UMR IRISA

Model-based testing

Jean-Marc Jézéquel
Benoit Baudry

UNIVERSITÉ DE RENNES 1

Inria INVENTEURS DU MONDE NUMÉRIQUE

Outline

1. Motivations
2. Model-based testing process
3. MBT with use cases
4. MBT with state-based models
5. Conclusion

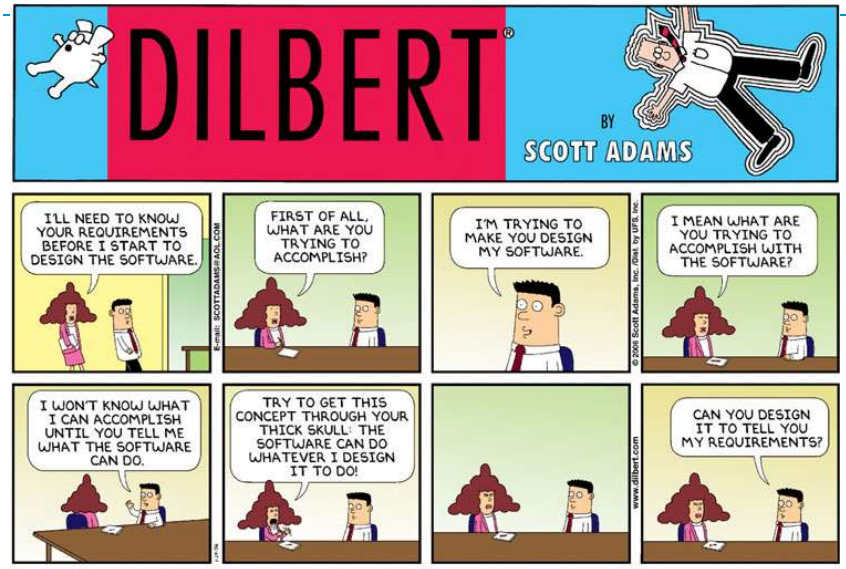
Motivations

- Generate test cases from requirements
- Automate test cases generation
- Capitalize test knowledge
- Address software product lines

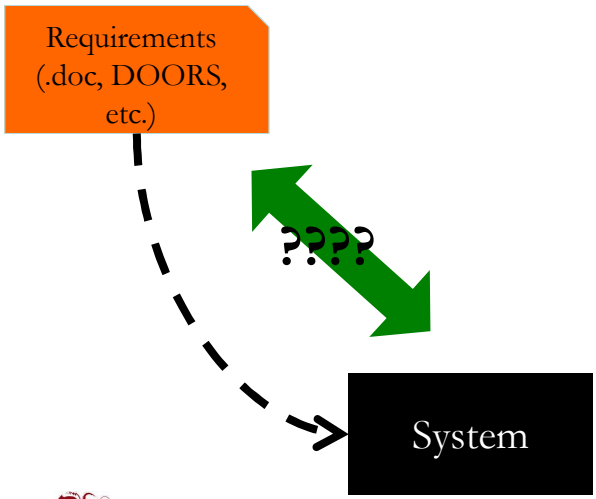
Requirements

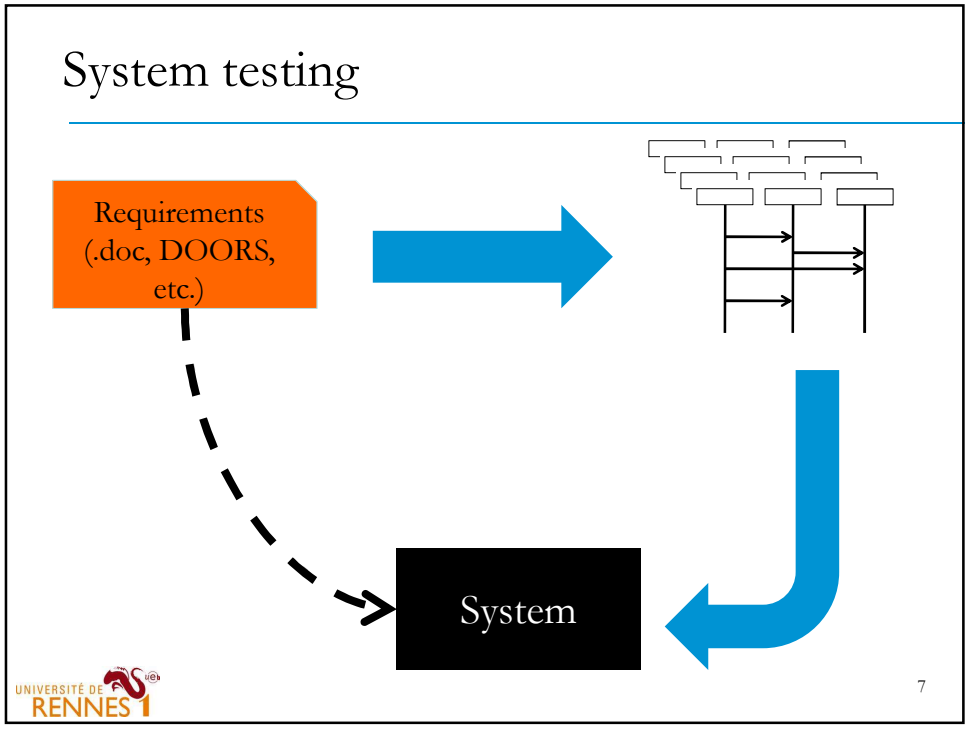
- Starting point for any SW project
- Approaches
  - Functional
  - Extra-functional?
  - Technical ?
- How do we validate fuzzy, informal descriptions?
- How can it be used to validate design and code?

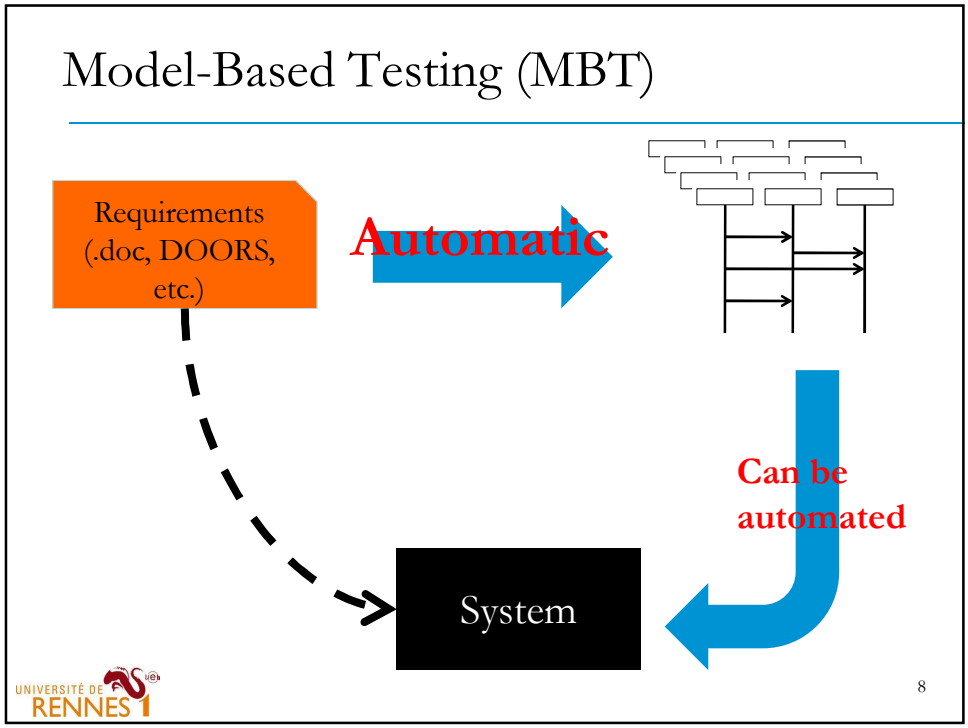
Requirements



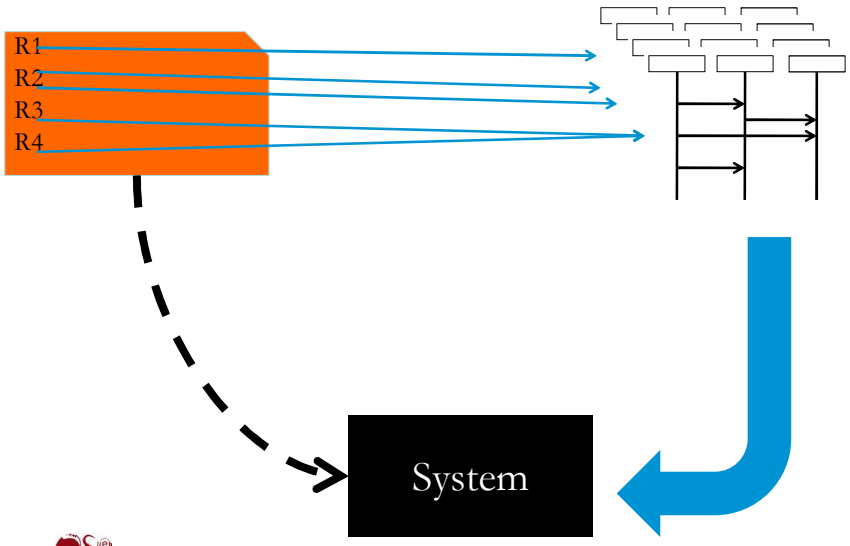
System validation




System testing
Requirements (.doc, DOORS, etc.)
System
UNIVERSITÉ DE RENNES 1
7


Model-Based Testing (MBT)
Requirements (.doc, DOORS, etc.)
Automatic
Can be automated
System
UNIVERSITÉ DE RENNES 1
8

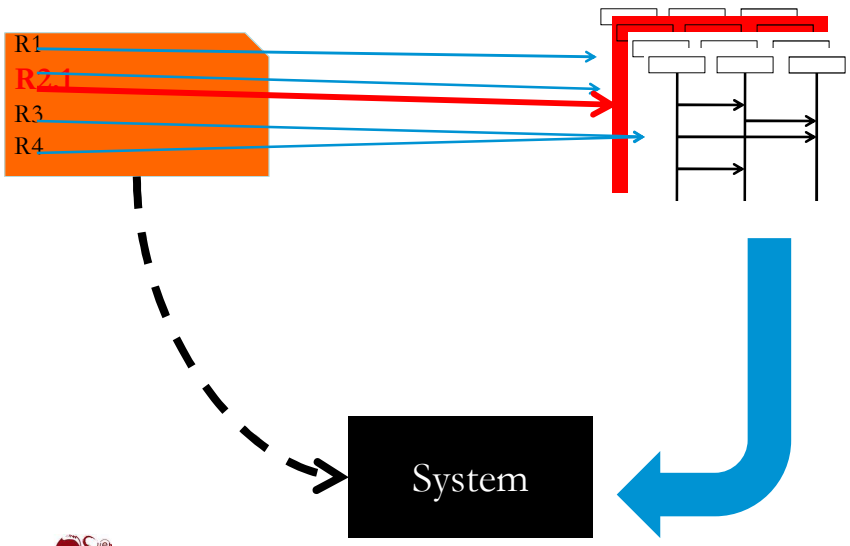
Model-Based Testing (MBT)



UNIVERSITÉ DE RENNES 1

9

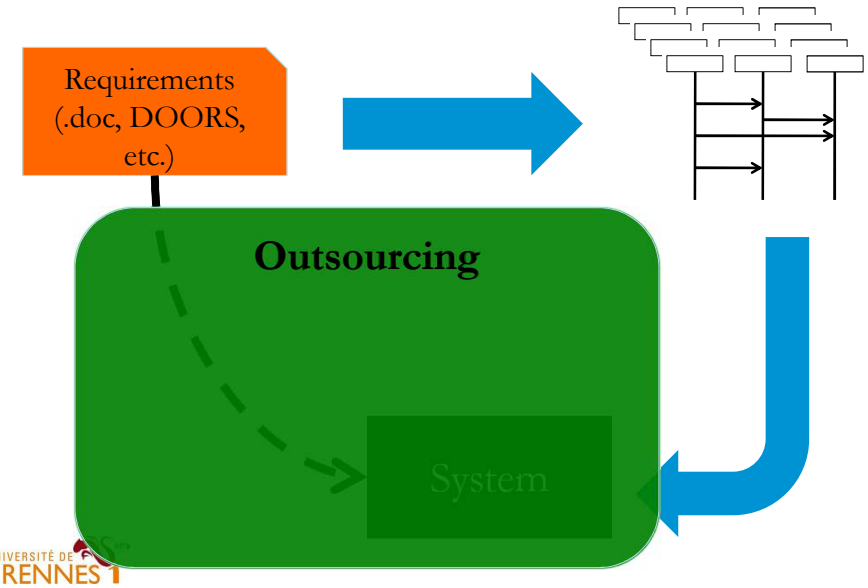
Model-Based Testing (MBT)



UNIVERSITÉ DE RENNES 1

10

Model-Based Testing (MBT)



UNIVERSITÉ DE RENNES 1

11

SW systems

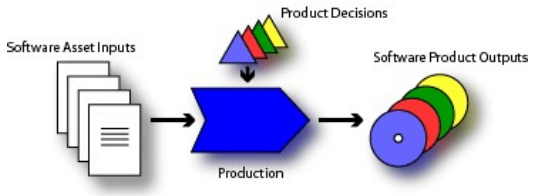
- Growing complexity
- System requirements ...
  - … change frequently
    - Nokia : 69% of requirements modified, 22% modified twice
  - … natural language

    → first need to formalize
- Software product lines
  - Core requirements common to all products + variations

UNIVERSITÉ DE RENNES 1

page 13

Software product lines

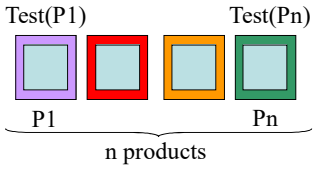


- Capitalize on commonalities
  - Avoid redundancies
- Need to deal with variability
  - Variation points
  - Decision models
- Model all possible configurations

UNIVERSITÉ DE RENNES 1

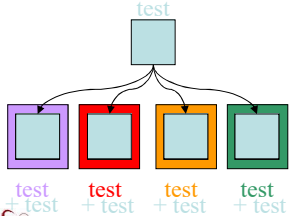
Test a SPL

Classical approach



- linear generation cost
- Highest cost for a new product

Ideal approach



- factorize test
- Reduce test generation cost
- Reduce time for a new product

UNIVERSITÉ DE RENNES 1

Challenges for requirements based testing

- Reduce cost for test generation
  - Automate
  - From an abstract model
- Constraints:
  1. Complexity of systems
  2. Introduction in a development process
  3. Adapt to SPL

Outline

1. Motivations
2. Model-based testing process
3. MBT with use cases
4. MBT with state-based models
5. Conclusion

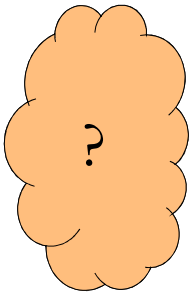
Requirements based test generation

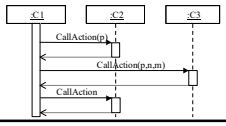
```
requirement 1.1 "Register a book"

the "book" becomes "registered" after the
"librarian" did "register" the "book".

the "book" is "available" after the
"librarian" did "register" the "book".
```

Requirements





pag

Test cases

UNIVERSITÉ DE RENNES 1

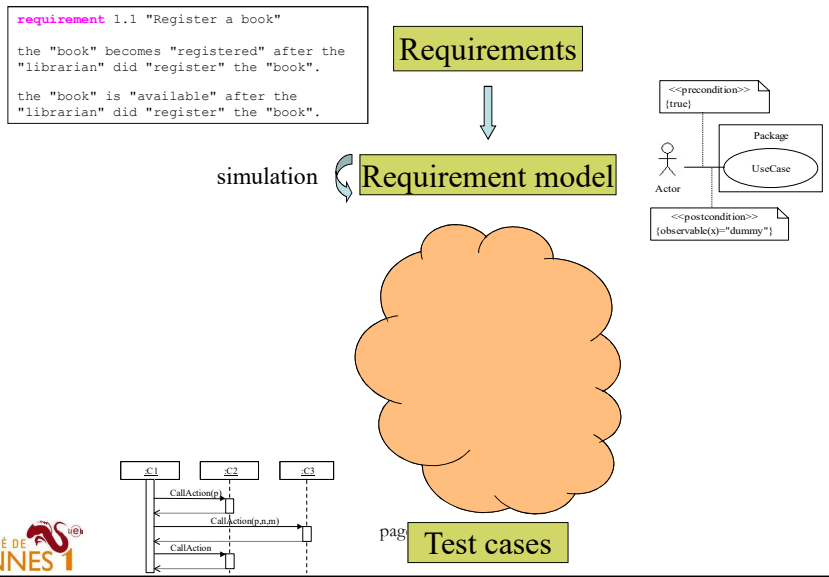
Requirements based test generation

- Start from textual requirements
- Model them
  - Use cases, state models, domain model, etc.
- Automatically generate
  - Objectives / test cases
- Applied in industrial practices

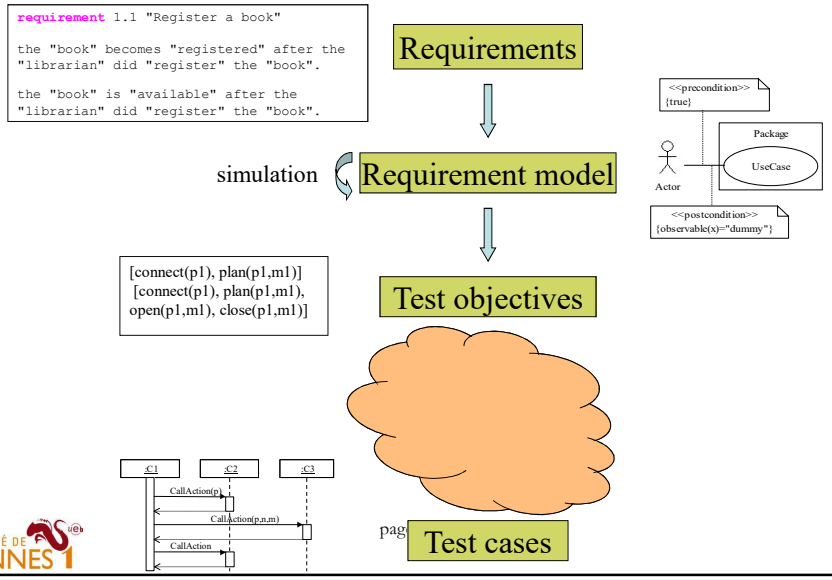
UNIVERSITÉ DE RENNES 1

page 19

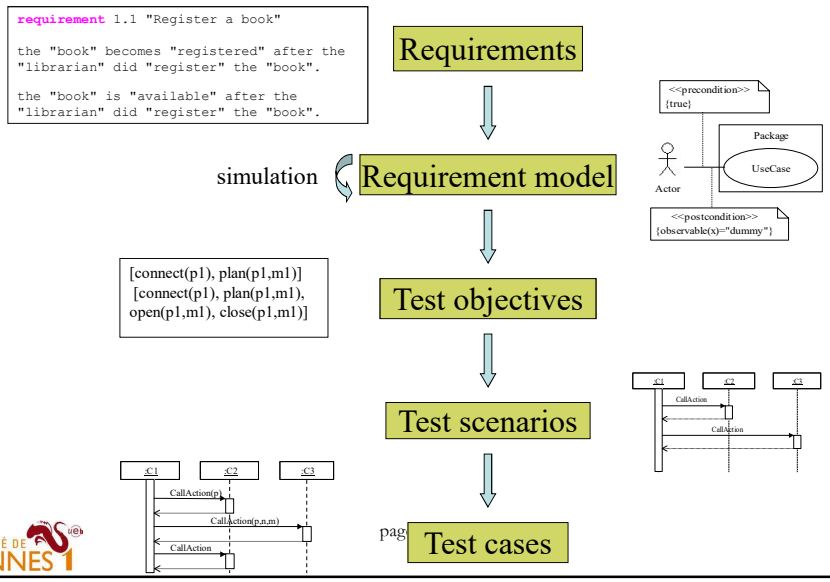
Requirements based test generation



Requirements based test generation



Requirements based test generation



Vocabulary

- Test objective = a sequence of services (use cases) to test the system
  - Requirements level
  - The actual service interface is unknown
- Test scenario = a sequence of calls on the system
  - Design level
  - Interfaces are known
- Test case = realization of a scenario (exact parameters)
  - Code level
  - Ex: JUnit test case
  - Generated from the scenario

Major challenges

- Modeling requirements
  - With enough details to generate test cases
- Generate test objectives
- Generate concrete test cases

Outline

1. Motivations
2. Model-based testing process
3. MBT with use cases
4. MBT with state-based models
5. Tools

MBT with use cases

- Use case:

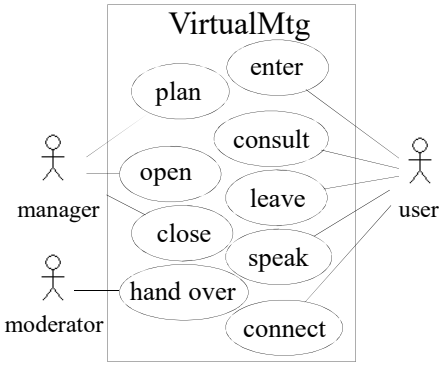
  « a particular form or pattern or exemplar of usage, a scenario that begins with some user of the system initiating some transactions or sequence of interrelated events ».

  I. Jacobson [Jacobson92]

  « Use cases are a means for specifying required usages of a system ».

  OMG, UML 2.0.

→ How can use cases drive test generation?



[Jacobson92] Jacobson et al, *Object-Oriented Software Engineering : a use case driven approach*, Addison-wesley

Why use cases ?

- According to Jacobson[1], use cases can serve for test generation:
  - Flow of events (nominal + exceptional)
  - All "features" mentioned in requirements can be traced from a use case
- Several questions (Binder[2]):
  - How to generate tests ?
  - What are the order dependencies ?
  - When are there enough test cases ?

[1] I. Jacobson et al. Object-oriented Soft. Eng.: a use-case driven approach. Addison Wesley, 1992

[2] R. Binder. Testing Object-oriented systems. Addison Wesley, 2000

MBT with use cases

« Use cases are a means for specifying required usages of a system ».

OMG, UML 2.0.

- Fröhlich et al (2000)
  - from cockburn-formatted use cases to state machines
- Ryser et al (2000)
  - from use cases to statecharts, dependency charts
- Riebish et al (2002)
  - statistical testing
- Basanieri et al (2002)
  - cow_suite approach
- Briand et al (2002)
  - TOTEM approach

Remarks

- From a functional testing perspective
  - Use cases capture functionalities under test
  - A use case can be a target for testing
- There are dependencies between test cases:
  - Sequences of use cases are required
  - It is necessary to generate valid sequences
-

Use cases for test generation

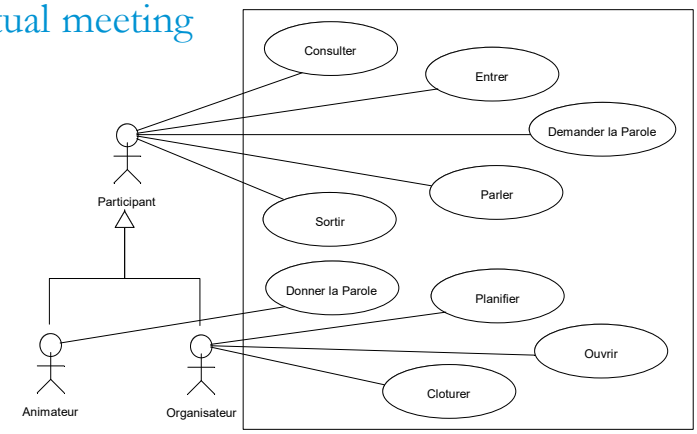
- Each use case
  - Runs in a context
  - Can be configured
  - Can be tested in isolation

MBT with use cases: a simple approach

- Use cases
- Sequence diagrams for all use-cases
  - Nominal scenarios
  - Exceptional scenarios « rares » and « fail »

MBT with use cases: a simple approach

- Virtual meeting



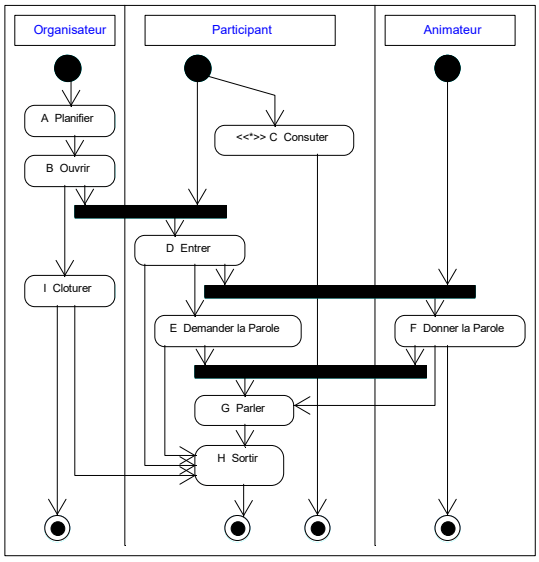
MBT with use cases: a simple approach

use case	Nominal scenarios	Rare scenarios	Failing scenarios
A Plan	N_{A1}, N_{A2}		E_{A1}, E_{A2}
B Open	N_{B1}		E_{B1}, E_{B2}
I Close	N_{I1}	R_{I1}	
C Browse	N_{C1}		E_{C1}
D Enter	N_{C1}	R_{D1}	E_{D1}, E_{D2}
E Ask to speak	N_{E1}		E_{E1}
G Talk	N_{G1}, N_{G2}	R_{G1}	E_{G1}, E_{G2}
H Leave	N_{H1}		E_{H1}
F Grant speaking	N_{F1}		E_{F1}, E_{F2}

MBT with use cases: a simple approach

- Minimum criterion:
- Cover each scenario with a test data
  - 27 data
- Covering combinations of use cases
  - Prerequesite : an activity diagram of use cases

MBT with use cases: a simple approach



Generate sequence diagrams

- Go "down" one level
  - High level : sequences of use cases
  - Lower level : scenarios associated to use cases
  - Aim : Deriving sequences of use case scenarios
- Each use case is described by a sequence diagram

Limitations

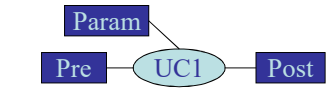
- Generation of objective well founded
- Acitivity diagram model
  - Simple
  - Cannot describe all possible dependencies between use cases
- SPLs not considered

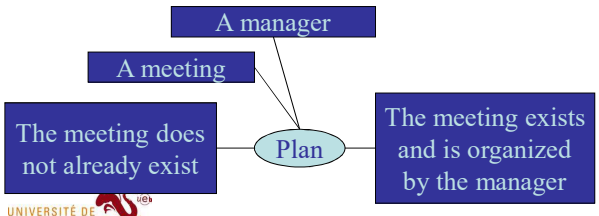
Use cases + pre/post

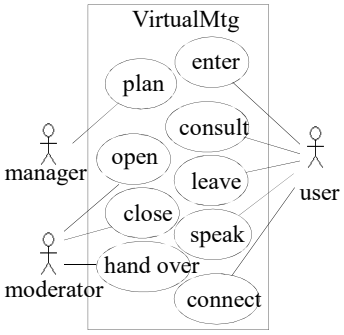
- Declarative approach
  - Specify the context for each use case
    - Pre-post
    - Contrats
  - More expressive

Augmented use cases

- Augmented UML use cases:
  - Parameters
    - actors or business concepts
    - entities involved
  - Contracts
    - precondition and postcondition
    - constraints on parameters





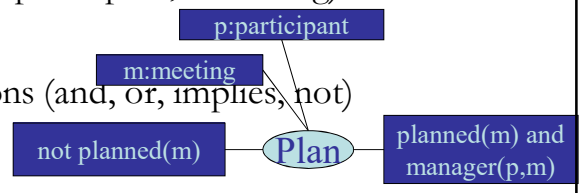


A contract language for use cases

- First order logic
  - Boolean properties (predicates) = name+typed parameters
    - Ex: planned(m:meeting)

      manager(u:participant,m:meeting)
  - Enumerations
  - Boolean operations (and, or, implies, not)
  - Quantifiers

  (forall, exists)



Infer dependencies between use cases

```
#use case OPEN
UC open(u:participant;m:mtg)
pre created(m) and moderator(u,m) and not closed(m) and
not opened(m) and connected(u)
post opened(m)

#use case CLOSE
UC close(u:participant; m:mtg)
pre opened(m) and moderator(u,m)
post ...
```

OPEN(u1,m1);CLOSE(u1,m1) is a correct sequence

Product lines

- 3 types of variation points
  - 0 or 1 variant = **optional**
  - 1 out of n variants = **choice** (xor)
  - m out of n variants = **multiple choice** (or)
- What can vary in use cases ?
  - use cases
  - parameters
  - contracts
  - scenarios

Variation dans un modèle de use cases

- How to represent variation ?
  - Annotate variable model elements
    - VP_name{variant_list}
  - Examples :

Alternative variation point

UC Record (p:participant, m:meeting)
{VP_Recording{true}}

UC enter(u:participant;m:meeting)
pre connected(u) and opened(m)
pre private(m) implies
authorized(u,m) {VPMeetingType(~~private~~)}
post entered(u,m)

Multiple variation point

Example

Edition	demo	personal	enterprise
meeting limitation	true	true	false
meeting types	{std}	{std,democ,priv}	{std,democ,priv}
recording	false	false	true
language	{En}	{En}	{En, Fr, Sp}
supervisor	false	false	true

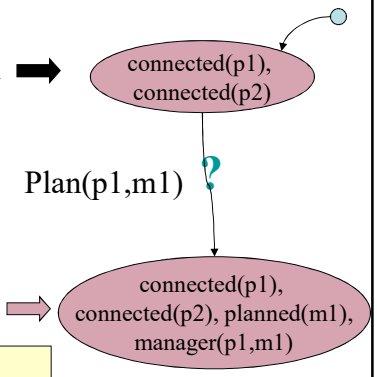
UC record(u:participant; m:mtg) {*VPRecording(true)*}
pre manager(u,m) and not opened(m)
post recorded(m)

UC enter(u:participant;m:meeting)
pre connected(u) and opened(u)
pre private(m) implies authorized(u,m) {*VPMeetingType(private)*}
post entered(u,m)

Simulating the use case model

- Decide on:
  - an initial state
  - a finite set of objects involved in simulation
    - there is no dynamic creation of objects

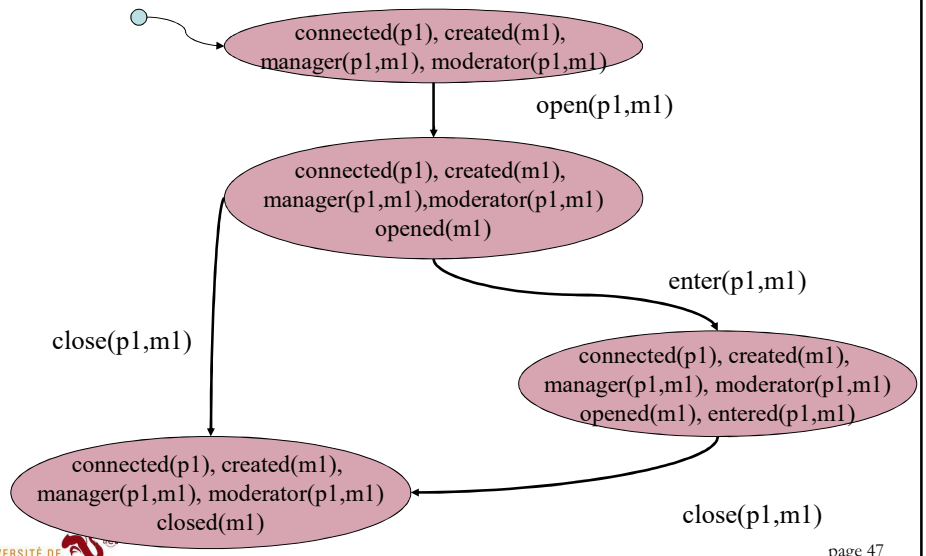
p1,p2:participant
m1,m2:meeting



- **"Run" an instanciated use case:**
  - check pre condition
  - update the current state

Plan(p:participant, m:meeting)
pre not planned(m) and connected(p)
post planned(m) and manager(p,m)

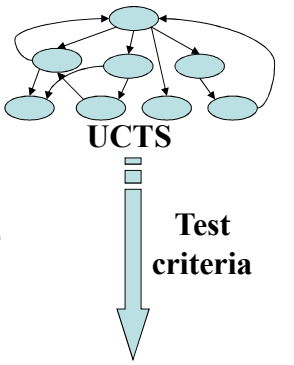
Use Case Transition System (UCTS)



UNIVERSITÉ DE RENNES 1

Test objectives generation

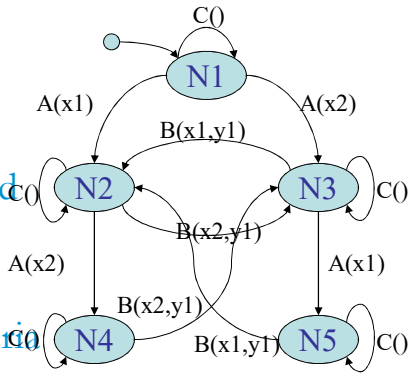
- Test objective
  - = a path in the UCTS
  - = a sequence if instantiated use cases
- Generate test objectives
  - Extract the shortest paths in the UCTS
  - A "reasonable" number of paths
  - Need criteria
    - 4 structural criteria
    - 1 semantic criterion
    - 1 robustness criterion



UNIVERSITÉ DE RENNES 1

4 structural criteria

- All transitions
- All nodes
- All instantiated use cases
- All instantiated use cases and all nodes
- need "semantic"-based criterion
- need robustness criteria

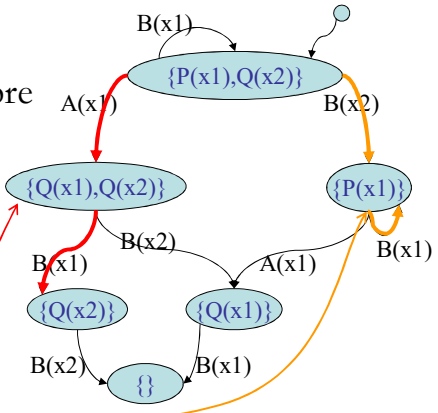


•A(x:X) x1,x2:X
•B(x:X,y:Y) y1:Y
•C()

Semantic criterion

- All precondition terms
  - Cover all possibilities for pre condition

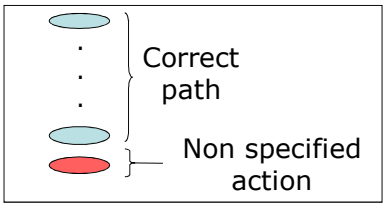
UC A(x:X)	UC B(x:X)
pre P(x)	pre P(x) or Q(x)
post not P(x) and Q(x)	post not Q(x)
x1,x2:X	



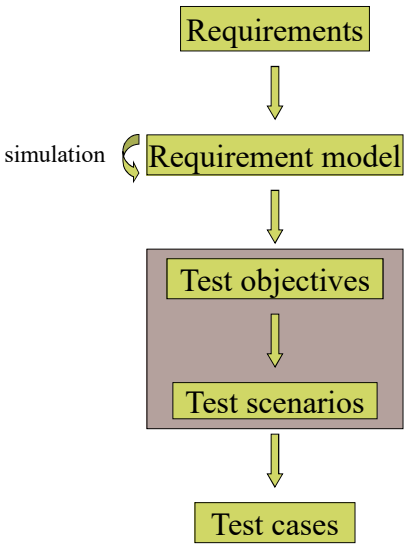
➢ 3 configurations
- •not P(x) and Q(x)
- •P(x) and not Q(x)
- •P(x) and Q(x) ╳

A robustness criterion

- Cover all valid paths that lead to pre condition violation



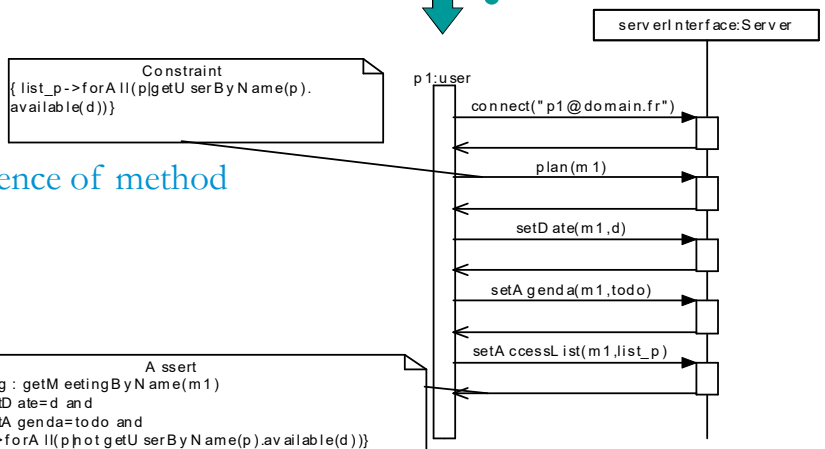
From objectives to test scenarios



Bridge a gap

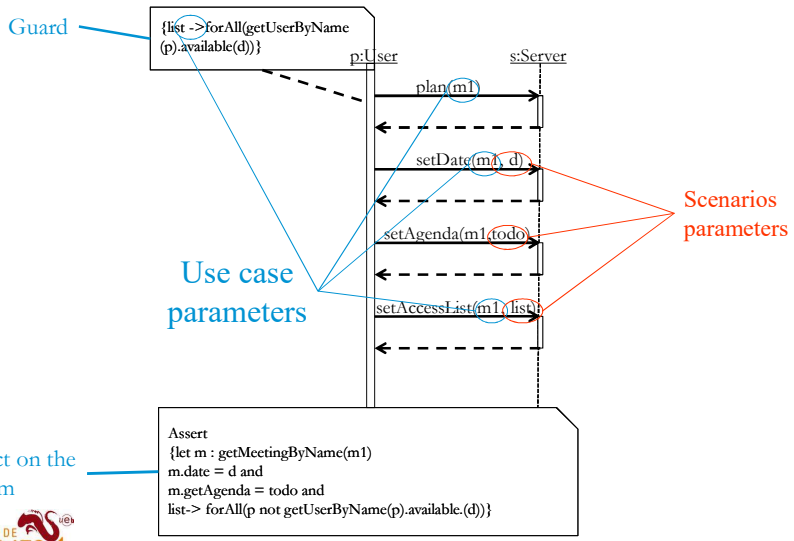
Requirements-based scenario

[connect(p1),plan(p1,m1)]

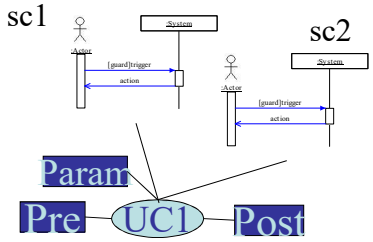


Sequence of method calls

Test scenario



Generating test scenario



Test objective

Test scenario

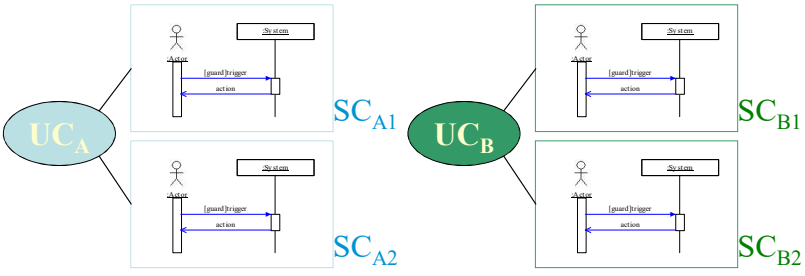
IUC1
IUC2
IUC3
…

Substitution

ISC1
ISCI2
ISCI3
…

- Strong sequential composition

Generating test scenario



Test objective → **Test scenario**

IUC$_A$ ↔ ISC$_{A1}$ • ISC$_{B1}$

IUC$_B$ ↔ ISC$_{A2}$ • ISC$_{B1}$

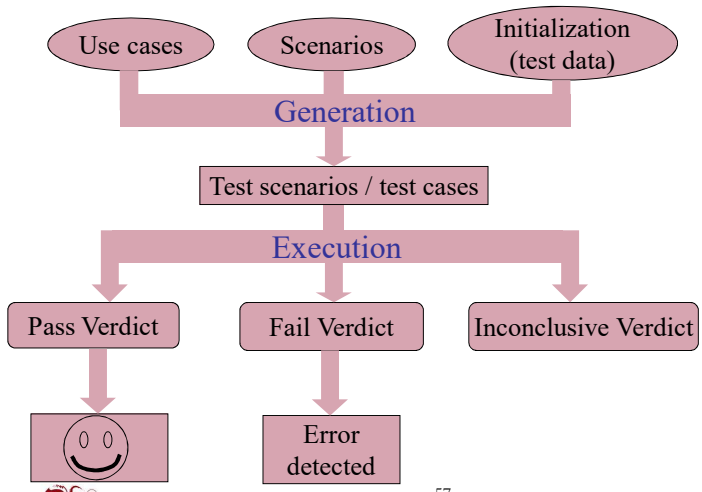
substitution

ISC$_{A1}$ • ISC$_{B2}$

ISC$_{A2}$ • ISC$_{B2}$

• = Strong sequential composition

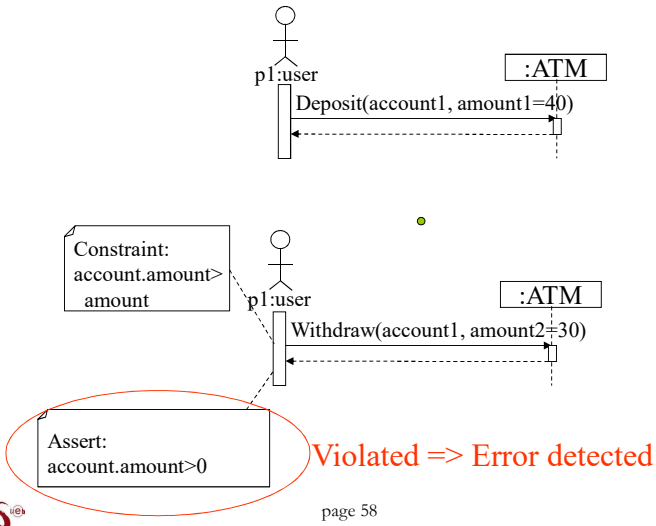
Analyze verdicts



UNIVERSITÉ DE RENNES 1

page 57

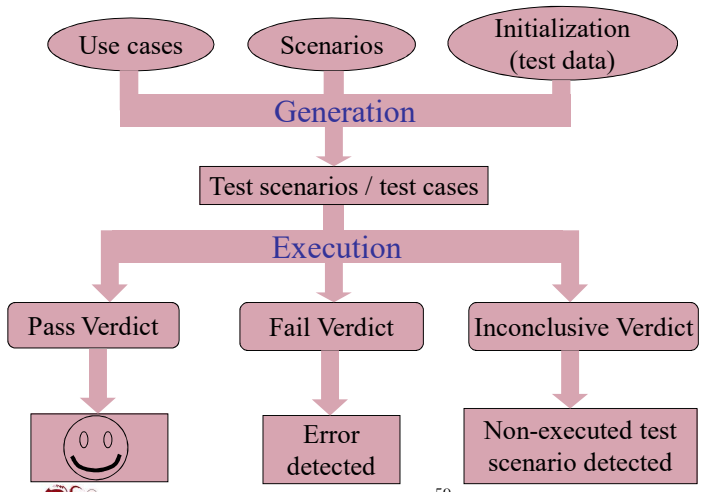
Fail Verdict



UNIVERSITÉ DE RENNES 1

page 58

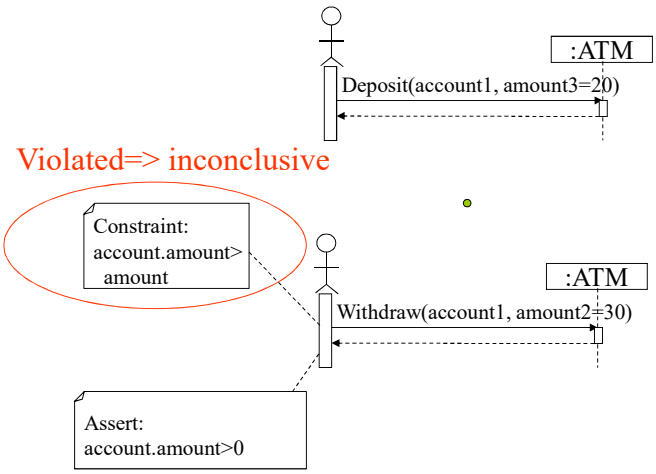
Analyze verdicts



UNIVERSITÉ DE RENNES 1

page 59

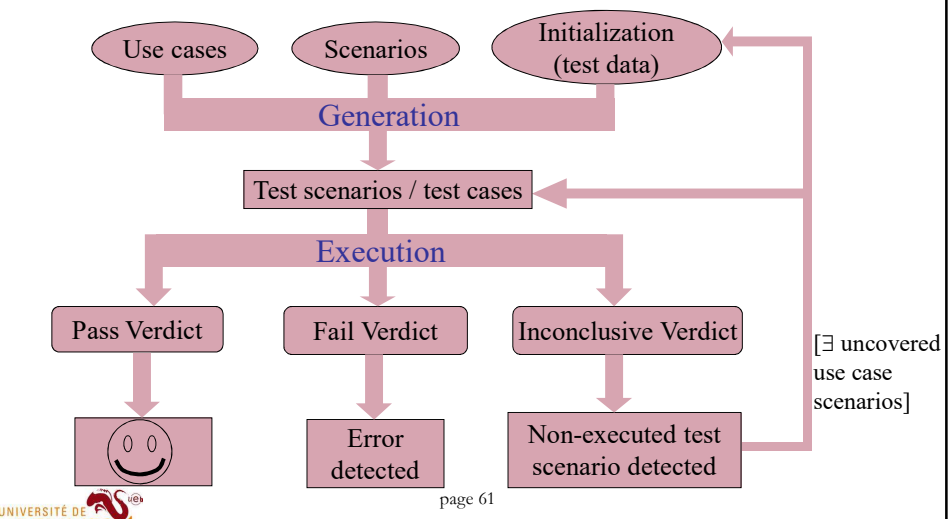
Inconclusive verdict



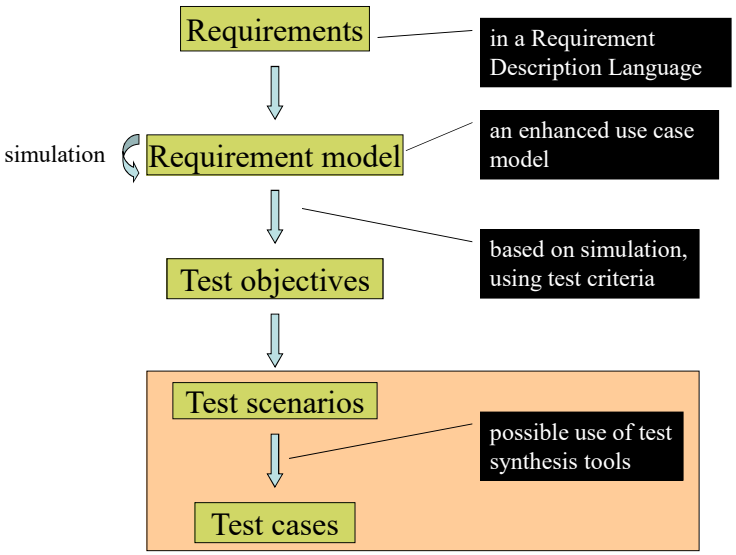
UNIVERSITÉ DE RENNES 1

page 60

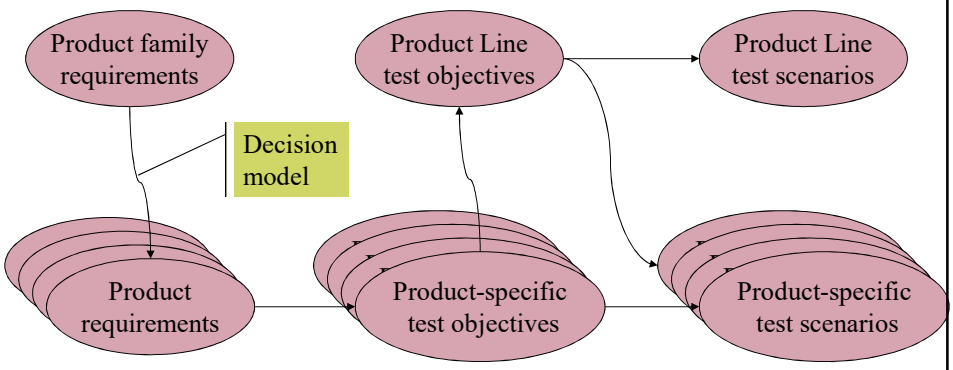
Analyze verdicts



From scenarios to test cases



Product lines



Application to SPLs

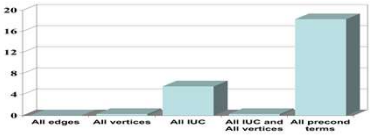
Edition	Demonstration	Personal	Enterprise
#generated test sc. with AIUC criterion	50	65	78
#generated test sc. with APT criterion	15	18	21
#generated test sc. for robustness	65	110	128
average size of tests	5	4	4

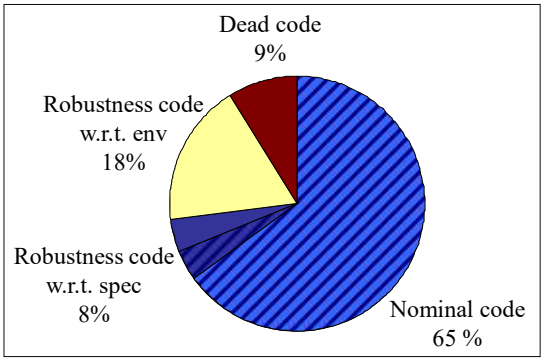
- Specific tests for each product
- Time savings:
  - Tests are automatically generated for each product

An experiment

Code coverage

for the virtual meeting example





#cov statements / #test cases

Code covered with APT + robustness criterion

UNIVERSITÉ DE RENNES 1

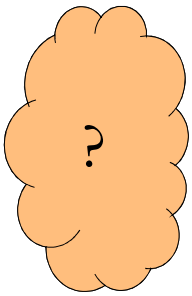
Requirements based test generation

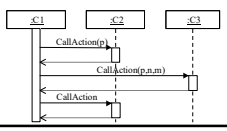
```
requirement 1.1 "Register a book"

the "book" becomes "registered" after the
"librarian" did "register" the "book".

the "book" is "available" after the
"librarian" did "register" the "book".
```

Requirements



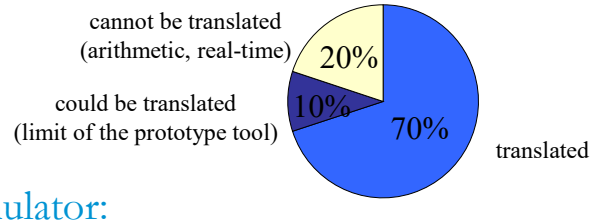


Test cases

UNIVERSITÉ DE RENNES 1

Experiments with TAS

- Two components: Mirage 2000-9 et Rafale
- Translating English to RDL



- Simulator:
  - to complete missing requirements

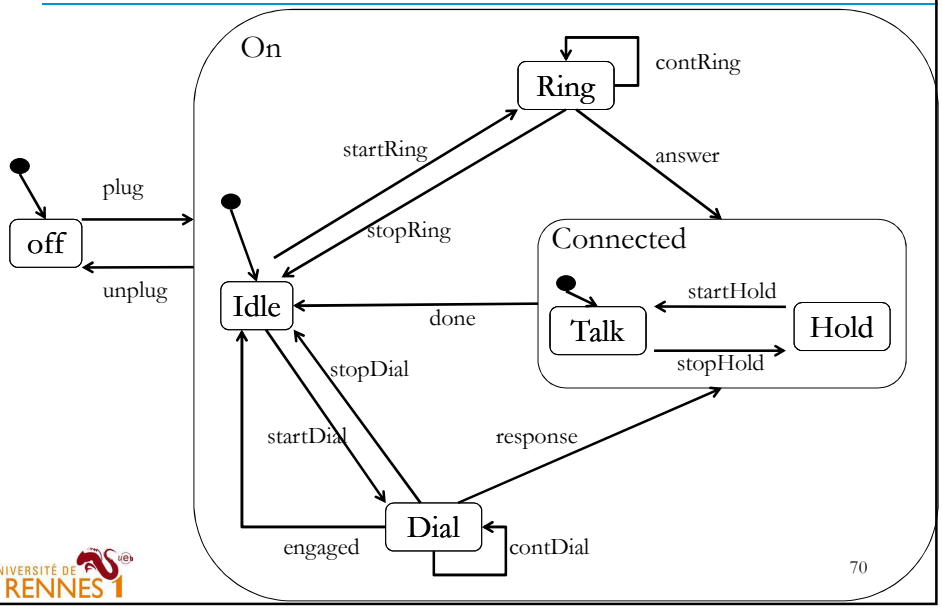
Outline

1. Motivations
2. Model-based testing process
3. MBT with use cases
4. MBT with state-based models
5. Conclusion

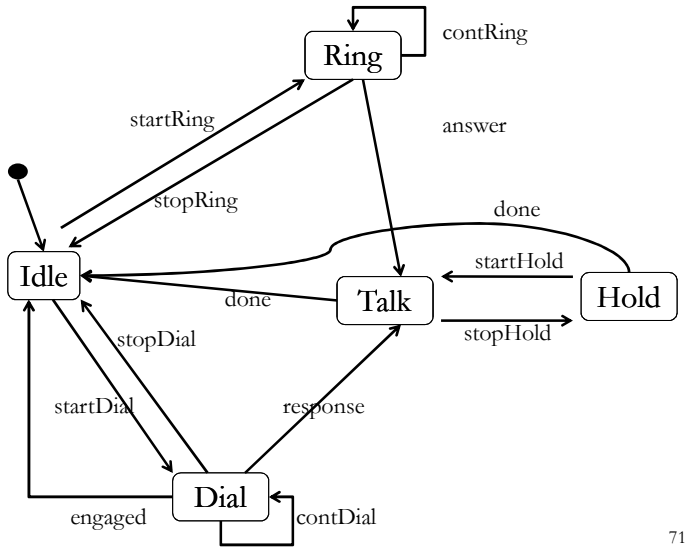
State-based approaches

- An automaton that models
  - the system's expected behavior
  - the system's environment
  - the system's configurations
- Extracts paths from the automaton
- Generate data to cover the paths

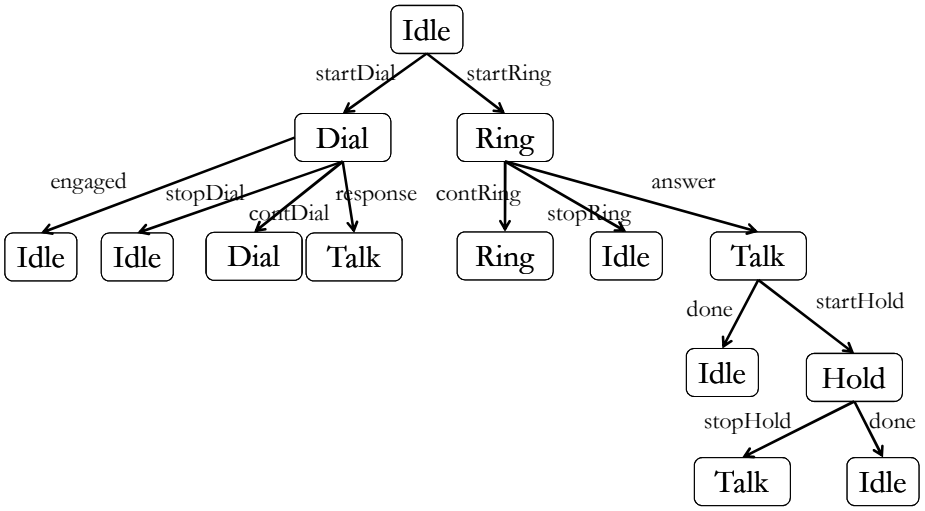
Initial statechart



Flatten statechart



Round-trip path

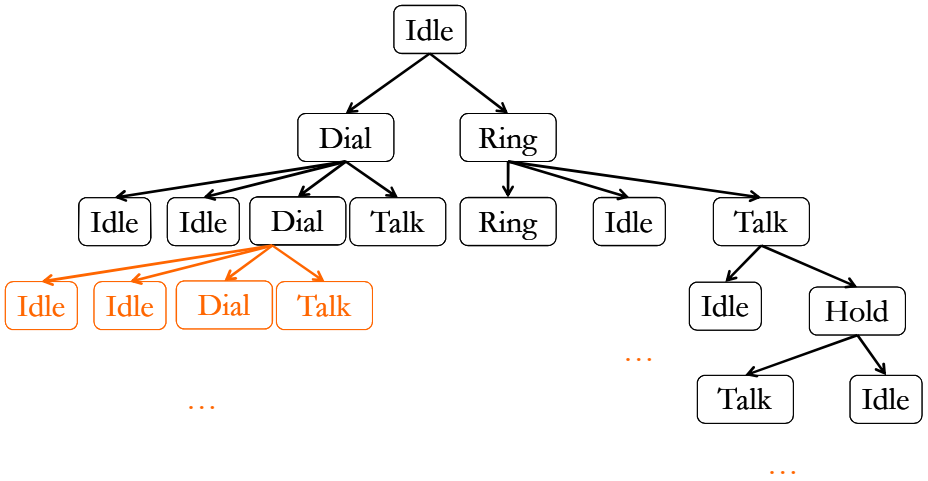


Round-trip path test generation

Test Sequence	Oracle
startDial; engaged	Dial; Idle
startDial; stopDial	Dial; Idle
startDial; contDial	Dial; Dial
startDial; reponse; startRing; contRing	Dial; Talk; Ring; Ring
startRing; stopRing	Ring; Idle
startRing; answer; done	Ring; Talk; Idle
startRing; answer; startHold; stopHold	Ring; Talk; Hold; Talk
startRing; answer; startHold; done	Ring; Talk; Hold; Idle

UNIVERSITÉ DE RENNES 1

Round-trip path - limitations



UNIVERSITÉ DE RENNES 1

Real world applications

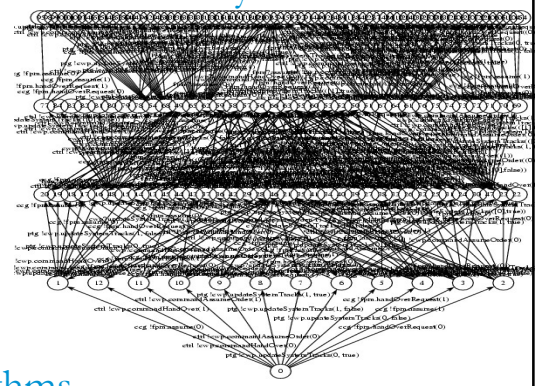
- Real world applications cannot directly be modeled with a single state machine
  - Transform the DSL/UML model into an IO-LTS (Input/Output Labeled Transition System)
- Principle of exhaustive simulation
  - aka ***model-checking***
  - Start with *initial state*, explore all possible outcomes (*fireable transitions*) recursively
  - storage and comparison of global states = (dynamic) unions of local states

UNIVERSITÉ DE RENNES 1

75

Transformation of a UML model into an (API to an) IO-LTS

- Input/Output Labeled Transition System
  - Simulation
  - Model-checking
  - Test Case Gen.
- Dynamism:
  - creations/deletions
- Infinite (in general)
- => On-the-fly algorithms



UNIVERSITÉ DE RENNES 1

76

Benefits of state-based test generation

- Automatic generation of test scenarios
  - can contain data and oracle
  - ROI
- Coverage of requirements
- Simulation of requirements
  - early error detection
  - fix inconsistencies and ambiguities early

UNIVERSITÉ DE RENNES 1

77

Limits of state-based test generation

- A model specifically for testing
- Integration with an industrial testing process
  - incremental construction of the test model
  - evolution of the test model
- A large number of test cases
  - difficult to understand and analyze

UNIVERSITÉ DE RENNES 1

78

Outline

1. Motivations
2. Model-based testing process
3. MBT with use cases
4. MBT with state-based models
5. Conclusion

Conclusion

- MBT
- Cost:
  - Explicit model of test artefacts
  - Can be wrong
- BUT:
  - Check inconsistencies in requirements
  - Automation => ROI
  - Efficiency