

Introduction to the TEMA Toolset

Tommi Takala

Department of Software Systems

Tampere University of Technology, Finland

tommi.takala@tut.fi



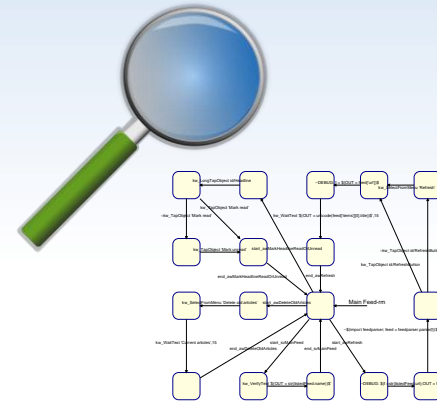
Contents

1. Toolset Architecture
 1. Modeling
 2. Test Design
 3. Test Generation
 4. Test Execution
 5. Test Debugging
2. Experiences from case studies
3. Demo

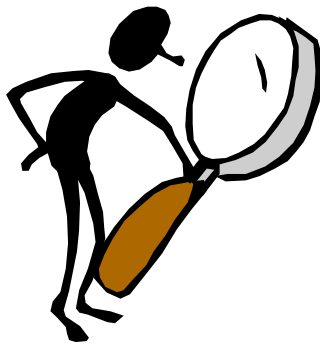
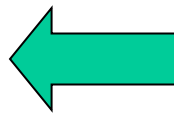
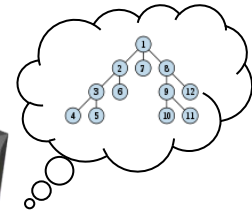
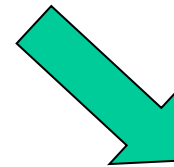


What is the TEMA Toolset?

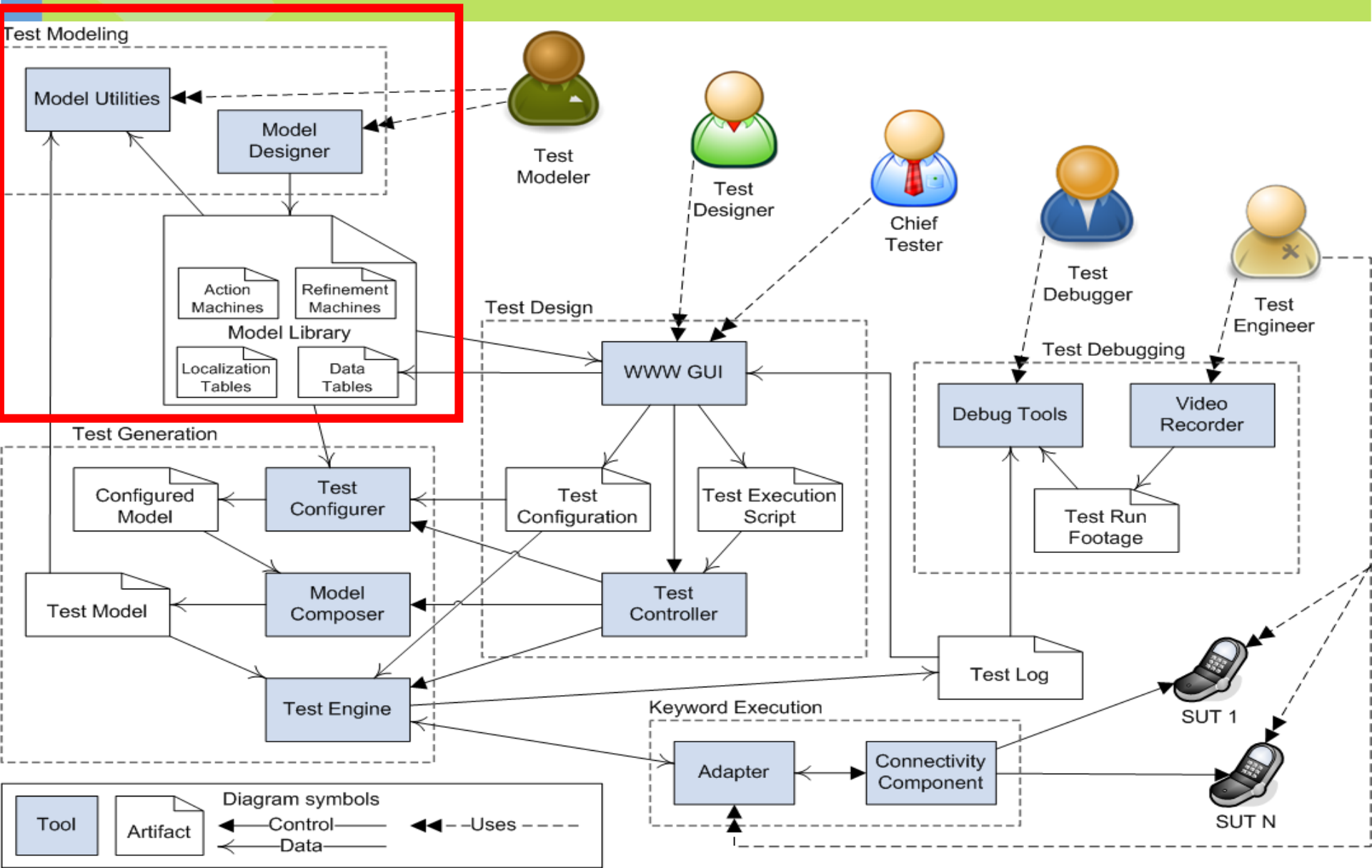
- Set of model-based GUI testing tools
- Includes tools for:
 - Modeling (design and validation)
 - Designing test objectives
 - Test generation
 - Test execution
 - Test debugging
- Released under the MIT licence



Cover all states!



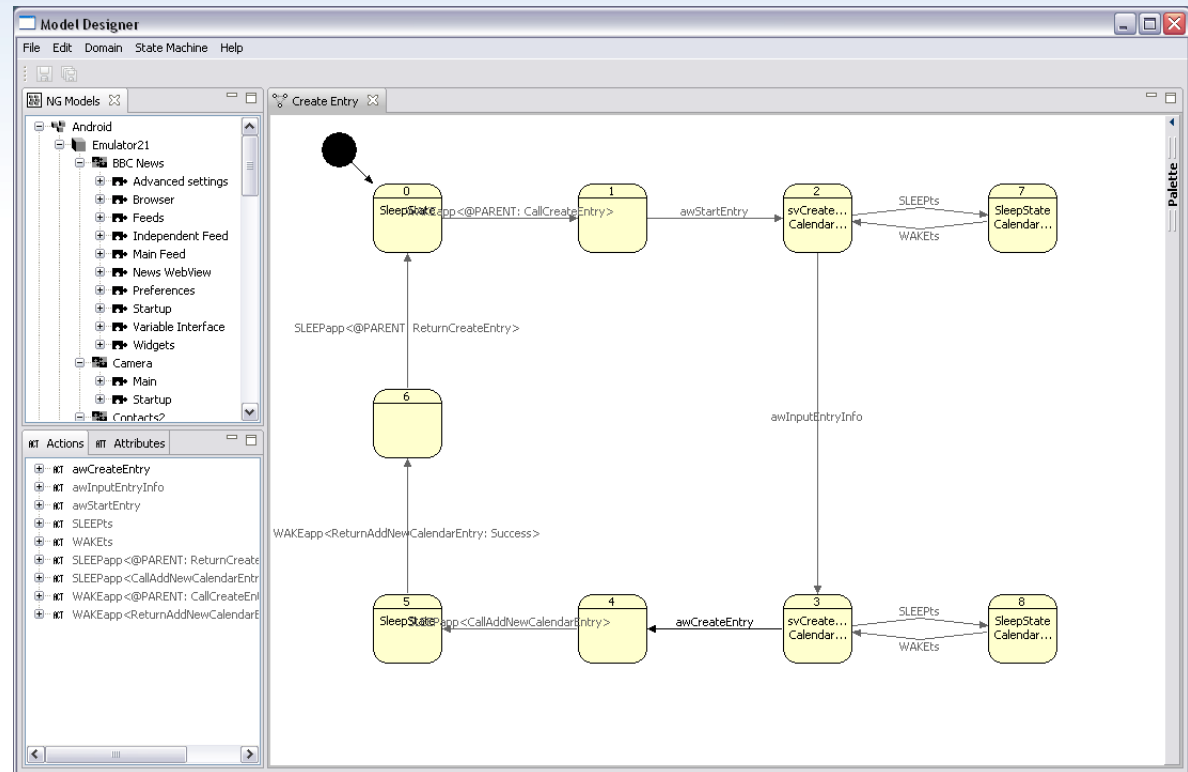
Test Modeling



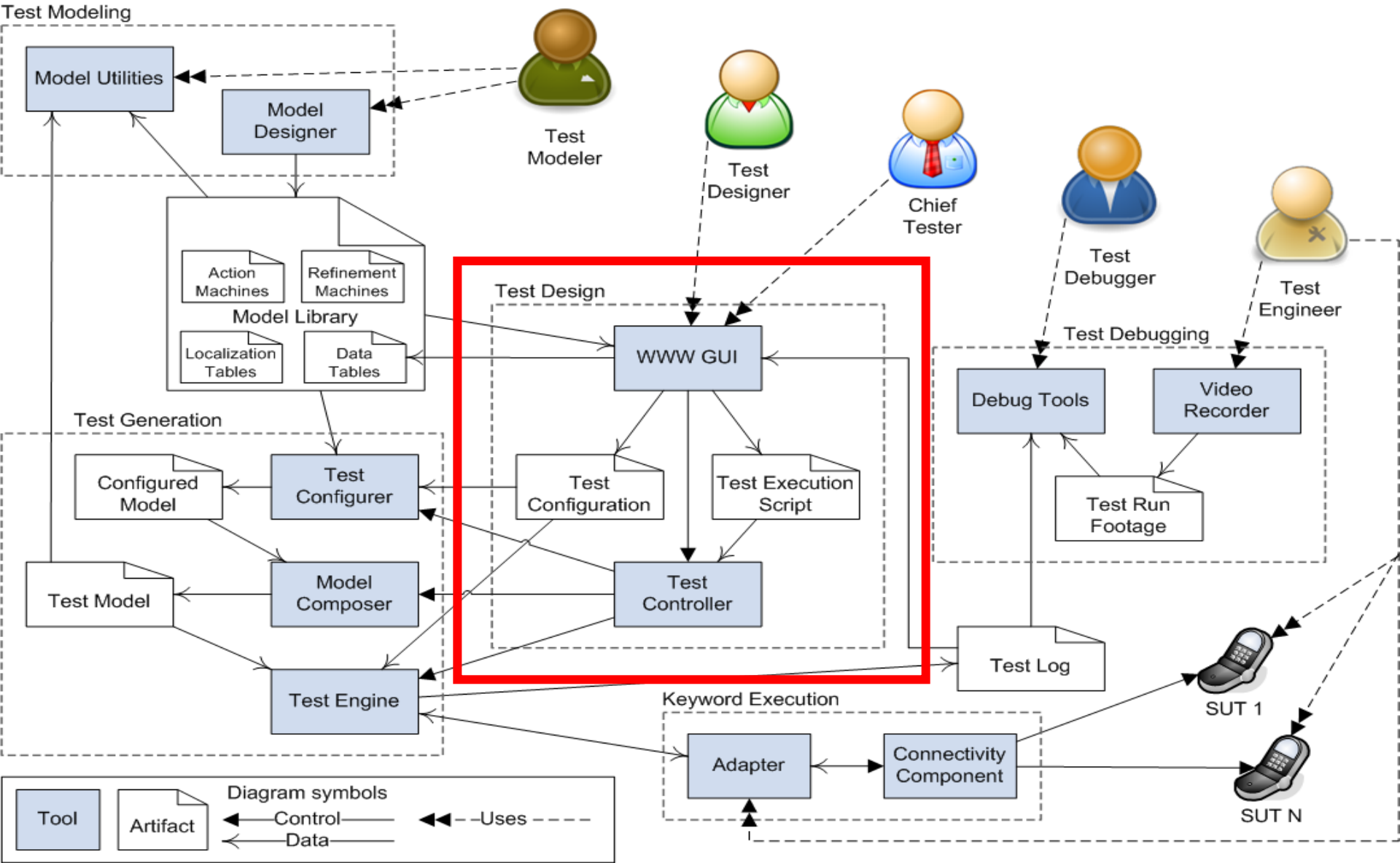
Test Modeling

Includes:

- Model Designer for developing state machine models
- TEMA Model features:
 - Division to small components
 - Two-level models
 - Concurrency
 - Embedded Python code
- Model simulation and validation tools for ensuring model correctness
- Conversion tools between model-formalisms



Test Design

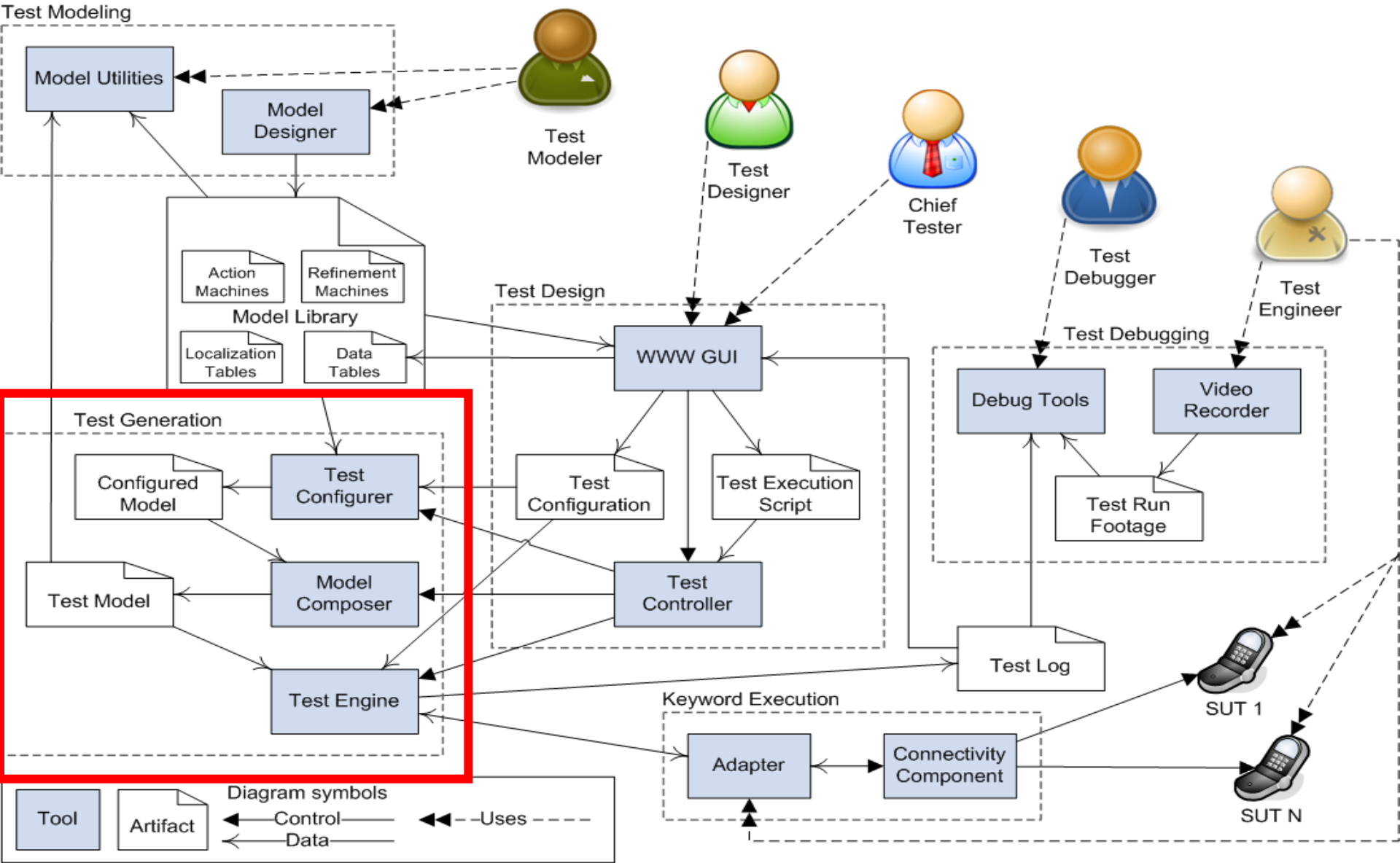


Test Design

- Since testers don't want to directly deal with models or test generation algorithms, we have abstracted them out in our web GUI
- TEMA web GUI is a testers' interface with the test server, used for designing and managing test configurations, running and tracking actual tests, and managing test model packages
- The test generation is abstracted behind different testing modes:
 - Bug-hunting test
 - Use case test
 - Smoke test
 - Basic coverage test
- This all boils down to allowing testers to just choose what they want to test and what physical device they want to run their tests on

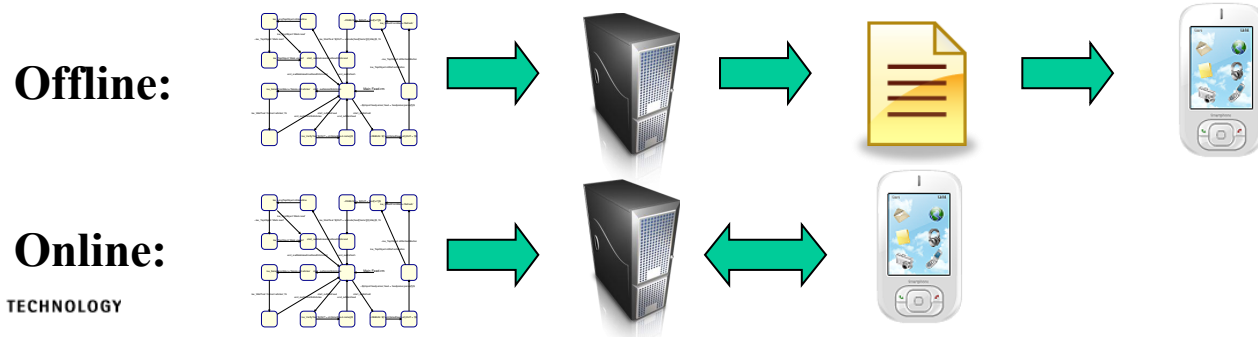


Test Generation

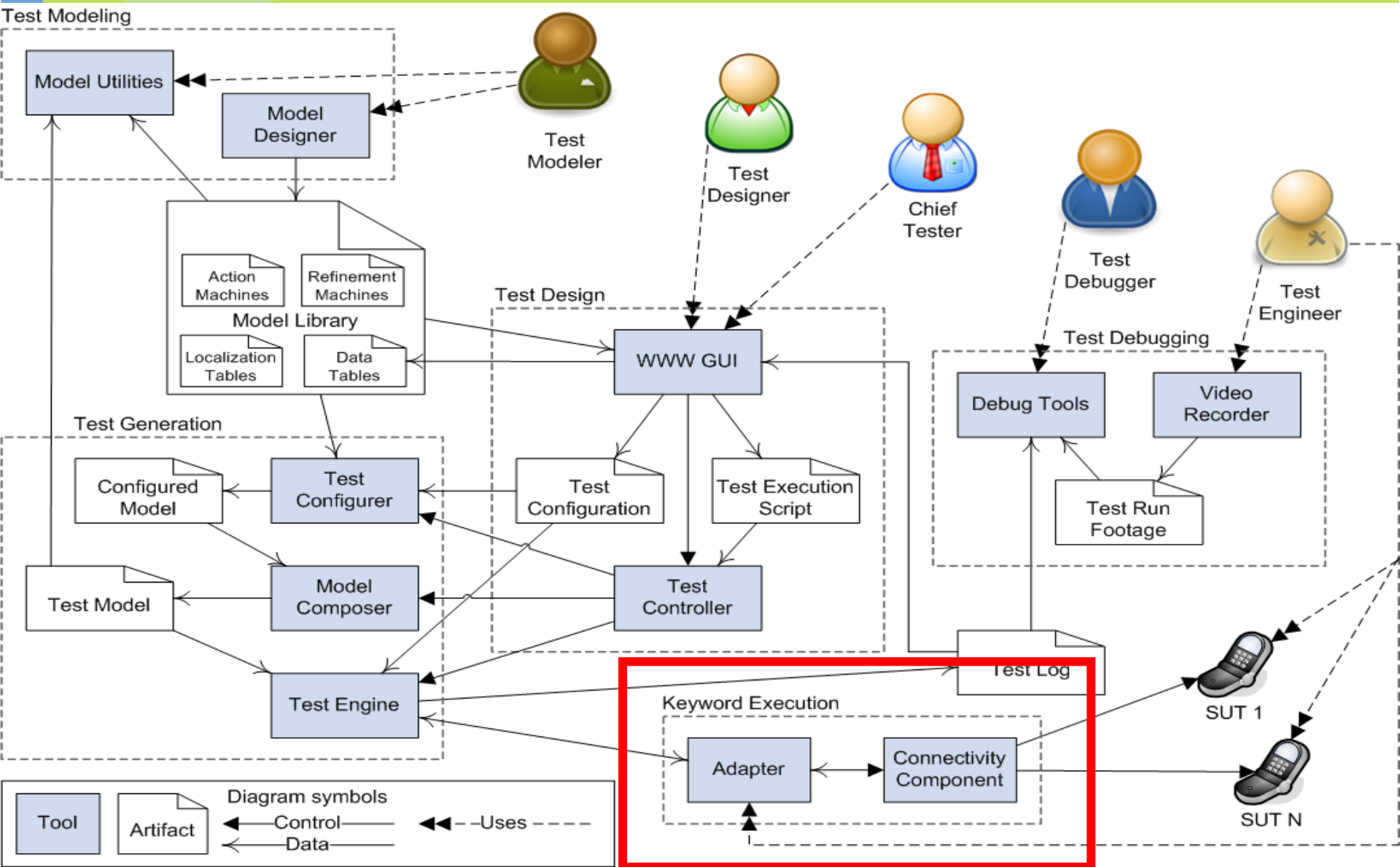


Test Generation

- Composes a full test model from the model components on the fly during execution
- We use various graph search algorithms, such as random and interaction-based algorithms
- The end user is only concerned with Web GUI modes and their parameters
- The Web GUI is not mandatory, advanced users can use the test generation engine with a command line interface
 - Convenient for scripting purposes
- Test generation is based on the **online** approach:



Keyword Execution

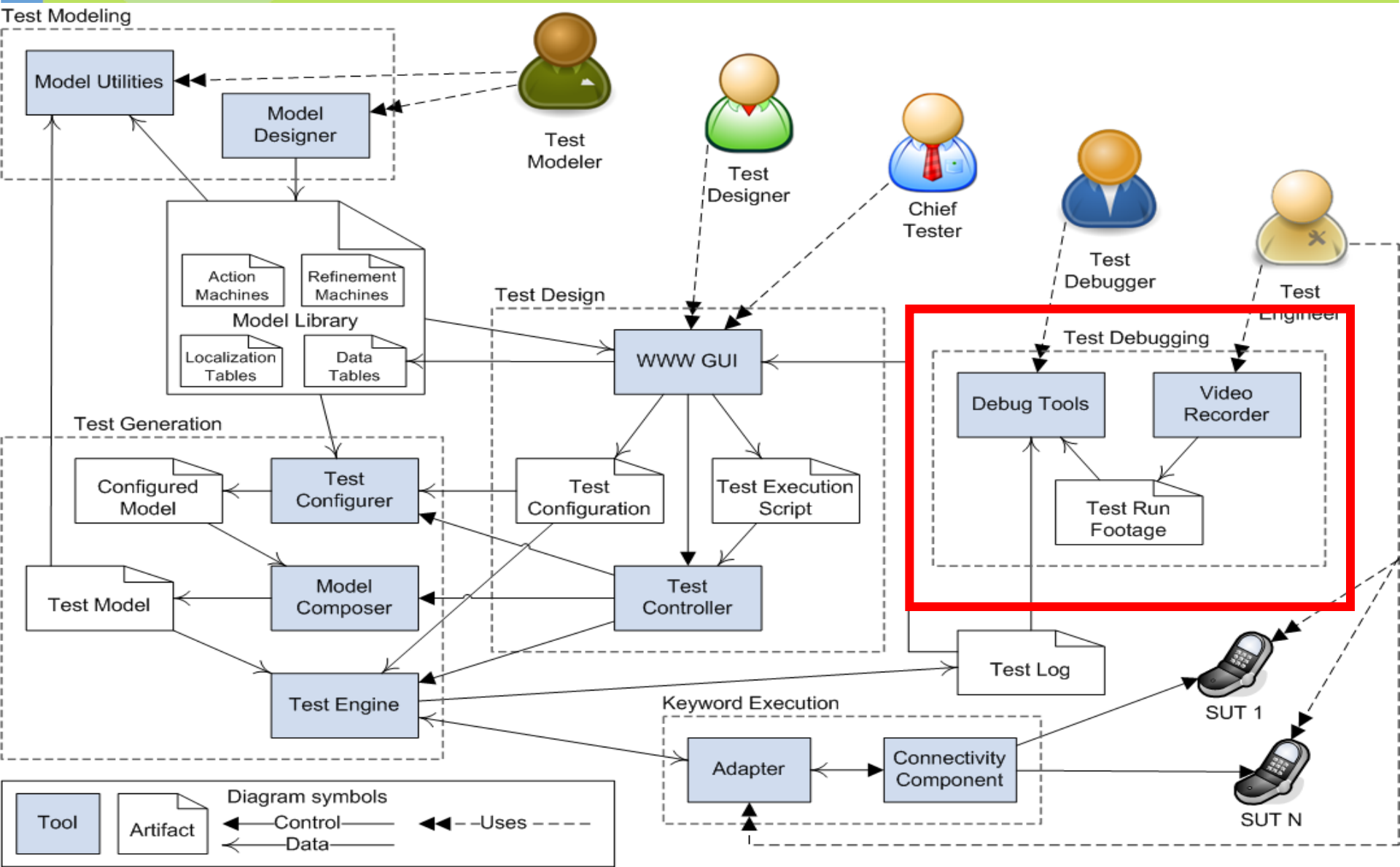


Keyword execution

- Test engine provides a socket interface for test automation clients.
- Test execution is based on keywords
 - PressKey, TypeText, ClickObject, VerifyText, ...
- After each keyword has been executed, the test execution provides the result (true/false) back to the test engine, following the online approach
- Existing public test automation support for the Android emulator, Linux (Gnome), QT and Robot framework libraries (Java Swing, Selenium)



Test Debugging



Test Debugging

- In long-period random testing, it can be hard to interpret what exactly causes a bug to occur.
- Video recorder and subtitle generation from test log
- Tools for searching a minimized sequence of actions that causes the bug
 - For example, a test run about 1850 keywords long that found a bug was shortened to about 100 keywords long



Domains Conquered

- We have been primarily focusing on mobile platforms, but our approach is also suited to desktop applications
- Our model library presently holds models for the following domains:
 - S60 (No public test automation support)
 - Mobile Linux
 - Android
 - Java Swing
- In addition, public test automation support is available for QT and Robot framework libraries.



Case: S60 (Project Starting Point)

- Built-in applications in S60 smartphones, such as Gallery, Music Player, Flash Player, Notes, Voice Recorder, Contacts and Messaging
- Keyword execution using proprietary and commercial test automation tools
 - Optical character recognition was used for verifications, which caused some reliability and maintenance issues
- 21 defects of different severities and priorities were found
 - Some of these defects existed in more than one smartphone model
 - The most severe of the defects caused the phone to hang with “System error” message on the display
 - About two thirds of the defects were discovered while modeling (reverse engineering), and the remaining third by execution (dynamic testing)
 - Most of the defects had already been previously found in traditional testing (both manual and automatic test execution), but they had not been fixed for some reason
 - However, there were also some that were totally new
 - Many of the defects were related to **concurrency issues**: performing some multimedia-related functionality in one application and then switching to another application caused unexpected behavior in some circumstances
 - In addition to defects found in applications, some were found in test automation tools, which was considered rather surprising, as these tools were quite mature



Case: Mobile Linux

- Media player application by Ixonos
 - New modeling challenge: real-time requirements
 - Playing videos, fast-forwarding, rewinding, pausing...
 - Although it was difficult, real-time support was eventually accomplished
- Keyword execution using Linux accessibility features
 - API access to GUI components
 - Easier and more reliable than in S60 case
 - Works on the GTK environment (Gnome)
- Some minor bugs were found (both during modeling and execution)



Case: Android Phone

- Messaging, Contacts and Calendar applications
 - High-level models created for S60 were reused
 - Calendar was modeled with ATS4 AppModel and converted to TEMA models with an automatic converter
- Keyword execution was based on A-Tool by Symbio
- Optical character recognition was implemented with MS Office Imaging, which could have been more reliable
- Some bugs were found (both during modeling and execution)



Case: Android revisited

- BBC News application
 - RSS reader optimized for BBC news feeds
 - About 300 000 users (downloads)
- Self-made test automation for the emulator
 - Based on API access -> improved reliability
- Defects found so far:
 - 14 in total
 - 8 found during modeling
 - 6 found during execution
 - 2 of the bugs cause the application to crash
- Some Android base applications were also modeled:
Gallery, Camera



Lessons learned

- Modeling typically uncovers more bugs and quirks than test execution itself
 - Reverse-engineering vs. use of system models
 - Lack of precise enough GUI specifications
 - Agile trend hasn't made matters easier....
- It is possible to find bugs in already well-tested applications
 - Mostly minor or cosmetic, but also serious (system errors, crashes etc.)
 - Otherwise hard-to-find concurrency-related bugs have been found (major upside of our approach)
- A talented student was able to create the first version of the S60 model library in 2 months (+1 month for debugging & maintenance)
- Automatic GUI testing requires mature test automation support from the domain



DEMO



Thank you!

TEMA TOOLSET NOW AVAILABLE!

<http://tema.cs.tut.fi/>

Acknowledgements:

Tekes, Nokia, Ixonos, Symbio, Cybercom Plenware, F-Secure,
Qentinel, Prove Expertise, Academy of Finland (grant #121012)

