EDITH COWAN UNIVERSITY

CSI3105: Software Testing

Module 9a: Model Based Testing Introduction

Creative thinkers made here.

Outline

Introduction

Model-Based Testing(MBT)

- Models in SW Testing
- Finite State Machines

Fundamental Tasks of MBT

- Choosing the Model
- Building the Model
- Generating Tests
- Running Tests
- Collecting Results

Key Concerns and Closing Thoughts

Module Outline

- Video 9b:
    - Issues With Other Testing Methods
- Video 9c:
    - Model Based Testing
- Video 9d:
    - Types of Models
- Video 9e:
    - Examples
- Video 9f:
    - Summary

EDITH COWAN UNIVERSITY

CSI3105: Software Testing

Module 9b: Issues With Other Testing Methods

Creative thinkers made here.

What are the problems of Software Testing?

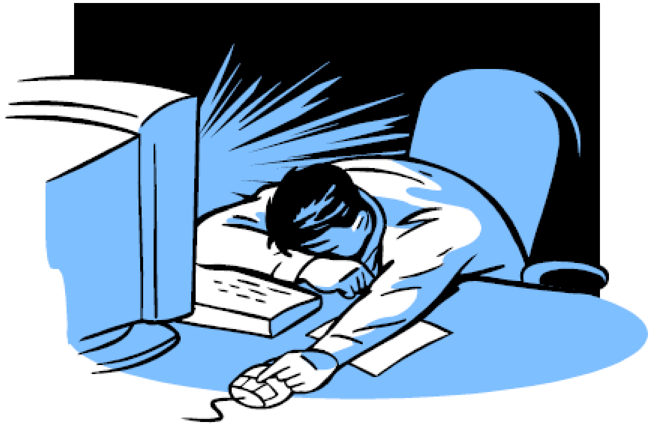
Time is limited

Applications are complex

Requirements are fluid

What is wrong with manual testing?

Testers are often perceived as bottlenecks to the delivery of software products. They are being asked to test more and more code in less and less time.



What is Test Automation?

Test automation is the **use of software to control the execution of tests**, the comparison of actual outcomes to predicted outcomes, the setting up of test preconditions, and other test control and test reporting functions.

Commonly, test automation involves automating a manual process already in place that uses a formalized testing process.

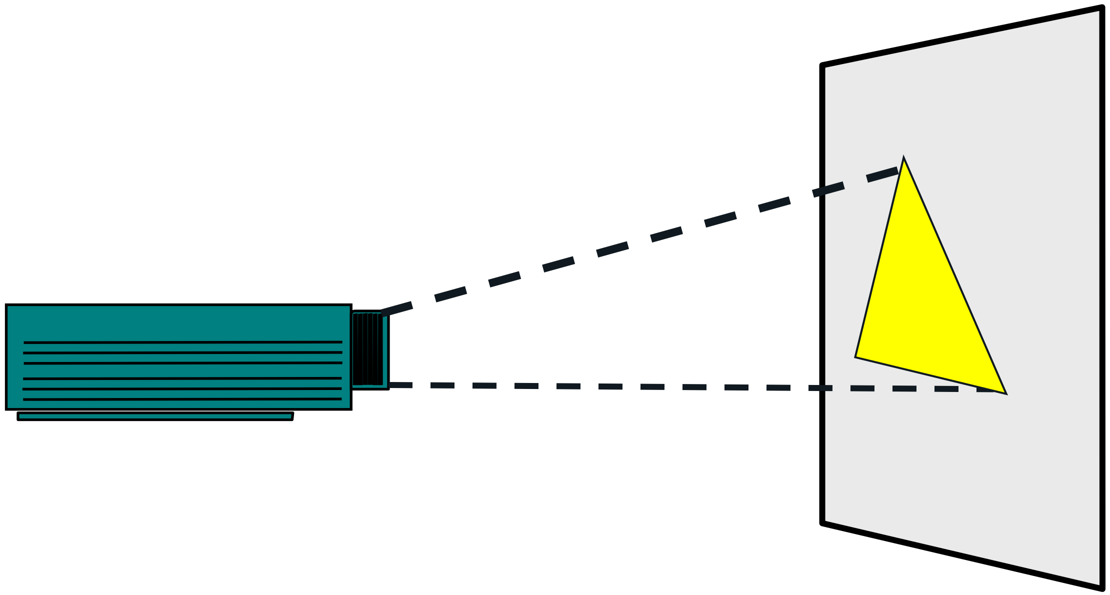
EDITH COWAN UNIVERSITY

CSI3105: Software Testing

Module 9c: Model Based Testing

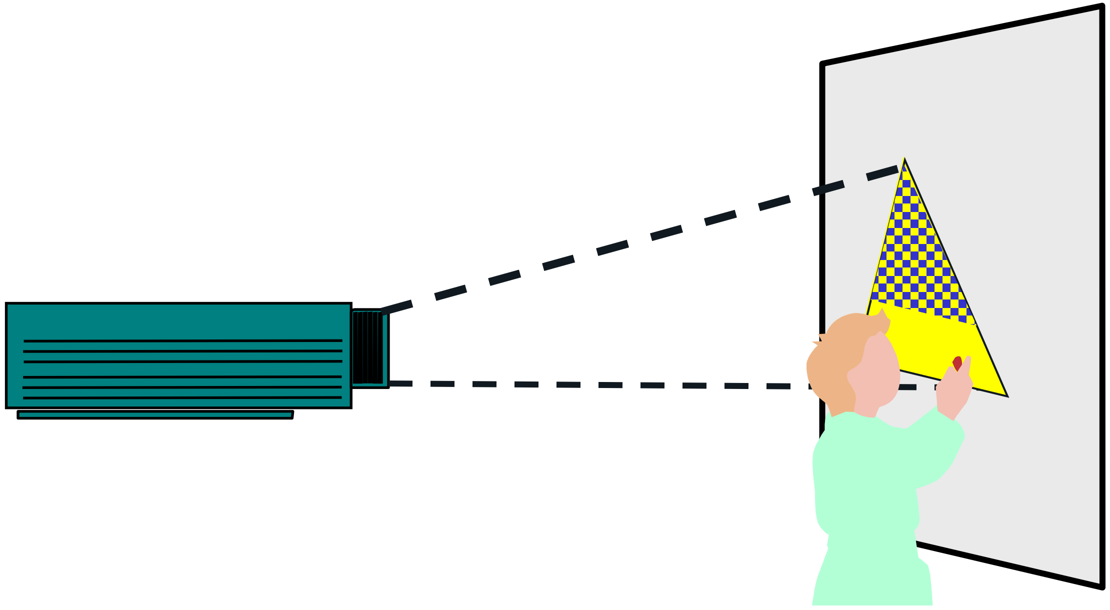
Creative thinkers made here.

Traditional Automated Testing



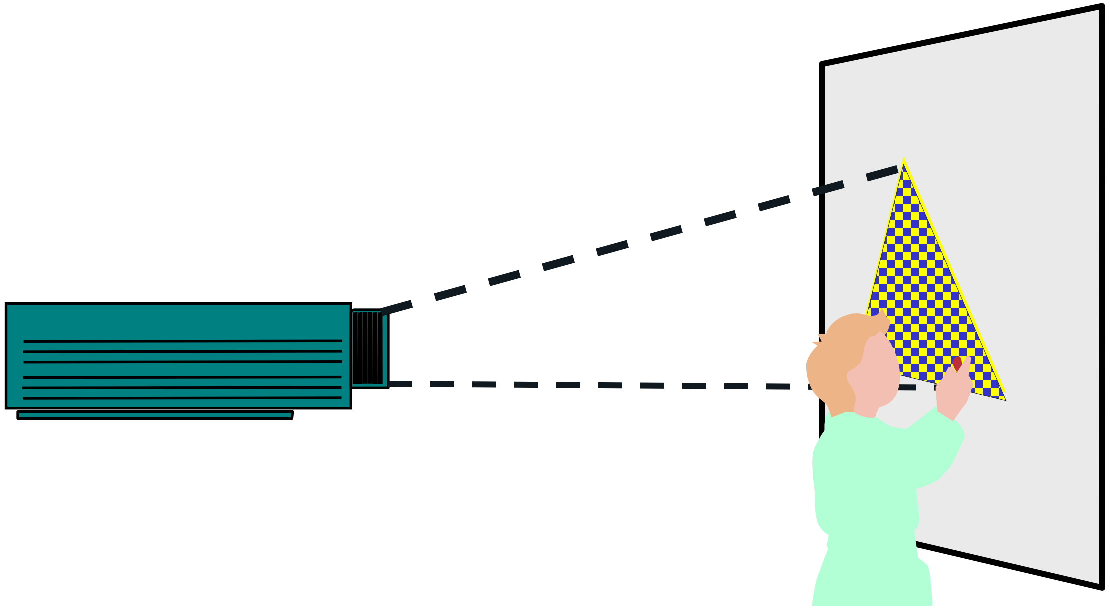
Imagine that this projector is the software you are testing.

Traditional Automated Testing



Typically, testers automate by creating static scripts.

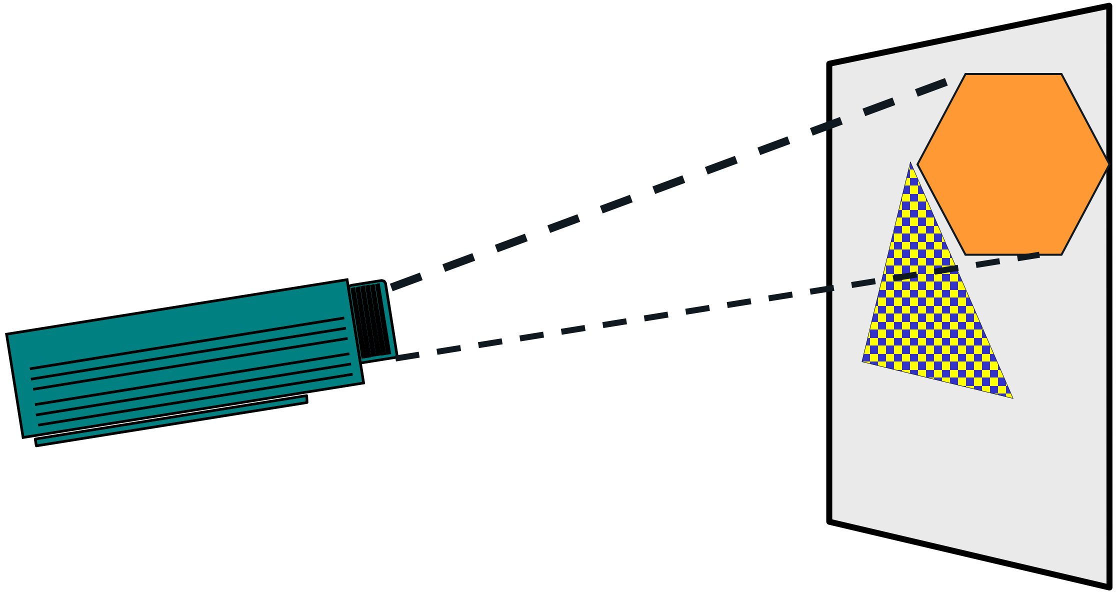
Traditional Automated Testing



Given enough time, these scripts will cover the behavior.

EDITH COWAN UNIVERSITY

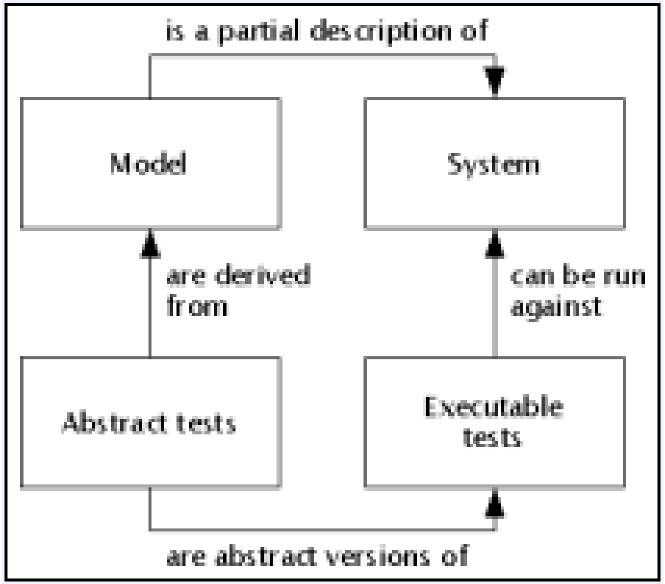
Traditional Automated Testing



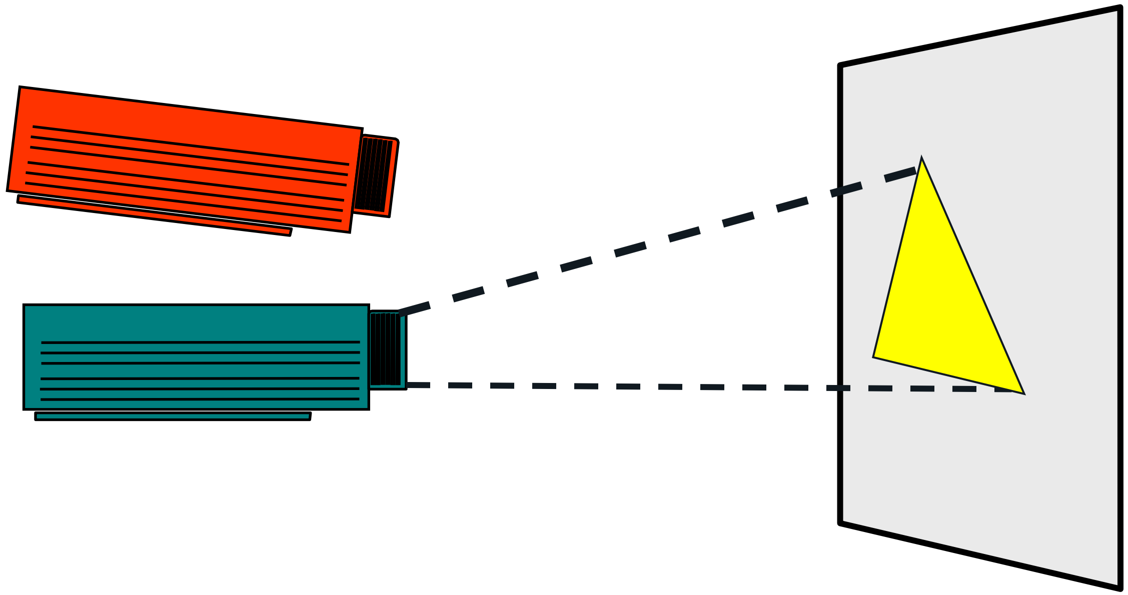
But what happens when the software's behavior changes?

Model-Based Testing

One way to generate test cases automatically is “model-based testing” where a model of the system is used for test case generation.

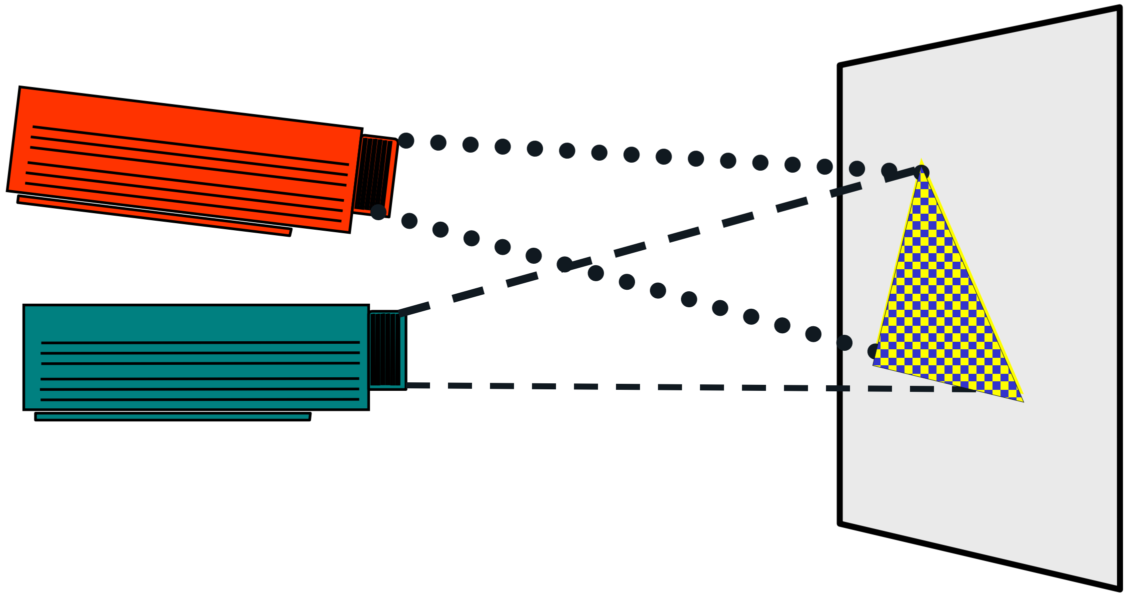


Model-Based Testing



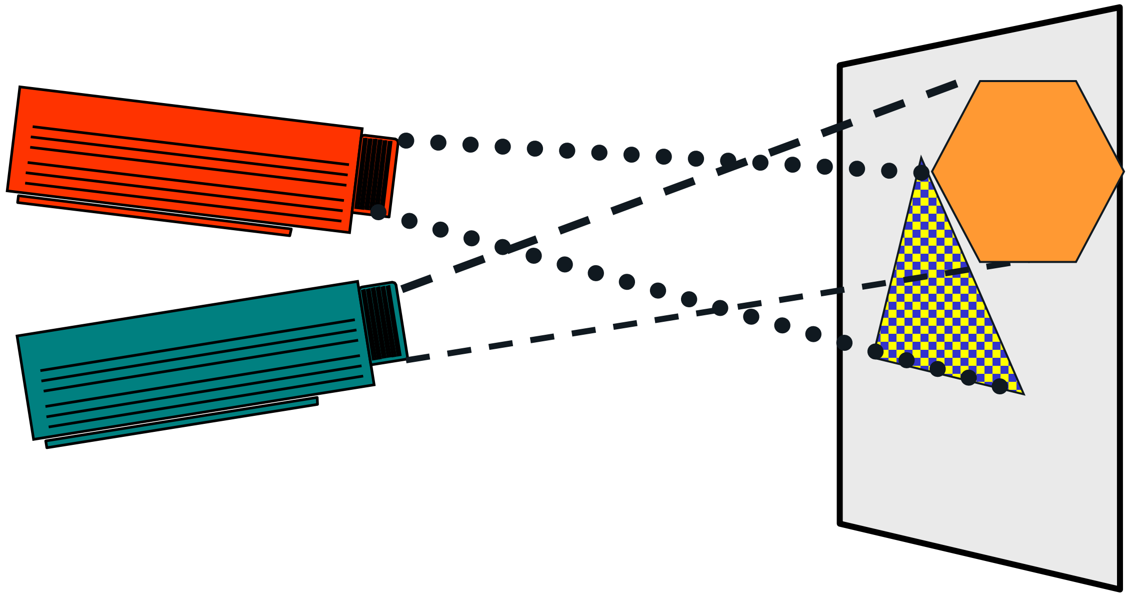
Now, imagine that the top projector is your model.

Model-Based Testing



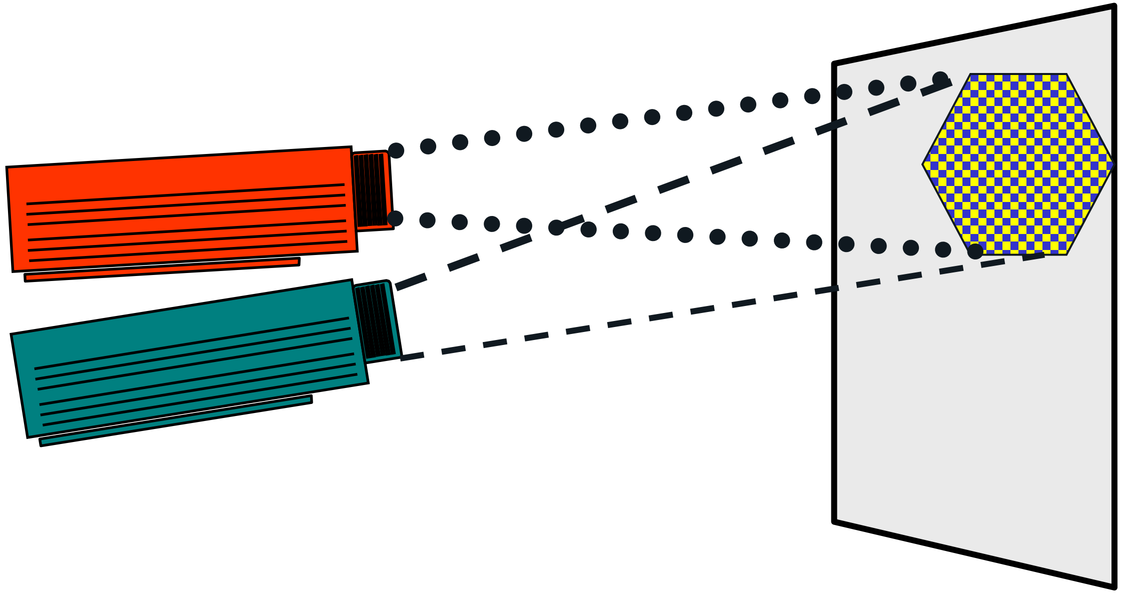
The model generates tests to cover the behavior.

Model-Based Testing



… and when the behavior changes…

Model-Based Testing



… so do the tests.

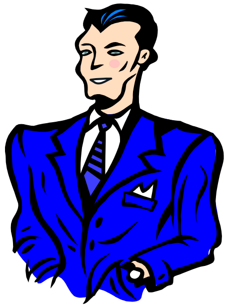
EDITH COWAN UNIVERSITY

CSI3105: Software Testing

Module 9d: Types of Models

Creative thinkers made here.

So What's a Model?



- A model is a description of a system's behavior.
- Models are simpler than the systems they describe.
- Models help us understand and predict the system's behavior.

Model-Based Testing

“Model-based testing is a testing technique where the **runtime behavior** of an implementation under test is **checked against** predictions made by a formal specification, or **model**.” - Colin Campbell, Microsoft Research

In other words:

- A model describes how a system should behave in response to an action.
- Supply the action and see if the system responds as you expect.

Models

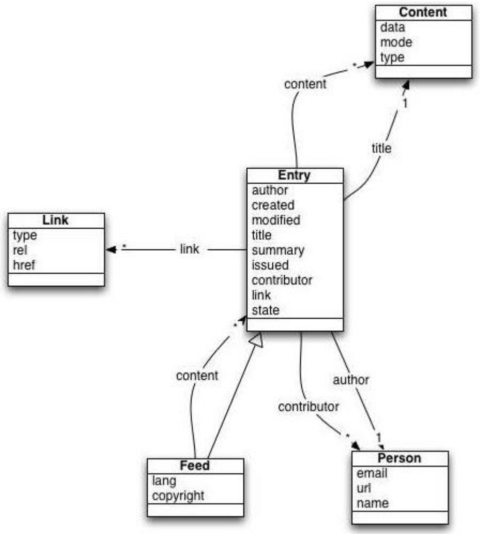
Simply, a model of software is a depiction of its behavior. Behavior can be described in terms of the

- input sequences accepted by the system,
- the actions,
- conditions,
- output logic, or the flow of data through the application’s modules and routines.

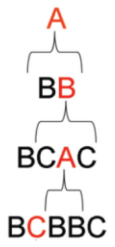
By using modeling -- a shareable, reusable, precise description of the system is acquired.

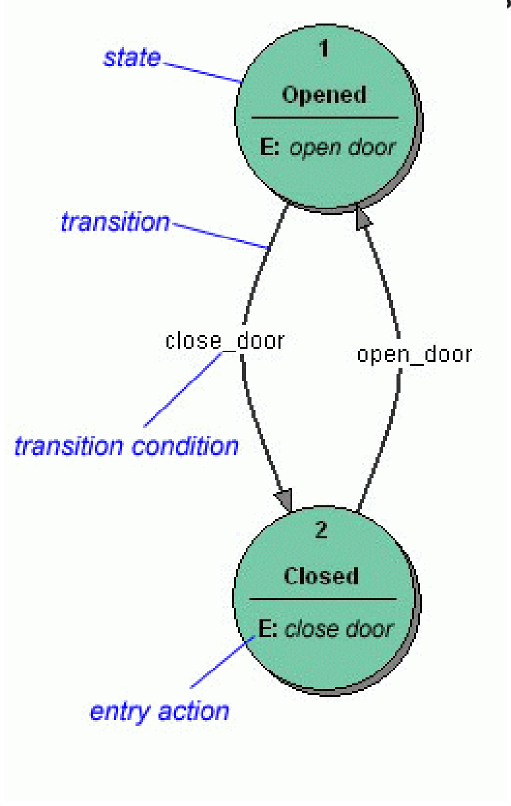
Models in SW Testing

- Finite State Machines
- Statecharts
- UML
- Markov Chains
- Grammars



$A \rightarrow BB$
$B \rightarrow CAC$
$C \rightarrow DD$





EDITH COWAN UNIVERSITY

CSI3105: Software Testing

Module 9e: Examples

Creative
thinkers
made here.

Finite State Machines

Formally a finite state machine representing a software system is defined as a quintuple *(I, S, T, F, L)*, where

- *I* is the set of **inputs** of the system.
- *S* is the set of all **states** of the system.
- *T* is a function that determines whether a **transition** occurs when an input is applied to the system in a particular state.
- *F* is the set of **final states** the system can end up in when it terminates.
- *L* is the state into which the software is **launched**.

Example: Light Switch

Consider a simple light switch simulator:

The lights can be turned on and off using one input.

The intensity of the light can be adjusted using two inputs for decreasing and increasing the intensity.

There are three levels of light intensity: dim, normal, and bright.

If the lights are bright, increasing the intensity should not affect the intensity.

The simulator starts with the lights off. Finally, when the lights are

turned on, the intensity is normal by default, regardless of the intensity of the light when it was last turned off.

Example: Light Switch

“Light Switch” = (I, S, T, F, L), where:

I(inputs) = {<turn on>, <turn off>, <increase intensity>, <decrease intensity>}

S(states) = {[off], [dim], [normal], [bright]}

T (transition function):

<turn on> changes [off] to [normal]

<turn off> changes any of [dim], [normal], or [bright] to [off]

<increase intensity> changes [dim] to [normal], or [normal] to [bright]

<decrease intensity> changes [bright] to [normal], or [normal] to [dim]

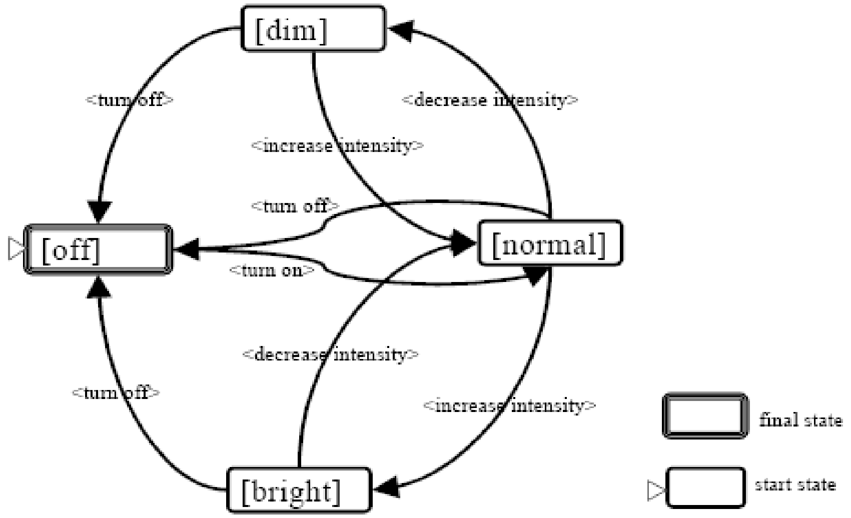
The inputs do not affect the state of the system under any condition not described above

F(final states) = [off]

L(initial state/ launched state) = [off]

State Transition Matrix And State Transition Diagram

State \ Input	<turn on>	<turn off>	<increase intensity>	<decrease intensity>
[off]	[normal]			
[dim]		[off]	[normal]	
[normal]		[off]	[bright]	[dim]
[bright]		[off]		[normal]



Fundamental tasks of MBT

Understanding The System: Forming a mental representation of the system's

functionality is a prerequisite to building models. This is a nontrivial task as most systems today typically complex functionality. Moreover, software is deployed within gigantic operating systems among a clutter of other applications, dynamically linked libraries, and file systems all interacting with and/or affecting it in some manner.

- *Determine the components/features that need to be tested based on test objectives.*
- *Start exploring target areas in the system.*
- *Establish communication with requirements, design, and development teams if possible.*
- *Identify the users of the system.*
- *Enumerate the inputs and outputs of each user.*
- *Study the domains of each input.*
- *Document input applicability information.*

Fundamental tasks of MBT(cont'd)

- *Document conditions under which responses occur.*
- *Study the sequences of inputs that need to be modeled.*
- *Understand internal data interactions and computation.*
- *Maintain one living document.*

Choosing the Model:

Not surprisingly, there are no software models today that fit all intents and purposes. Consequently, for each situation decisions need to be made as to what model (or collection of models) are most suitable.

- HTML files for a web browser or mathematical expressions for a scientific calculator: **Grammar**
- Phone systems etc.: **Finite State Machine**
- Parallel Systems: **Statecharts** (each component is a FSM)
- Systems that have intention of statistical analysis of failure data: **Markov Chain**

Fundamental tasks of MBT(cont'd)

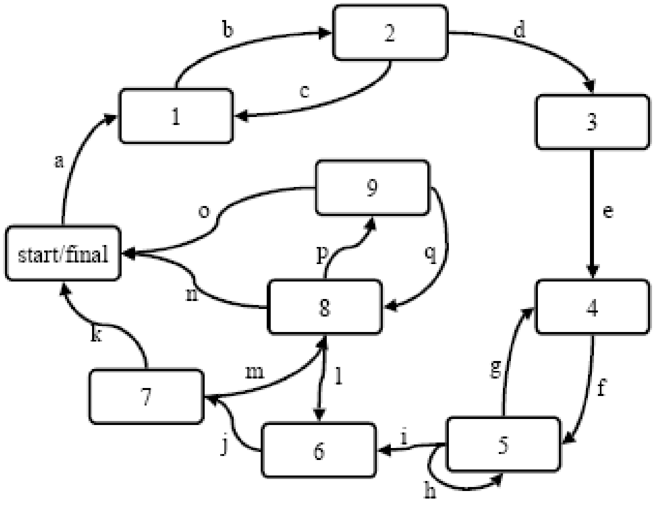
Building the Model:

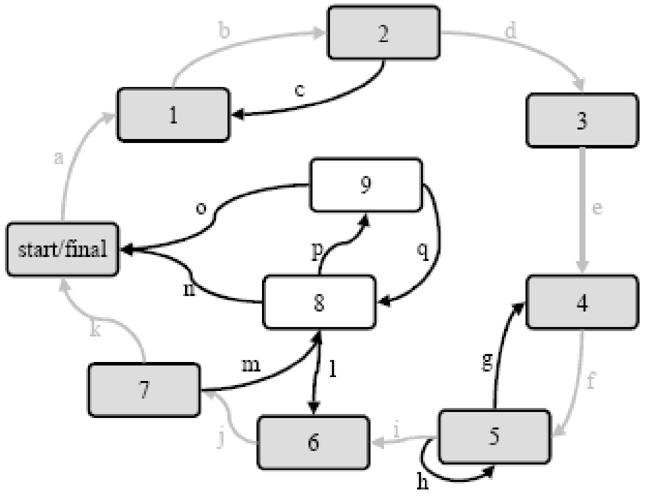
The general procedure that underlies many MBT methodologies:

- Make a list of the inputs.
- ***input applicability constraints*** for each input. For example, consider a simple telephone. The input "take the handset off the hook" can only be applied when the handset is hung up. (It is impossible to pick up the handset if the handset has already been picked up.)
- ***input behavior constraints*** for each input according to the current state context. For example, the input "take the handset of the hook" can cause a number of behaviors. If the context is that the phone is idle, the resulting behavior will be that the system will generate a dial tone. If the context is that the phone is ringing due to an incoming call, then the system will connect the two phones to allow conversation. Two different behaviors are caused by the same input.

Generating Tests

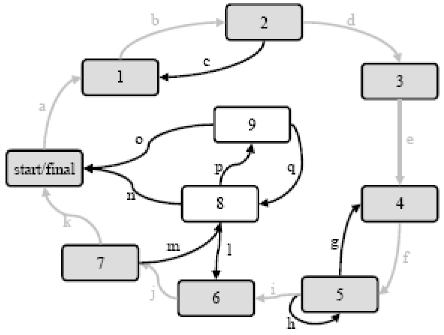
In the case of finite state machines, traverse the state transition diagram. The sequences of arc labels along the generated paths are, by definition, tests. For example, in the state transition diagram below, the sequence of inputs “a, b, d, e, f, i, j, k” qualifies as a test of the represented system.





A state machine (left) and a test path in the state machine (right)

Running Tests



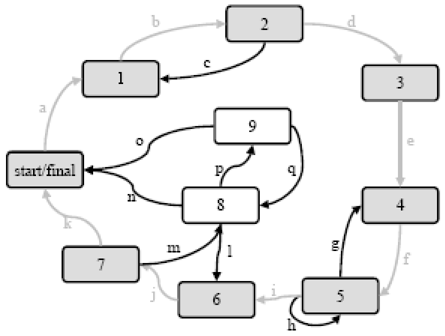
```
Procedure a() {
  // Simulation code for input a
}

Procedure b() {
  // Simulation code for input b
}

Procedure c() {
  // Simulation code for input c
}
.
.
.
```

```
Procedure Test_Script() {
    // Code to drive software
    // into start state
    a();
    b();
    d();
    e();
    f();
    i();
    j();
    k();
}
```

First, test scripts are written to simulate the application of inputs.

Next, the test suite can be easily translated into an executable test script.

Running Tests

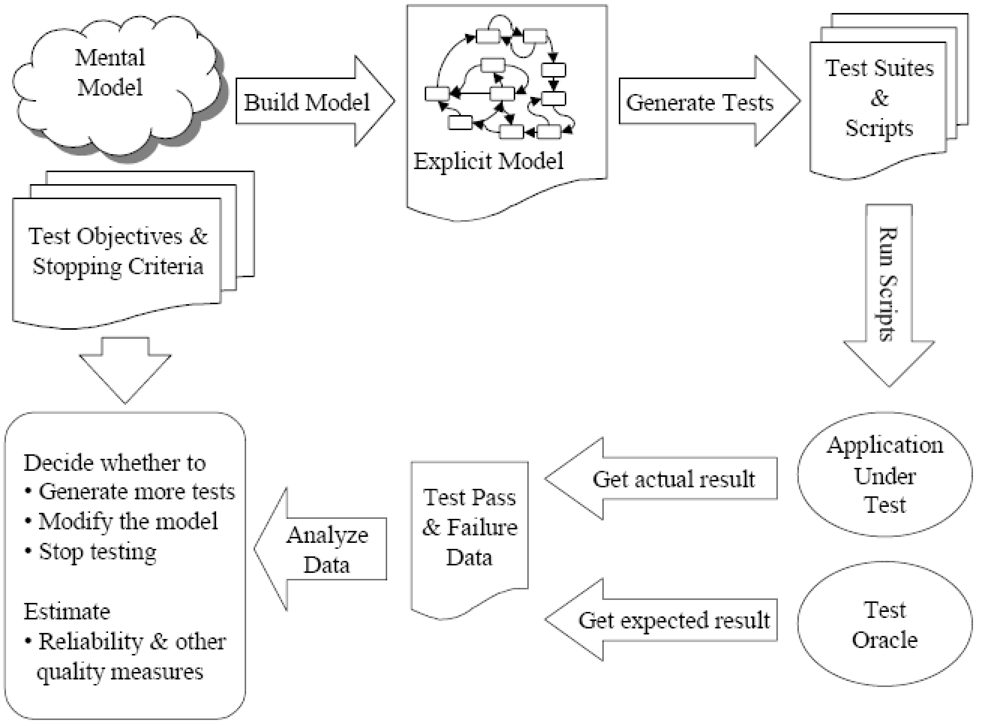


```
Procedure a() {
  // Simulation code for input a
}

Procedure b() {
  // Simulation code for input b
}

Procedure c() {
  // Simulation code for input c
}
.
.
.
```

```
Procedure Test_Script() {
    // Code to drive software
    // into start state
    a();
    b();
    d();
    e();
    f();
    i();
    j();
    k();
}
```

```
// in model
Procedure a(){
            assert( webpage.getTitle(“Staff Page”), Login(valid)));
}

// in real system
Procedure a(){
            assert( webpage.getTitle(“Staff Page”), btnClick(“Login”, “staffTemp”, “hgZfg123”));
}
```

Collecting Results



Key Concerns

State Space Explosion: Number of states can be increased in a dramatical manner.

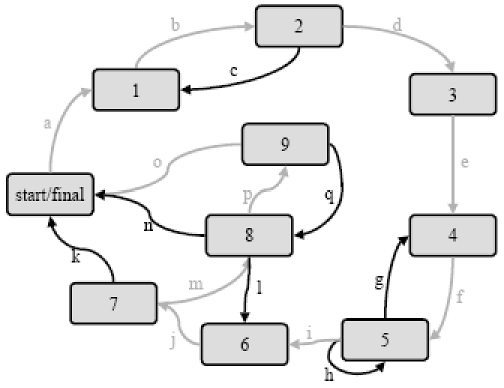
Solutions:

- *Abstraction:* Abstraction for MBT involves the incorporation of complex data into a simple structure. For example a dialog box that requires the entry of multiple fields of data followed by the OK button, could be modeled as two abstract inputs: “enter valid data” and “enter invalid data.”
- *Exclusion: S*imply dropping information without bothering to abstract it. This is mostly done when decomposing a set of software behaviors into multiple models. Each model will contain certain inputs and exclude others. The inputs excluded by a modeler would have to be tested in some other way, obviously.

Key Concerns (cont'd)

Meeting Coverage Criteria

For state machines, coverage is expressed in terms of inputs, states, and transitions. For example, there is the criterion of covering all states.



EDITH COWAN UNIVERSITY

CSI3105: Software Testing

Module 9f: Summary

Creative thinkers made here.

Attractive Attributes of MBT

Succesfull applications when MBT was employed:

- Rosaria and Robinson (2000) on testing graphical user interfaces;
- Agrawal and Whittaker (1993) on testing embedded controller software;
- Avritzer and Larson (1993) and Dalal et al. (1998) on testing phone systems.

A model serves as a **unifying point of reference** that all teams and individuals involved in the development process can share, reuse, and benefit from. For example, confusion as to whether the system under test needs to satisfy a particular requirement can be resolved by examining the model.

Most popular models have a **rich theoretical background** that makes numerous tasks such as generating large suites of test cases easy to automate. For example, graph theory readily solves automated test generation for finite state machine models.

Attractive Attributes of MBT

Reduced cost in testing and early fault detection

Difficulties and Drawbacks of MBT

Skills, time, and other resources need to be allocated for making preparations, overcoming common difficulties, and working around the major drawbacks.

This overhead needs to be weighed against potential rewards in order to determine whether a model-based technique is sensible to the task at hand.

Certain skills of testers (basic familiarity with formal languages, automata theory, and perhaps graph theory and elementary statistics).

The most prominent problem for state models (and most other similar models) is state space explosion. Briefly, models of almost any non-trivial software functionality can grow beyond management even with tool support.

Wrapping up

So lets walk away today a high level definition of what MBT is and what potential models we could use

Generating the actual test cases **is out of the scope** of this unit as it requires us to delve into the formal models.