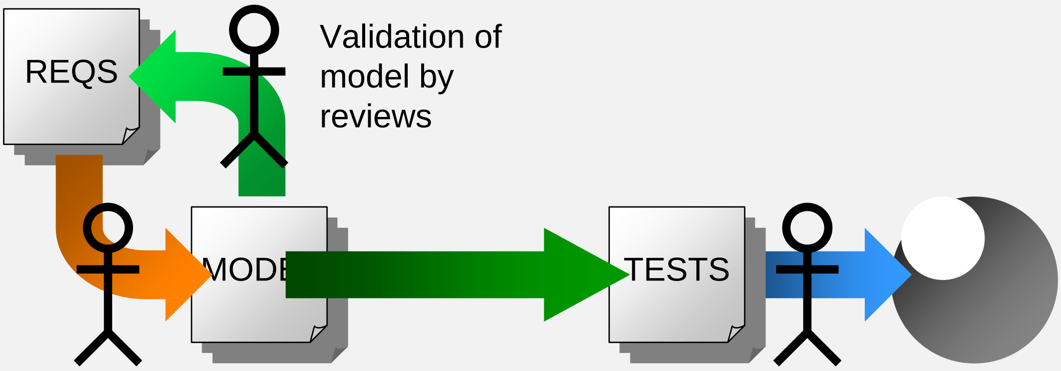


Validation of tests by reviews

“Test designer” creates tests based on requirements

“Test engineer” either runs the tests or deploys them in an automatic test execution framework

Qtronic Driven Test Design



Validation of model by reviews

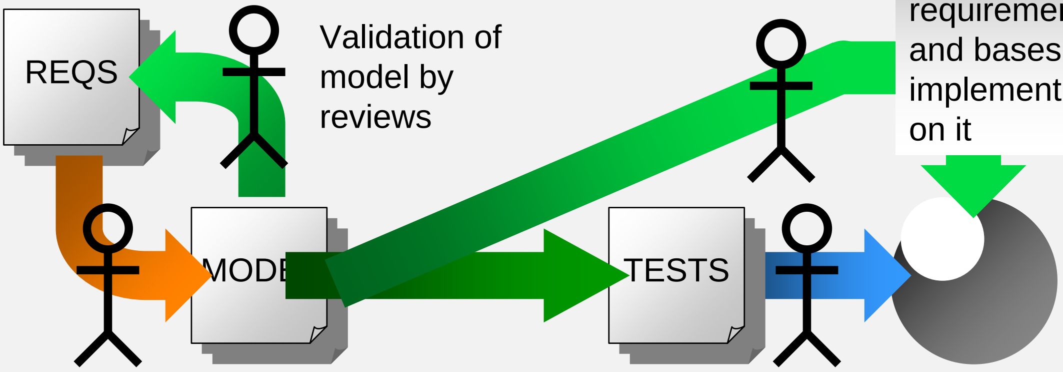
“Model driven test designer” creates a model based on requirements

Qtronic generates tests from the model

“Test engineer” either runs the tests or deploys them in an automatic test execution framework

Qtronic-Based TDD Process

“Developer” uses the model itself as functional requirements and bases implementation on it



Validation of model by reviews

“Model driven requirements engineer” creates a model based on high-level requirements

Qtronic generates tests from the model

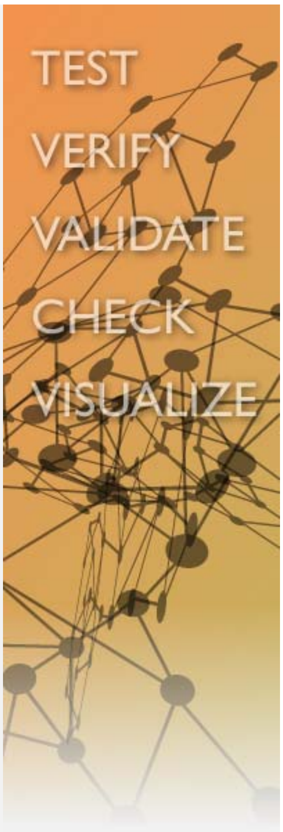
“Test engineer” either runs the tests or deploys them in an automatic test execution framework

What Qtronic Does For You

1. Creates comprehensive test cases
2. Creates executable test suites
3. Selects, runs and analyzes tests

How It Helps You

- ✓ Removes risk of defective tests
- ✓ Reduces costs of test design
- ✓ Improves test coverage
- ✓ Derives “test oracles” automatically
- ✓ Eases maintenance of test suites
- ✓ Makes test documentation more understandable



Logic of MDT

- The model describes the expected behavior of the SUT as an **open system**
- **The MDT tool synthesizes an environment that drives the real SUT in order to check that it works as the model predicts**
- Model driven testing heuristics are used to make sure that all important functionality of the model is exercised

Properties of MDT Models

- **Functional and executable**
  - Describe how the **system should work**
  - Could in theory be executed
  - Like abstract **reference implementation**
  - Not only e.g. **class diagrams**
- **System oriented**
  - Models are about the system, not about how to test it
- **More abstract than system**
  - Smaller and more compact than implementation
- **Linked to requirements**
  - Models can link behavior to higher-level requirements for traceability

Reference Implementation

- The model is basically an **abstract reference implementation** of the SUT, because it
  - is executable
  - describes the behavior of the SUT (albeit generally on a higher level of abstraction)
- As a matter of fact, the SUT model can be often **simulated**
- An “axiom” of model driven testing: **Tests generated from a model never fail when executed against a simulation of the same model**

Models

- Extended Java/C#
- *Optional* UML state charts
- Created in
  - Any text editor (textual parts)
  - Qtronic Modeler (UML)
  - Third party UML tool

Supported Constructs

- Full **data** (strings, numbers, records, classes, arrays…)
- Full **time** (timeouts, dynamic timeouts…)
- Full **control structures** (methods, dynamic polymorphism…)
- Full **concurrency** (multiple Java threads in model, ITC primitives…)
- Full **Java** + templates, macros, record values, type inference…

Online and Offline Testing

- **Online testing** = test steps are selected, run and the results checked in parallel
- **Offline testing** = tests are created as test cases or scripts that can be executed later (oracles included in the generated cases)

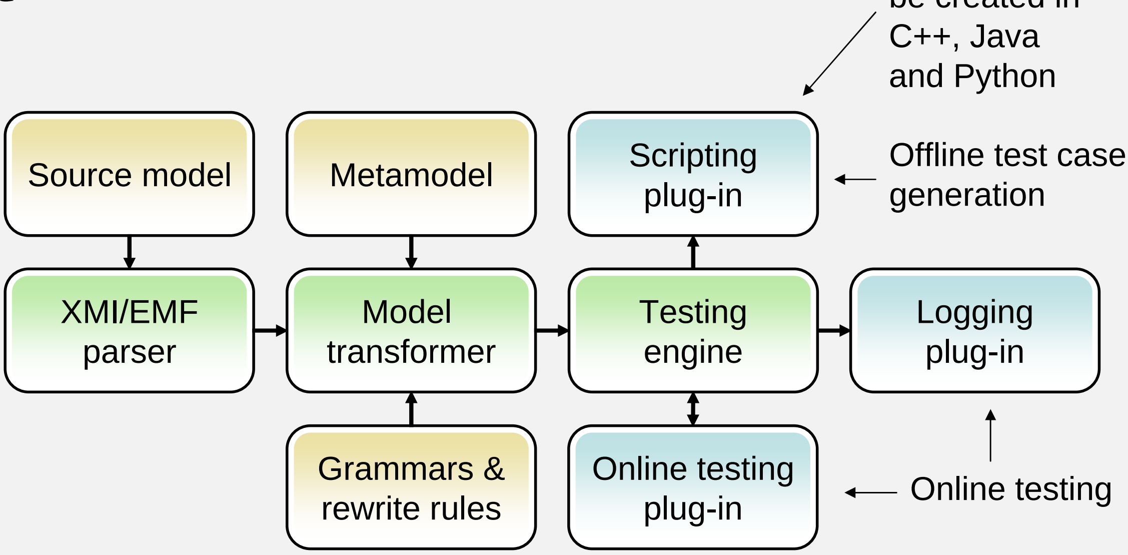
Key Differences

	Online testing	**Offline testing**
Supports **nondeterministic** models (systems that can make independent choices that have externally visible implications)	YES	NO*
Relative length of the **update model ⇔ test system** cycle	SMALL	LARGE
Relative computational burden at **test execution** time	HIGH	LOW
Tests can be **inspected, analyzed and changed manually** before execution	NO*	YES
Immediate **repeatability of tests**	NO*	YES

Coverage Criteria

- **Transition coverage**
  - Cover all transitions in all state charts
- **State coverage**
  - Cover all states in all state charts
- **Branch coverage**
  - For every **if** and **while** loop, cover both the positive and the negative branch
- **Condition coverage**
  - For every **x and y** and **x or y**, cover combinations of the truth values of x and y (but taking short-circuited evaluation into account)
- **Requirements coverage**
  - For every requirement link in the model, cover the link
- **Boundary value pattern**
  - For every test **x < y**, cover cases x = y – 1; x < y – 1; x = y; and x > y
  - Other comparators work analogously

Qtronic Architecture



Plug-ins can be created in C++, Java and Python

Offline test case generation

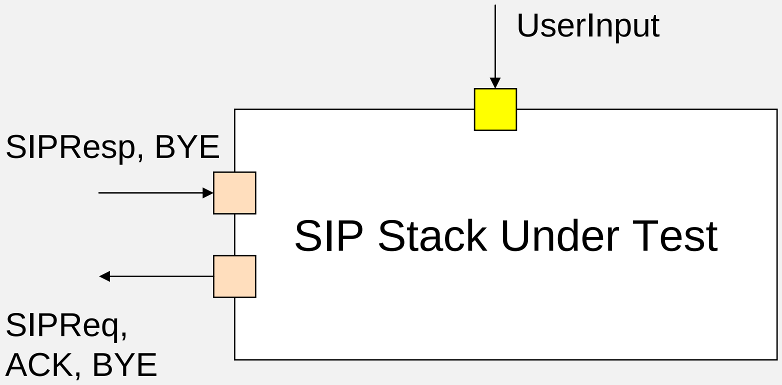
Online testing

Demonstration

- High-level testing of SIP/UAC session initiation and tear down
  - Implementation: Sofia SIP stack from NRC
- **SIP** = Session Initiation Protocol
- **UAC** = User Agent Client

- Human readable test cases
- TTCN-3 generation
- Online testing

Testing Architecture



🟨 Control / application interface (C API)

🟧 Peer to peer interface (SIP packets over UDP)

Further Pointers

- **Books:**
  - **Practical Model-Based Testing: a Tools Approach**
  - Written by Mark Utting and Bruno Legeard
- **Some companies (in alphabetical order)**:
  - Conformiq Software
    - **Conformiq Qtronic** for offline and online test generation
  - Leirios
    - **Leirios Test Generator** for offline test generation
  - Microsoft Research
    - Has published freely available **SpecExplorer** tool for model-based testing
  - Reactive Systems
    - **Reactis** for offline test generation, especially in the controller domain
  - Telelogic I-Logix
    - **Rhapsody ATG** generates tests for Rhapsody models

Questions, comments?

- Kimmo Nupponen
- Lead Developer
- Email: Kimmo.Nupponen@conformiq.com
- Phone: +358 40 564 7805