Making Model-Based Testing More Agile: a Use Case Driven Approach

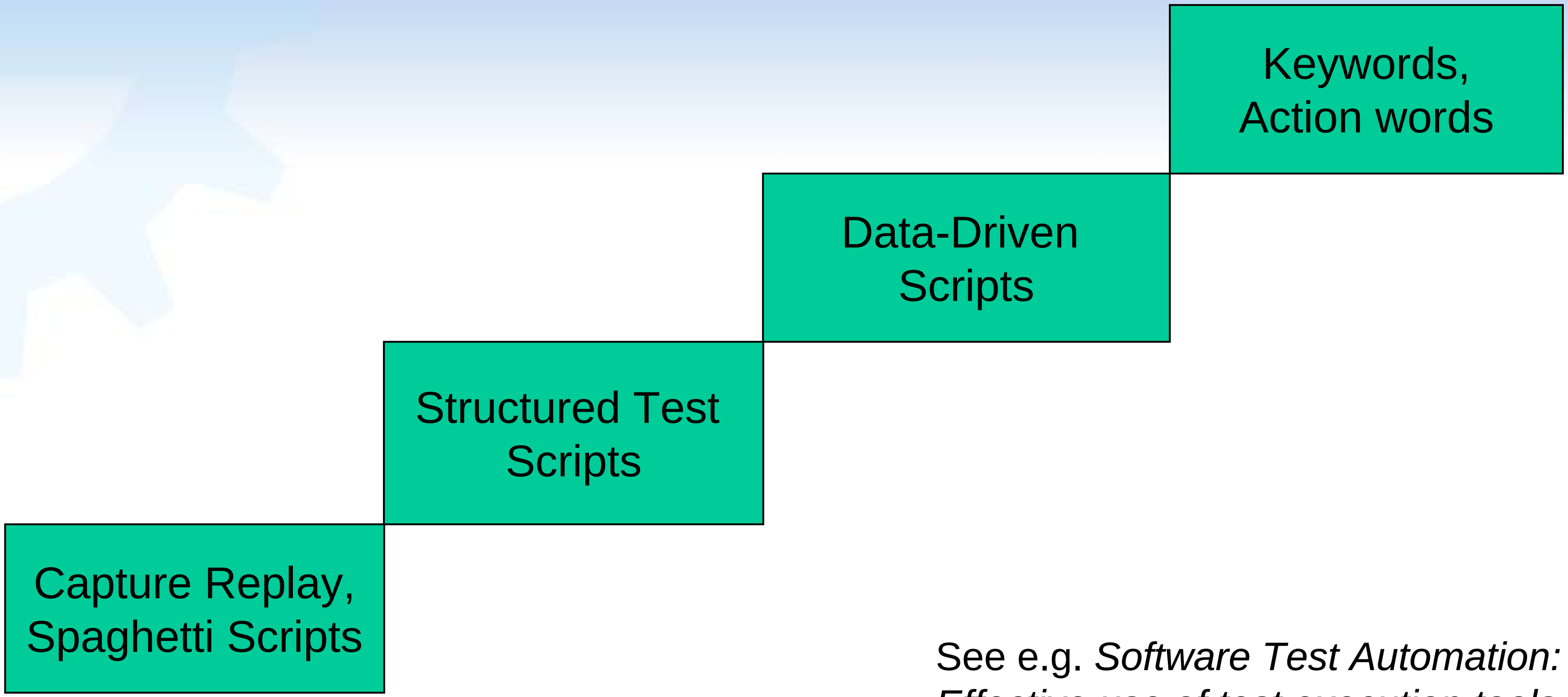
Oct 23–26, 2006 @ Haifa Verification Conference, Israel

Mika Katara and Antti Kervinen
Institute of Software Systems
Tampere University of Technology, Finland

Agenda

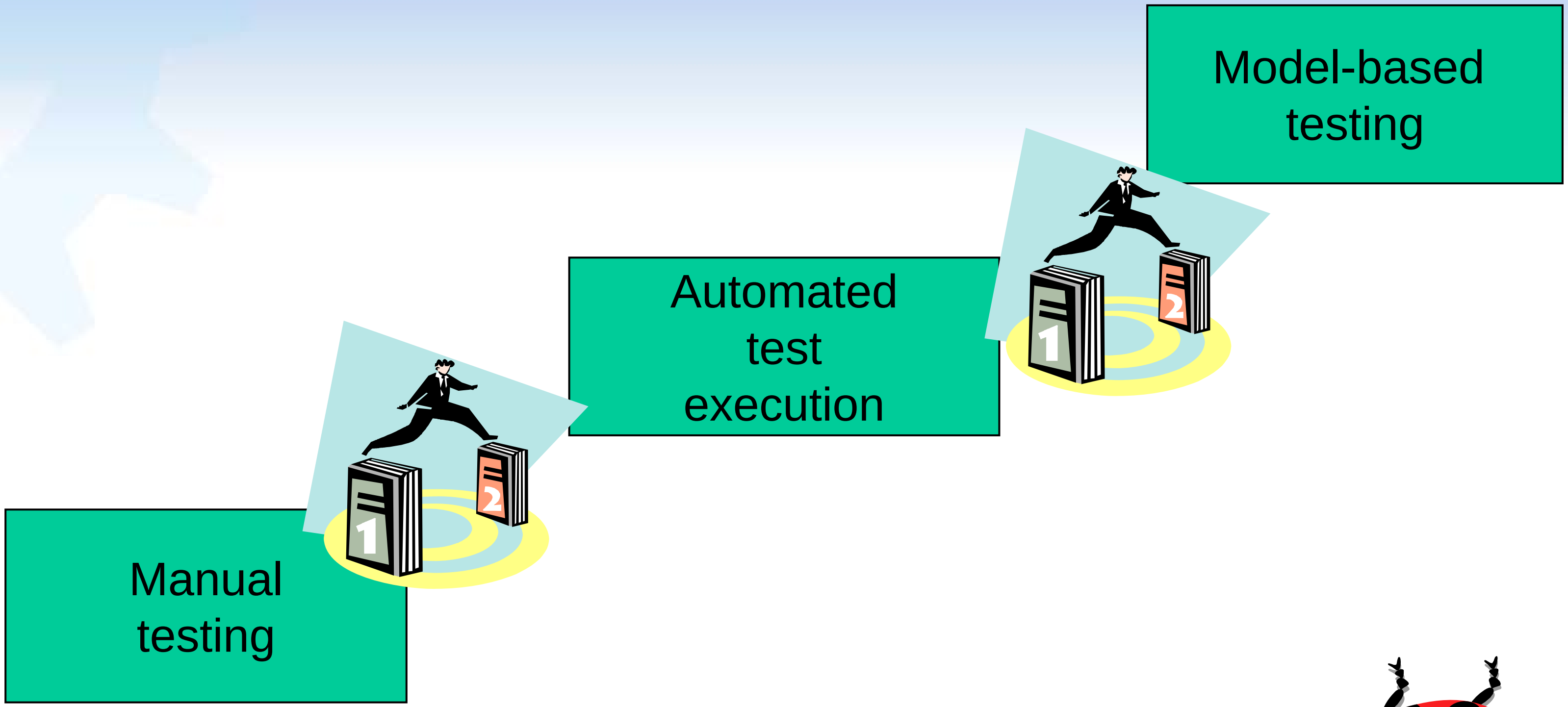
1. Background
2. Motivation
3. Use-case driven MBT
4. Coverage language
5. Test generation algorithm

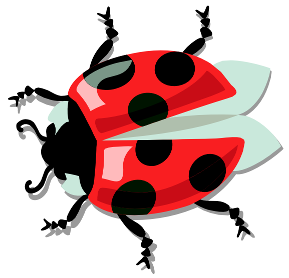
Background: Generations of GUI Test Automation



See e.g. *Software Test Automation: Effective use of test execution tools*
By Mark Fewster and Dorothy Graham, Addison Wesley, 1999.

… and the Future?





TEMA Project: Academic & Industrial Collaboration

Tampere University of Technology/Institute of Software Systems
TEKES, the Finnish Funding Agency for Technology and Innovation
Nokia
Conformiq Software
F-Secure
Plenware Group
Mercury Interactive

A general goal of the project:
Industrial deployment of MBT in GUI testing of Symbian S60 smart phones
- ”Nokia alone has cumulatively shipped 50 million S60 enabled devices by end of February 2006” (Source: www.s60.com)

Disclaimer: these slides represent the views of the presenter, not necessarily the views of the above or any other parties

Pros and Cons of Model-Based Testing

Pros:

- Higher level of **abstraction** helps to concentrate on the right things – details are hidden
  - Better chances when fighting against the increasing complexity
- Models (small ones) can be **visualized** easier than code – better comprehension
- Automatic test generation – better **coverage**
  - **Maintenance** should be easier also – not much studied subject

Cons:

- Based on reported industrial experiences **deployment** can be very challenging
- Modular treatment of **large models**
- ”Bubbles don’t crash” – easier to **ignore** real problems

Domain-Specific Test Modeling of Product Families

Domain-specific modeling prefers a stable domain

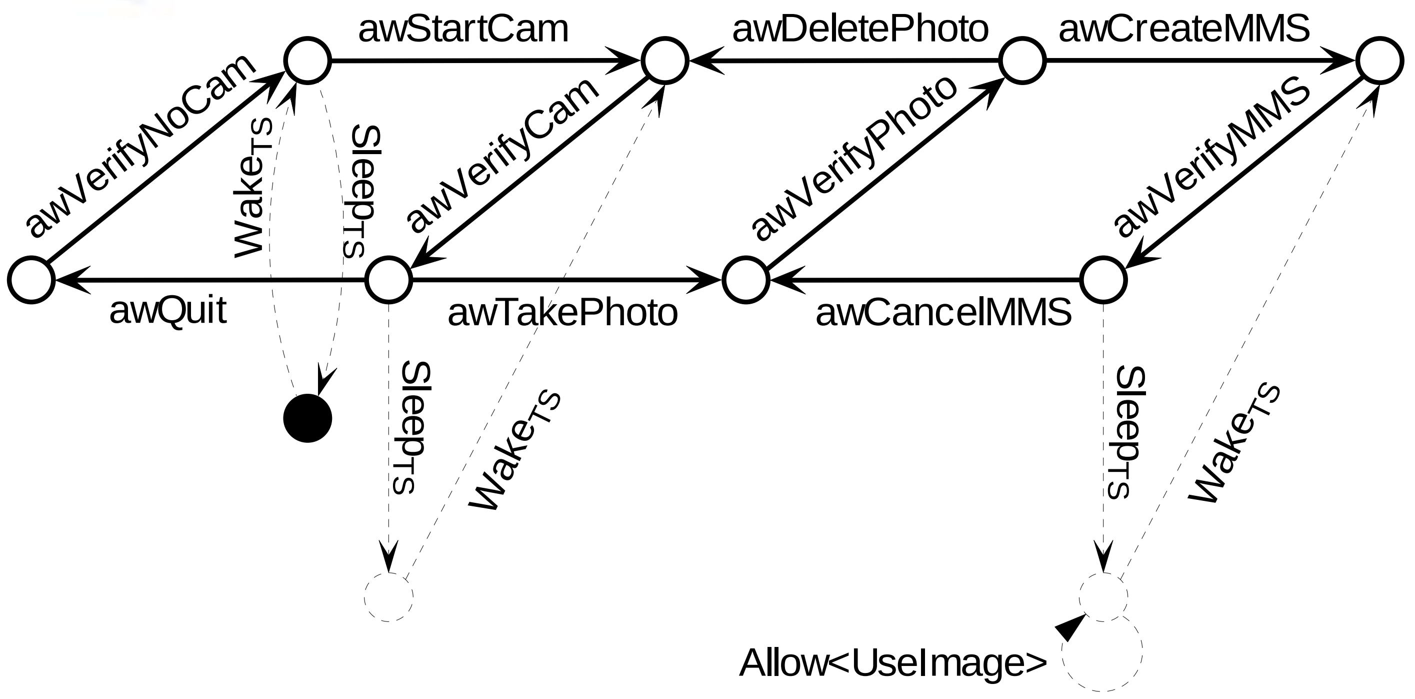
In a product family context there are some parts that stay the same and some things that change

Our context:

- Symbian OS, an operating system for smart phones
- The look & feel stays the same across the family of phones using S60 GUI platform on top of Symbian OS
- In our approach, the smart phones are tested through a GUI using commercial test automation software
- For this purpose, we have defined a domain-specific test modeling language based on action words and keywords

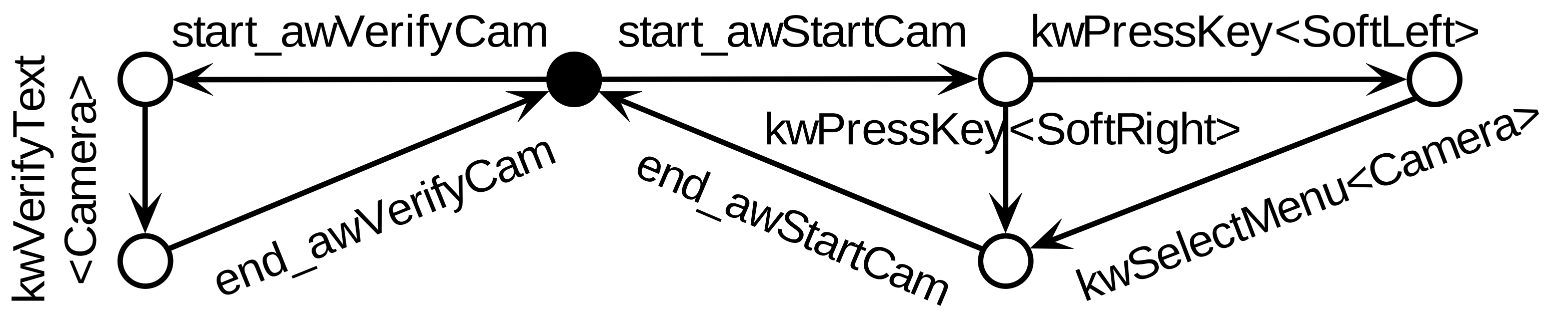
Example Action Machine

S60 Camera application, action word model

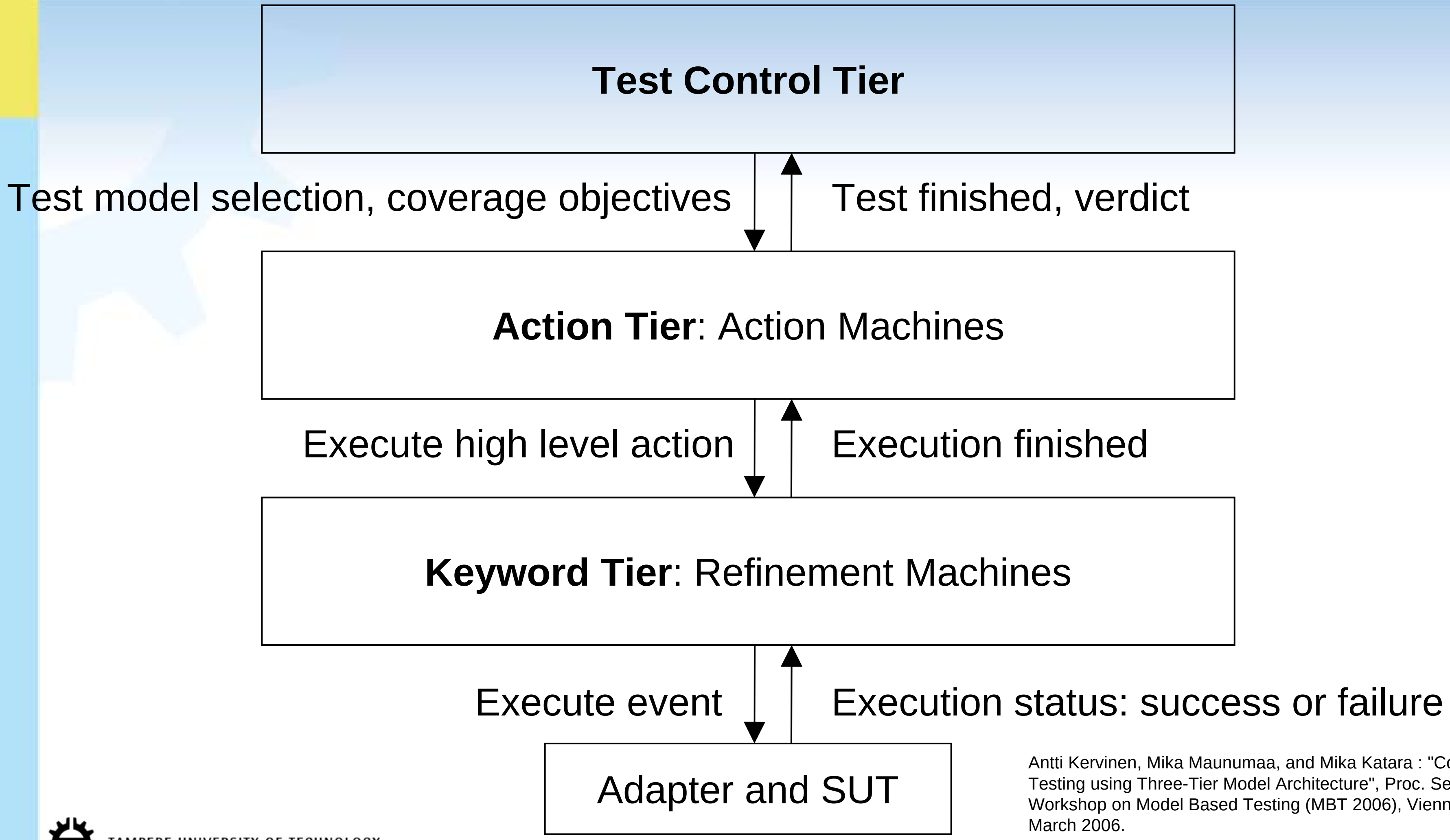


Example Refinement Machine

S60 Camera application, keyword model



TEMA 3-Tier Test Model Architecture


Test Control Tier
Test model selection, coverage objectives
Test finished, verdict
Action Tier: Action Machines
Execute high level action
Execution finished
Keyword Tier: Refinement Machines
Execute event
Execution status: success or failure
Adapter and SUT

Antti Kervinen, Mika Maunumaa, and Mika Katara : "Controlling Testing using Three-Tier Model Architecture", Proc. Second Workshop on Model Based Testing (MBT 2006), Vienna, Austria, March 2006.

Motivation: Why was this paper written?

Even with domain-specific modeling language and model recording it's very difficult to get testers to create test models

Comments from industrial partners:

”If you could disguise your MBT method as an agile approach, it would be much more easier to sell it inside our company.”

”Conventional test automation scripts can be directly derived from requirements, there is a clear mapping between the test and the requirements. MBT lacks this which makes it impractical.”

Experiences so far

Based on our experiences with the prototype implementation of our testing tools, the biggest obstacles in the practical use of our methodology concern the creation of the action word models

Moreover, even though the domain-specific language can help domain-experts without programming skills to build models, there is no clear relationship between the models created for testing, and other artifacts, especially the requirements and design documents

Another related question is about requirements coverage

“How can we know when some requirement has been covered by the generated tests?”

Different Testing Modes

We have initially identified three different testing "modes" that should be supported in an agile project:

- Firstly, smoke tests are needed for verifying that a build has been successful
  - However, we do not restrict to a static sequence of tests such as in conventional test automation
  - Instead, we may set limits on the duration the test and explore the test model on a breadth-first fashion within those limits
- Secondly, we need to be able to cover certain requirements
  - Goals are often set in terms of requirements coverage: for example, at least requirements R2 and R4 should be tested
- Thirdly, we would like to do serious bug hunting
  - In this case our primary motivation is not to cover certain requirements or to stop within few minutes; we try to find as many defects as possible
  - However, requirements can be used to guide the test generation also in this mode
  - Furthermore, the coverage data obtained in the previous test could be used to avoid retesting the same paths again

What is a Use Case?

Use cases are one of the most popular way to **capture requirements**

The definition of a use case varies in the literature

We will use an informal use case format, including three elements: **name**, **identifier** and the **basic course of action**, which is a sequence of event descriptions corresponding to the high-level interaction between the user and the system

We will use the following informal use case as a running example:

Name: Alice asks Bob to meet for lunch

Identifier: UC1

Basic course of action:

1. Alice sends Bob an SMS asking him to meet for lunch
2. Bob replies by sending a multimedia message containing a picture of a bumper taken at a traffic jam and a text ”I will call you when I’ll get there”
3. After a while, Bob calls to Alice, but she is talking to Carol so the line is busy
4. After a few minutes Alice calls back to Bob and they agree to meet at the cafeteria

All use cases are not equally important: In order to cope with this, some risk-based testing practices should be deployed

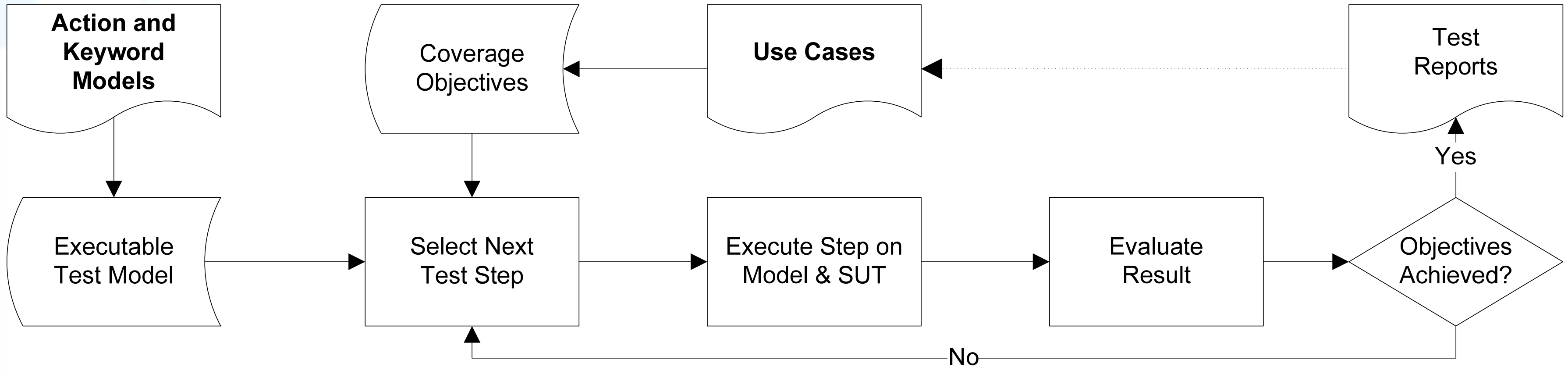
The use cases must be prioritized based on some formal risk analysis or customer intuition

Use Case Driven MBT

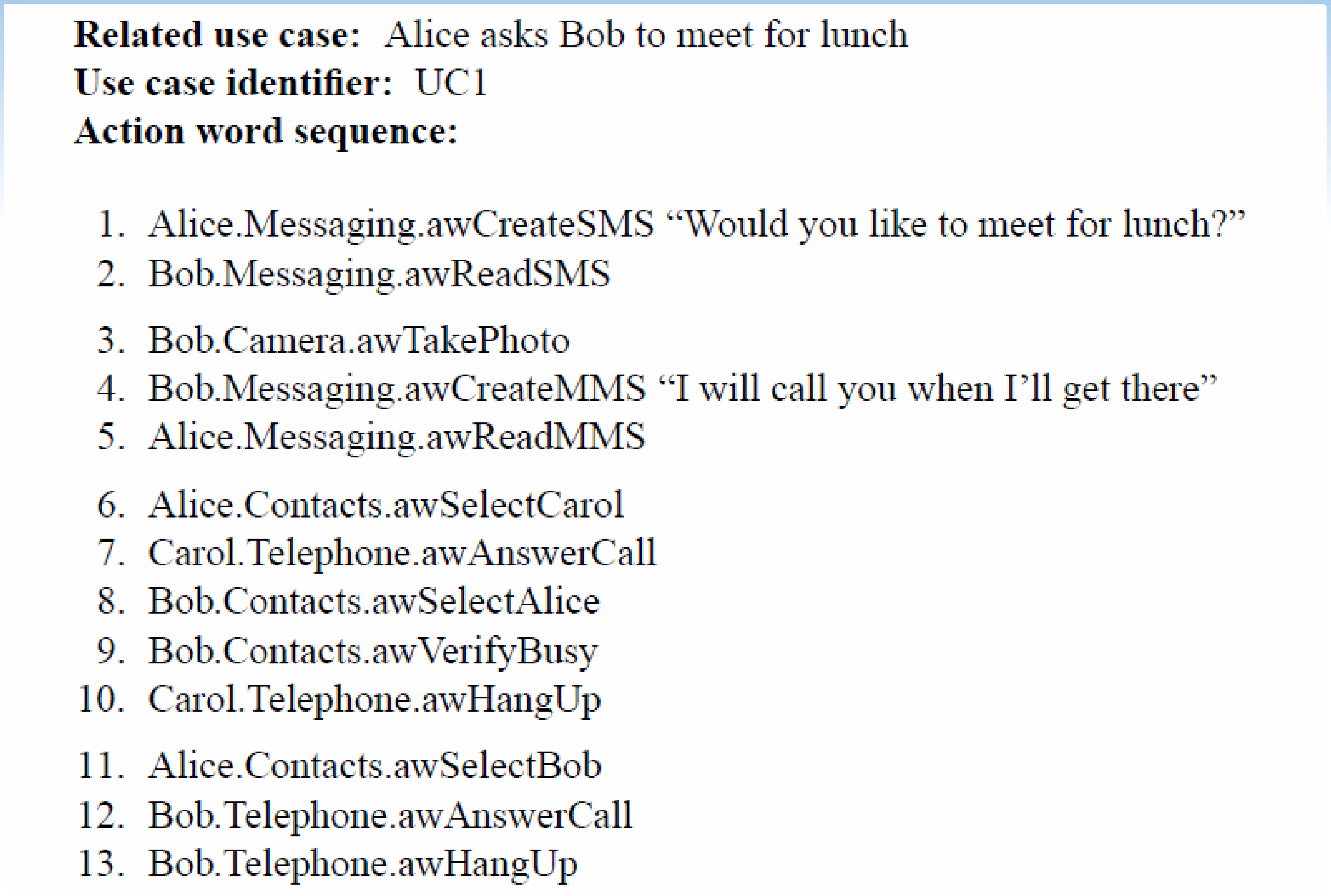
Since our domain is quite restricted, experts can handle the actual test modeling while testers define test objectives based on use cases

On the one hand, use cases should be familiar to most system level testers

On the other hand, use cases can help us to solve one re-occurring problem in model-based testing, i.e., how to restrict the set of generated tests



An Action Word Sequence Based on a Use Case

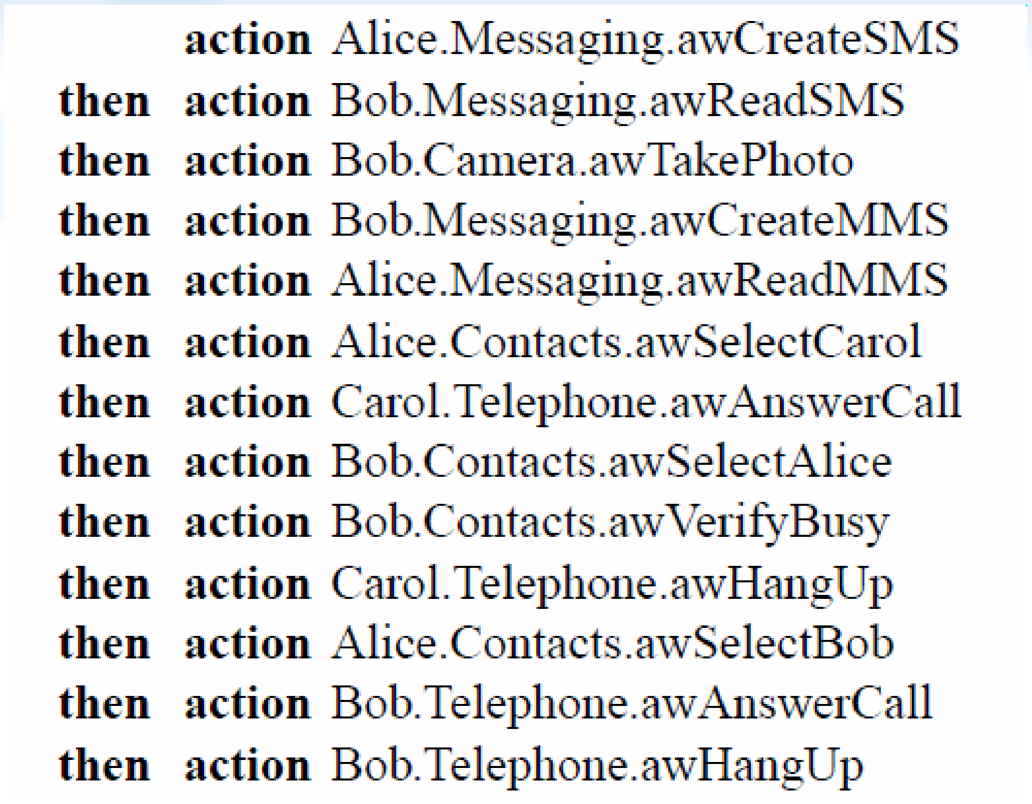


Related use case: Alice asks Bob to meet for lunch
Use case identifier: UC1
Action word sequence:

1. Alice.Messaging.awCreateSMS “Would you like to meet for lunch?”
2. Bob.Messaging.awReadSMS
3. Bob.Camera.awTakePhoto
4. Bob.Messaging.awCreateMMS “I will call you when I’ll get there”
5. Alice.Messaging.awReadMMS
6. Alice.Contacts.awSelectCarol
7. Carol.Telephone.awAnswerCall
8. Bob.Contacts.awSelectAlice
9. Bob.Contacts.awVerifyBusy
10. Carol.Telephone.awHangUp
11. Alice.Contacts.awSelectBob
12. Bob.Telephone.awAnswerCall
13. Bob.Telephone.awHangUp

Translation into a Sentence in the Coverage Language

Example use case:



```
     action Alice.Messaging.awCreateSMS
then action Bob.Messaging.awReadSMS
then action Bob.Camera.awTakePhoto
then action Bob.Messaging.awCreateMMS
then action Alice.Messaging.awReadMMS
then action Alice.Contacts.awSelectCarol
then action Carol.Telephone.awAnswerCall
then action Bob.Contacts.awSelectAlice
then action Bob.Contacts.awVerifyBusy
then action Carol.Telephone.awHangUp
then action Alice.Contacts.awSelectBob
then action Bob.Telephone.awAnswerCall
then action Bob.Telephone.awHangUp
```

Use cases with the same priority are combined using **and** and use cases with different priorities with **then** operator!

Coverage Language

Design principles:

1. Syntax should be concise and readable
2. Elements can be required to be covered in a free order (**and**) or in some specific order (**then**)
3. There can be alternative coverage criteria (**or**)
4. Criteria can relate to both test environment and test model (**action**)
   - A test criterion can be fulfilled, for example, if the test run has already taken too long, if some resources in the test system are running low, or if some elements in the test model are covered in sufficient detail
5. Execution paths in the test model must not be restricted by the language
   - We keep the roles of the coverage criteria and the test model separate
   - The test model (alone) specifies what can be tested, whereas the coverage criteria specifies the stopping condition for test runs
   - This enables us to combine different criteria with **and**, **then**, and **or** freely

On-line Test Generation Algorithm

- Greedy bounded-depth search

```
NextStep($s$ : state, $depth$ : integer, $c$ : coverage requirement)
if $depth = 0$ or $s.outTransitions() = \emptyset$ or $c.getRate() = 1$ then return $(c.getRate(), \{\})$
$best\_rate = 0$; $best\_transitions = \{\}$
for each $t \in s.outTransitions()$ do
    $c.push()$
    $c.markExecuted(t)$
    $(new\_rate, dont\_care) =$ NextStep$(t.destinationState(), depth - 1, c)$
    $c.pop()$
    if $new\_rate > best\_rate$ then $best\_rate = new\_rate$; $best\_transitions = \{t\}$
    if $new\_rate = best\_rate$ then $best\_transitions = best\_transitions \cup \{t\}$
end for
return $(best\_rate, best\_transitions)$
```

Conclusions

There is a mismatch between the artifact produced in agile projects and those needed by conventional MBT

We presented an approach for using informal use cases to drive on-line test generation and facilitate reporting in term of requirements coverage

The approach is domain-specific and relies on the existence of domain and MBT experts to develop the behavioral test models

We also introduced the associated coverage language and test generation algorithm

Prototype tools are being developed, industrial case studies have been scheduled

For further information contact:

Mika Katara

mika.katara@tut.fi

Visit our web site at http://practise.cs.tut.fi (TEMA project)