



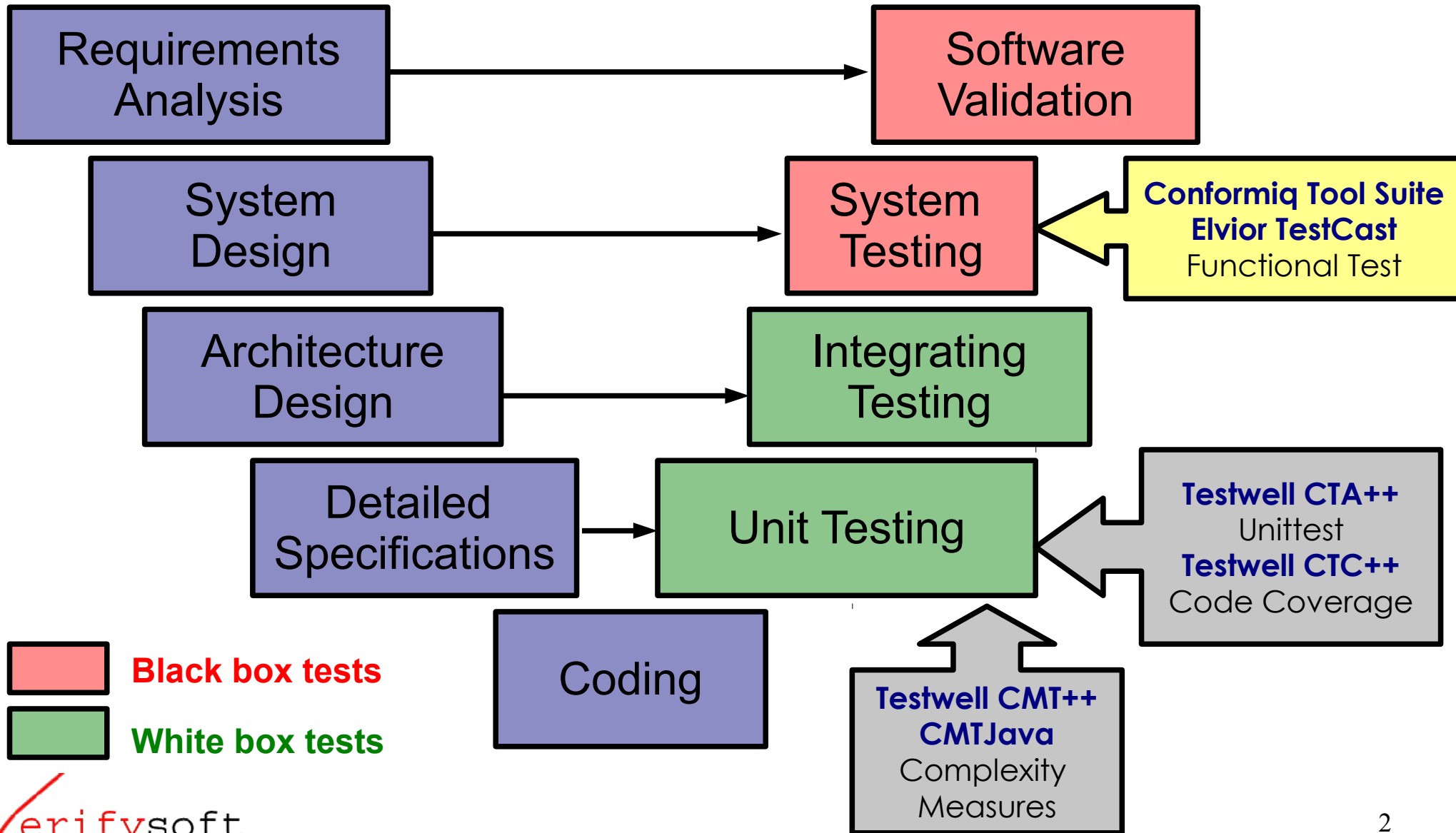
Conformiq Tool Suite

Automatic Test Design

5 October 2011



Software development process

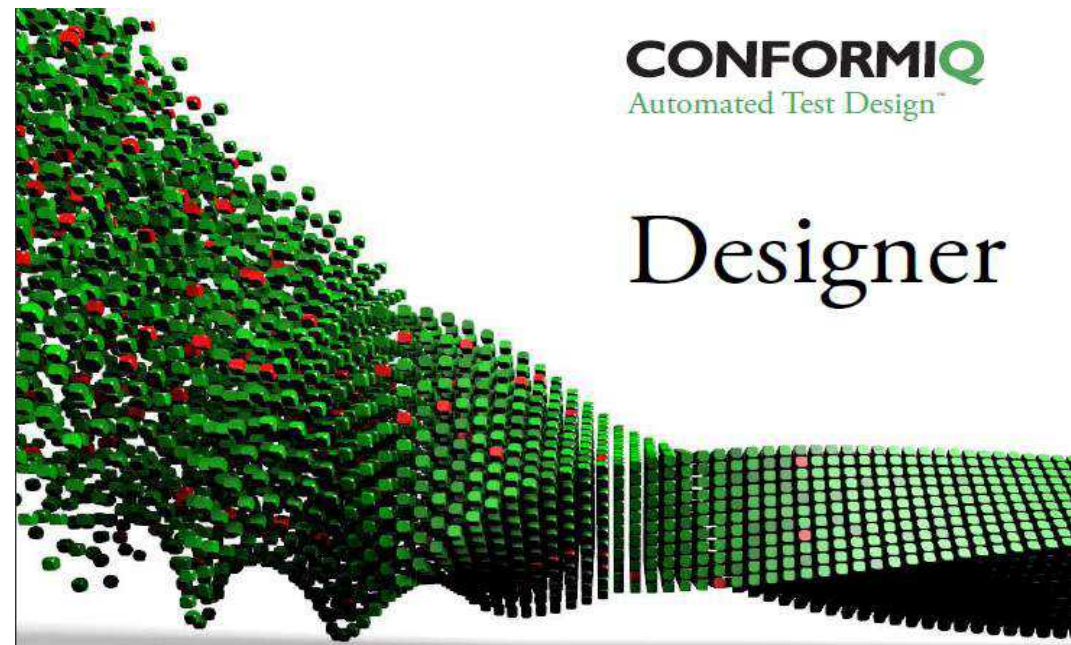




Conformiq Tool Suite

Conformiq Designer:
Model-Based Automated Test Design
for functional tests

(black box tests)
for software and systems



Conformiq Designer

Manual tests takes time...
and leads to risks

- 💣 incorrect tests
- 💣 missed tests
- 🕒 redundant tests
- 🕒 maintenance of tests





Conformiq Designer

Our solution: **Automated Test Design**

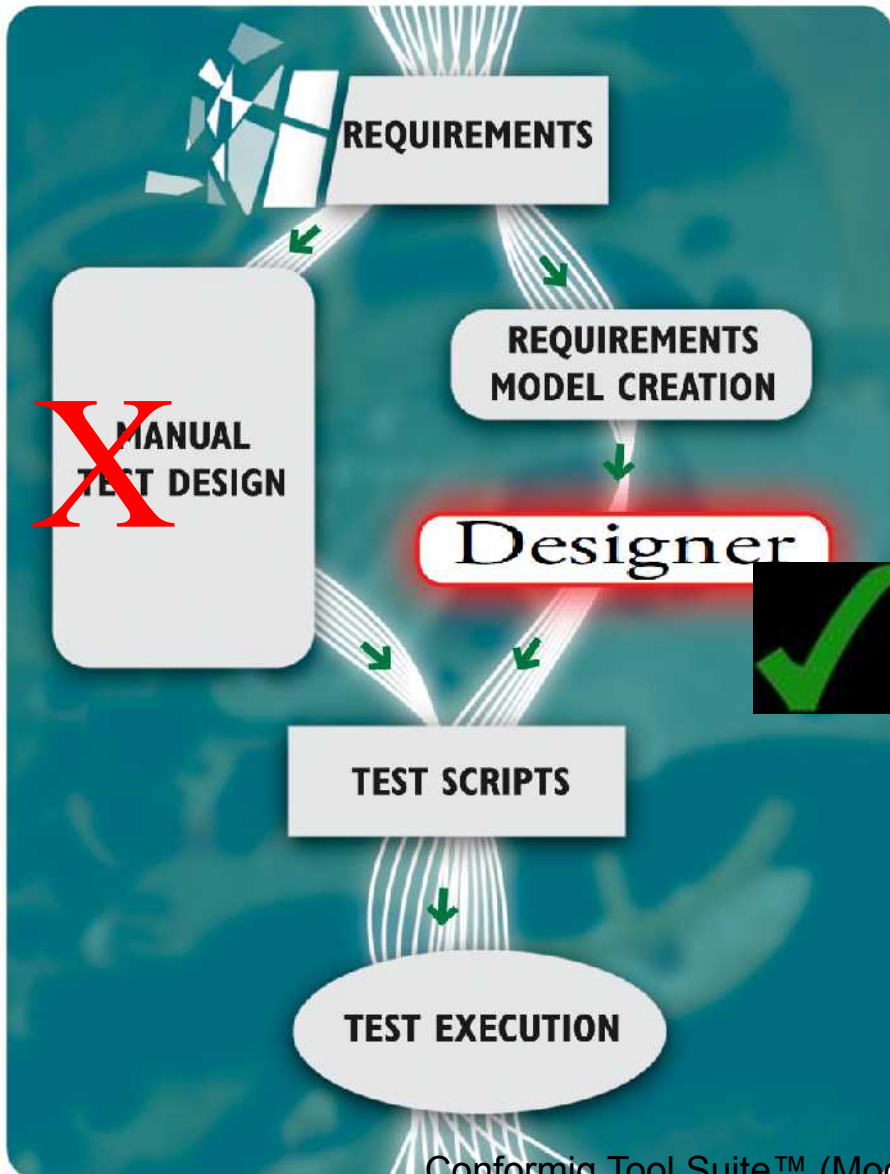
→ model driven testing, model based testing,
specification based testing,
specification driven testing,
...



Manual vs. Automatic

Automatic test generation and execution based on your design models

instead of writing the tests manually



Test generation

Process:

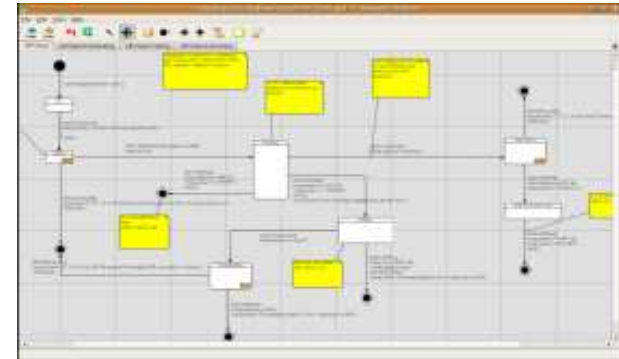
Creation of the model

Import of the model

Selection of the test coverage requirements and formats for the test scripts.

Automatic test generation with Conformiq Designer

Test execution in your environment.



Test generation

Process:

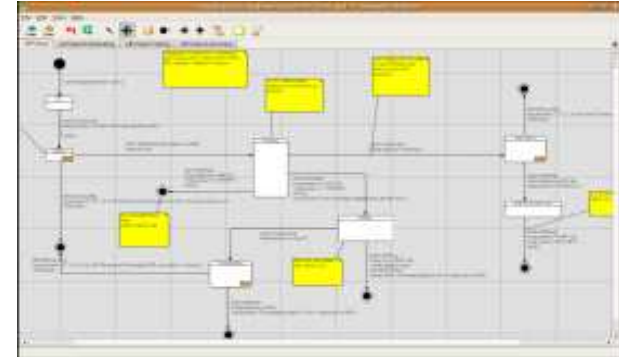
Creation of the model

Import of the model

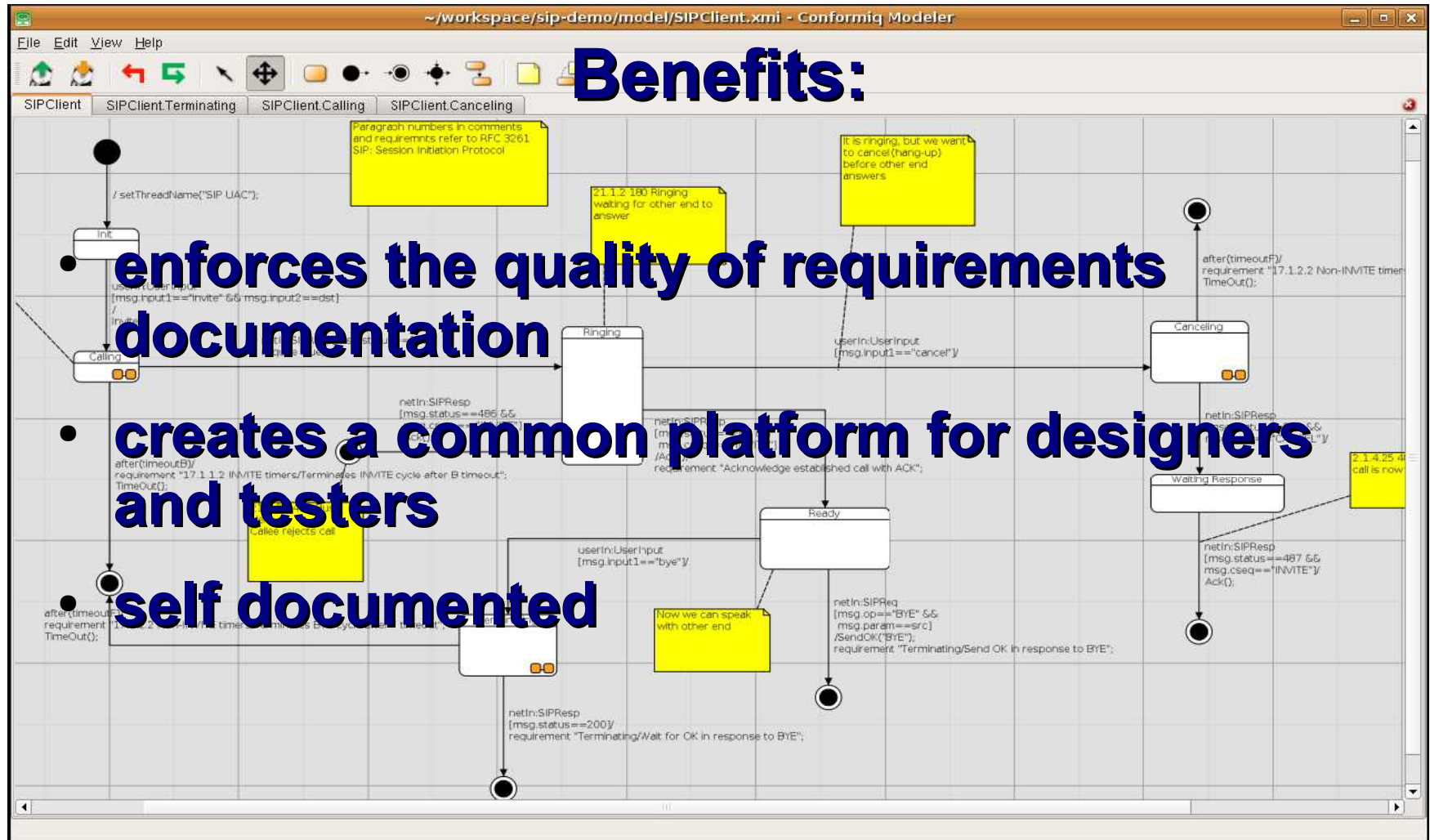
Selection of the test coverage requirements and formats for the test scripts.

Automatic test generation with Conformiq Designer

Test execution in your environment.



Model Driven Testing





Modelling – The Model

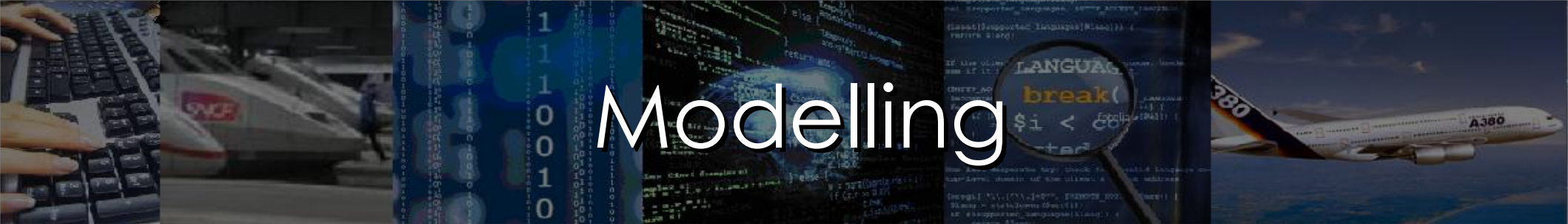
- When designing your MBT model you can have two different approaches:
 - Design your model from scratch – independently from other existing models
 - Reuse and adapt other existing development models:
 - Customize the model to your needs
 - Simplify the model
 - Strip the model from all irrelevant information
 - Make sure that it is black box, and describes the system's behaviour in a proper way



Modellierung

Supported Constructs:

- **Data** (strings, numbers, records, classes, arrays...)
- **Time** (timeouts, dynamic timeouts...)
- **Control structures** (methods, dynamic polymorphism...)
- **Concurrency** (multiple Java threads in model, ITC primitives...)
- **Java** + templates, macros, record values, type inference...



Modelling

- Textually in Java (with elements in C#)
- Grafically in UML State Charts (optional)
- The model can be created with:
 - Text editor („Java“)
 - Conformiq Modeler (UML State Charts)
 - or Third Party Modeling (UML) tools

Test generation

Process:

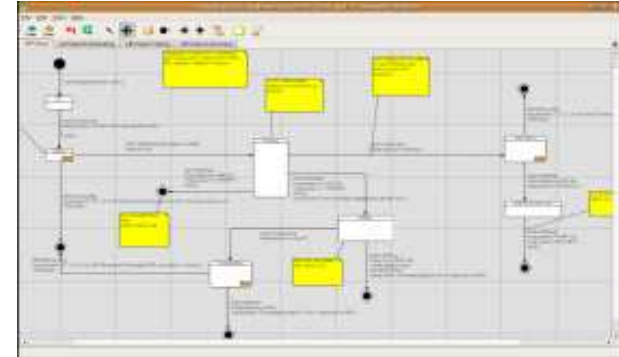
Creation of the model

Import of the model

Selection of the test coverage requirements and formats for the test scripts.

Automatic test generation with Conformiq Designer

Test execution in your environment.



Test generation

Process:

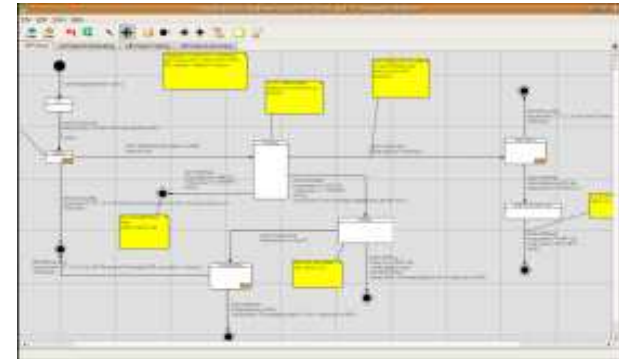
Creation of the model

Import of the model

Selection of the test coverage requirements and formats for the test scripts

Automatic test generation with Conformiq Designer

Test execution in your environment.





Coverage Criteria

- defines the exhaustiveness of testing
- choice depends on the model; Designer also assists here by only showing criteria that is relevant for a particular model
- the number of checkpoints thus generated is linearly proportional to the size of the model



State Charts

- Transition coverage
 - Cover all transitions in all state charts
- 2-transition coverage
 - Cover all transition sequences of length 2 in all state charts
- State Coverage
 - Cover all states in all state charts



Conditional Branching

- Boundary value coverage
 - For every test $x < y$, cover cases $x = y - 1$; $x < y - 1$; $x = y$; and $x > y$
 - Other comparators work analogously
- Branch coverage
 - For every **if** and **while** loop, cover both the positive and the negative branch
- Atomic 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

- to add traceability to system requirements
 - example: Inventory system must give an alarm under appropriate circumstances
- to extend test coverage by expressing testing targets that are not covered by model coverage criteria



Control Flow

- Method coverage
 - covers every QML level method
- Statement coverage
 - covers every QML level statement



Test generation

Formats for the test scripts generation:

Python

TCL

TTCN-3

C, C++

Visual Basic

Java

Junit

Perl

Excel

HTML

Word

Shell Scripts

Test generation

Process:

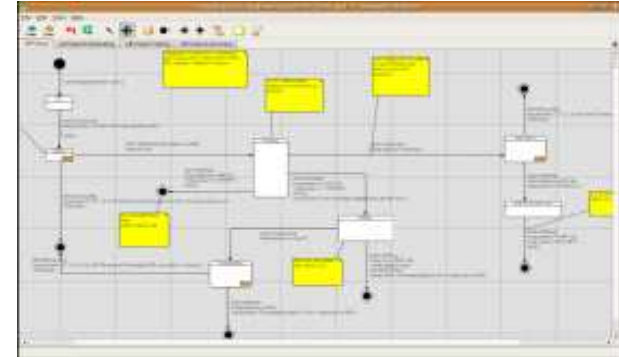
Creation of the model

Import of the model

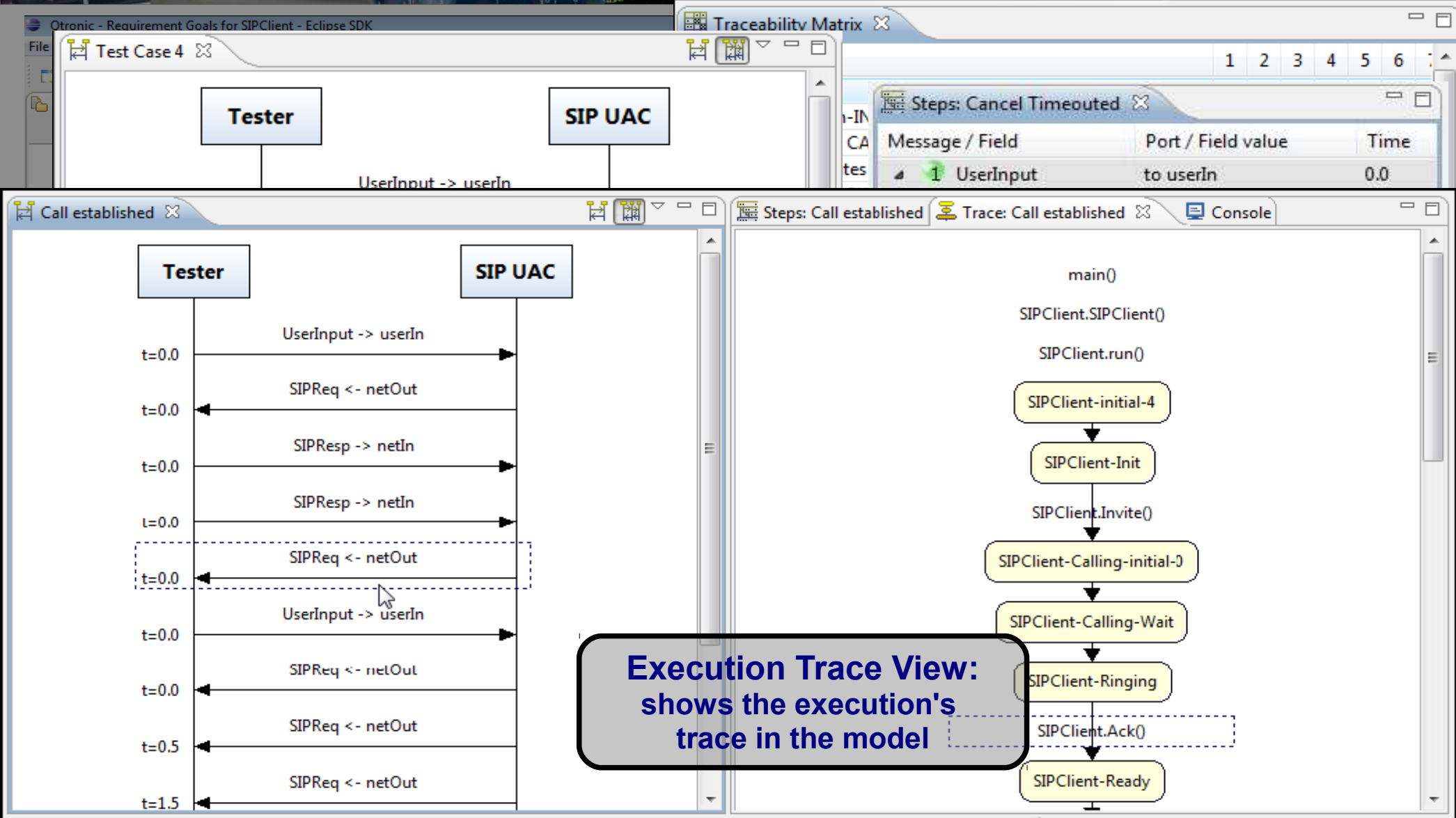
Selection of the test coverage requirements and formats for the test scripts.

Automatic test generation with Conformiq Designer

Test execution in your environment.

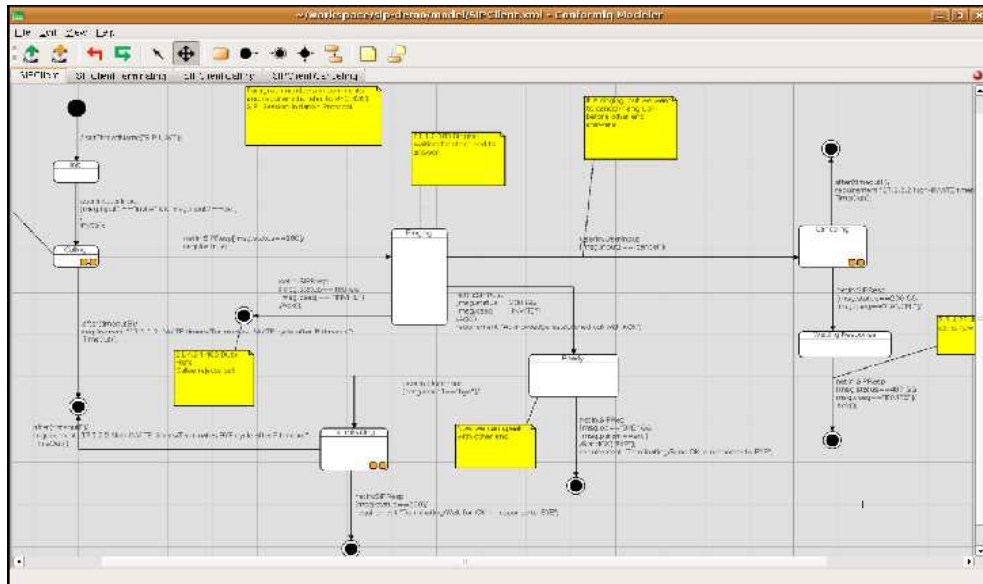


Test scripts generation



Conformiq Designer

Model



Test scripts generation



Test generation

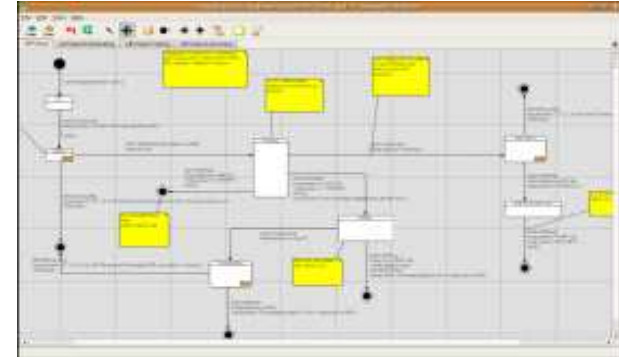
Process:

Creation of the model

Import of the model

Selection of the test coverage requirements and formats for the test scripts.

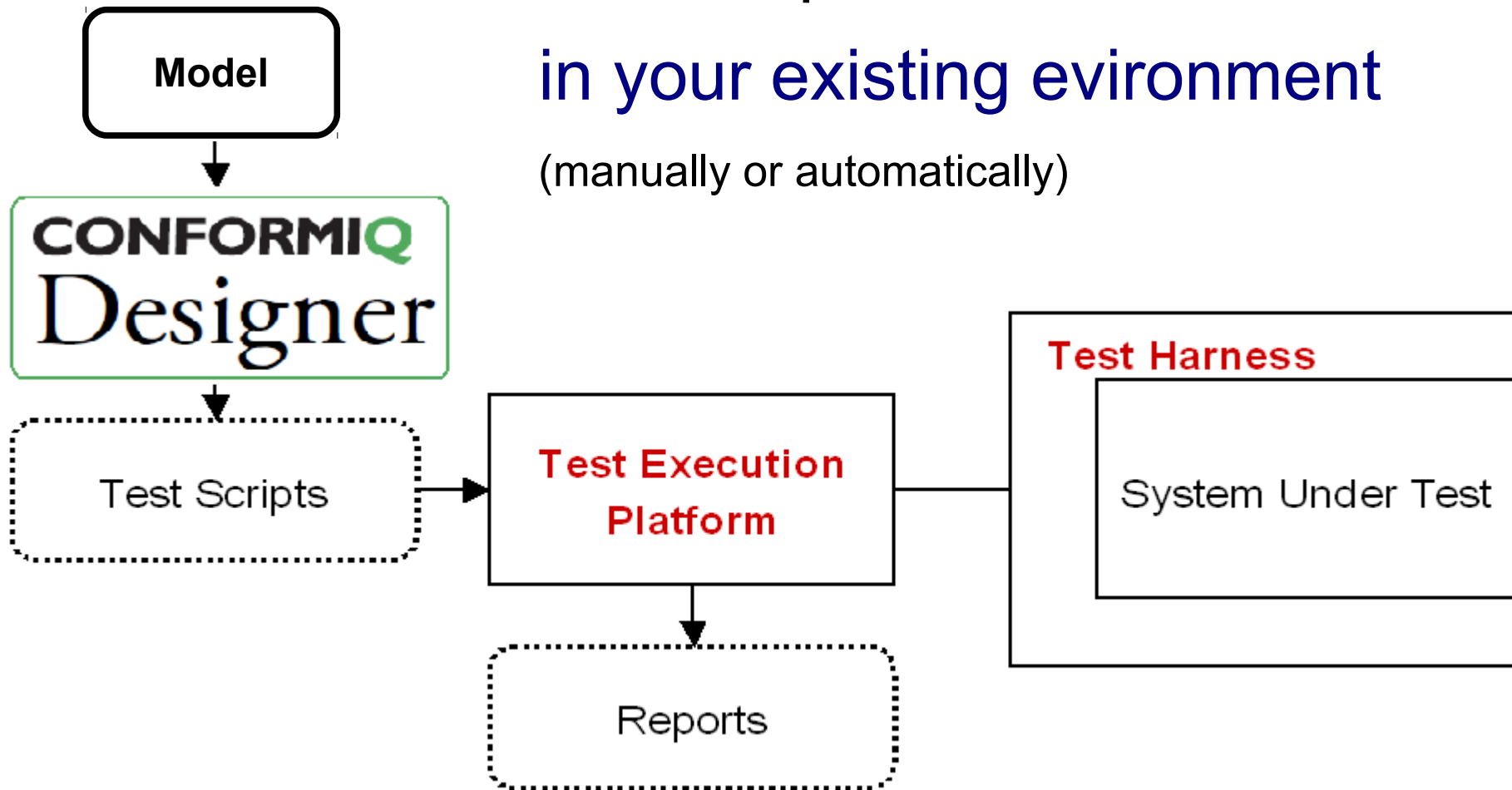
Automatic test generation with Conformiq Designer



Test execution in your environment

Test scripts execution

Test scripts execution
in your existing environment
(manually or automatically)





Conformiq Tool Suite

Issues with traditional test case development

- Unknown & inadequate functional testing coverage
- Difficulties to establish requirement traceability to the test cases
- Missed test cases or scenarios
- Growing test suite after a series of revisions not knowing which test cases are redundant
- Update on test suite after each revision is time consuming
- Pressure to conduct testing quicker
due to late delivery of implementation



Conformiq Tool Suite

MBT vs. Traditional testing



MBT vs. Traditional testing

- Compared to traditional test methods, MBT is a complete paradigm shift.
- New competences are required, and training must be taken care of.
- New roles must be established within the test organisation, especillay the model designer (“test architect”).
- The testers must learn to not think as testers.

When designing the model, the tester shall not think in terms of test cases – the tester should, ultimately, only think of the system behaviour.



MBT vs. Traditional testing

- The tester must have a thorough understanding of the new functionality, and be involved in (and contribute to!) the development project already in the early stages.
- Basic level of “stability“ (organisation, test environments, etc.) is needed.
- And from experience:
For a tester to work with MBT, is challenging, inspiring and Fun!



Conformiq Tool Suite

Customer Feedback



Customer Feedback

- The generated test cases are better than our traditional test cases.
- The generated test cases cover several events (Conformiq Designer requirements), while the traditional test cases normally only cover one event/situation.
- And we have found faults with the generated test cases that we would not have found with the traditional test cases.



Customer Feedback

- The generated test suite is easy to read, follow and understand.
Conformiq provides plug-ins that will render the generated test cases into an html document and/or automated test scripts.



Customer Feedback

- The output consists of a test suite, which is divided into a number of test cases, which in turn contain a number of test steps.
- The number of generated test cases is greater than the number of test cases we would have written “manually” with our traditional methods.



Customer Feedback

- The generated test cases are better than our traditional test cases. The generated test cases cover several events (Conformiq Designer requirements), while the traditional test cases normally only cover one event/situation. And we have found faults with the generated test cases that we would not have found with the traditional test cases.



Customer Feedback

- The generated test suite is easy to read, follow and understand.
Conformiq provides plug-ins that will render the generated test cases into an html document and/or automated test scripts.



Customer Feedback

- The generated test scripts get a good structure, and they are also easy to read, follow and understand. They are also easy to edit, which is good for trouble-shooting purposes.
- Furthermore you can also easily build your own test cases using the structure from the generated test suite.



Conclusions - Experiences

Hard experiences:

- Time:
 - First project : approximately same lead time and same number of resources (no one with experience in this way of working).
 - Following projects: same lead time but fewer resources.
 - Forthcoming projects: shorter lead time and fewer resources.

Major gains have been achieved when implementing similar functionality on different platforms, and with incremental development.



Conclusions - Experiences

Hard experiences (continued):

- Coverage:
 - Faults found that we would not have found with traditional test design methods.
- Quality:
 - No decrease in the quality of the tested product has been observed.
 - Too early to say whether the quality has increased



Conclusions - Experiences

Soft experiences:

- Most testers think that MBT is a very interesting way of working, and are eager to learn.
- High motivation among the testers that have been working with it – and everyone think is Fun (most of the time).
- Not all testers are suited for working with modelling.

Conclusions - Recommendations

If you are a test organization, and if you benefit from have automated test suites:

Try out the MBT way of working

But

→ Don't go for a full deployment from the start.
Start with a smaller, well defined, well encapsulated, area/functionality.

And

→ In the beginning: Stay away from functionality where you suspect you cannot avoid models with parameter explosion or great depth.



more Information

 **verifysoft**
TECHNOLOGY

Tel: +49 781 127 8118-0
quality@verifysoft.com
www.verifysoft.com



 **verifysoft**
TECHNOLOGY

Thank you!