Software Applications - What Are they?
- enterprise software
- accounting software
- office suites
- graphics software
- media players
- Databases
- Graphical user interfaces
- Web applications or applications that, on server side, communicate with clients via web protocols such as:
  - HTTP/HTTPS
  - WSDL
  - SOAP
- Applications that employ technologies such as:
  - Ajax
  - CSS
  - ASP.Net
  - JavaScript
  - Java EE
Software Applications - What Are they?
- enterprise software
- accounting software
- office suites
- graphics software
- media players
- Databases
- Graphical user interfaces
- Web applications or applications that, on server side, communicate with clients via web protocols such as:
  - HTTP/HTTPS
  - WSDL
  - SOAP
- Applications that employ technologies such as:
  - Ajax
  - CSS
  - ASP.Net
  - JavaScript
  - Java EE
Software application development suffers from a host of issues, including, but not limited to:

- Requirements churn
- Scope creep
- Tight to near impossible deadlines
- Insufficient resources at times - (far too often)
- Increasing functional complexity
- Requirement timeliness
- Requirement ambiguity
- Requirement Error
- Requirement incompleteness
Research into the domain of software development shows that:

- Requirements gathering, analysis and architectural design accounts for between 60% and 70% of all defects introduced into a software product (from studies conducted by Kirova)
- Coding accounts for 30% to 40% of defects discovered in software products (Kirova)
- Up to 80% of all software development time is spent on locating and correcting defects (includes test) (NIST 2002)
Attempts have been made to eliminate or remove error early in the development lifecycle:

- Fagan’s review process has shown under experimental conditions that it is capable of removing 34% of seeded error
- Modelling techniques under similar experimental conditions and with the same errors as seeded in Fagan experiments have shown that the error removal rate was 90%
Traditional testing is challenged by four compounding problems:

- Time and labour intensiveness in handcrafting tests
- Questionable test quality where other than formal techniques are used for test derivation
- Time and resource intensiveness of manually executing/re-executing tests or automating tests via scripting
- Pesticide Paradox (Beizer) - Tests become stale quickly

At TestOptimal we believe the answer is test automation through Model Based Testing
Model Based Testing ensures that there is a very tight coupling between the generated test sequences and the originating requirements.

Models are an abstraction or simplification of the behavior of the application to be tested focused to resolving a particular issue(s).

Many modeling approaches are available.
There are very many modeling techniques that may be applied to testing software, these include:

- Finite State Machines
- Control Flow Graphs
- Binary Search Graphs
- Truth Tables
- Classification Trees
- Decision Tables
- Equivalence Classes
- Data Flow Models
- Entity Relationship Diagrams
- Message Sequence Charts

Many More Besides
MODEL BASED TESTING OF SOFTWARE APPLICATIONS

Finite State Machine
Cyclomatic Complexity Calculation – Tells us we Have 42 unique paths through this graph so at least 42 Test Cases

Control Flow Graph
If we could harness the potential of model based testing with some form of automation then testing would be in a more powerful place to deal with the issues presented by advanced and advancing Software Applications.
Generating models in a machine readable and executable format gives rise to the potential for comprehensive test automation on a massive scale
One Model Based Testing architecture - is Offline with Oracle:
MODEL BASED TESTING OF SOFTWARE APPLICATIONS

- On-The-Fly Architecture
Model Based Testing requires generating models in machine readable format

A few frameworks exist to support model based testing:

- **Nmodel - C#, .Net environment, Finite State Machine approach** - heavy on coding, not easily assimilated into test teams

- **Spec. Explorer - Integral to Visual Studio 2010, .Net, Finite State Machine approach** (not yet a practical solution look for it into the future)

- **ConformIQ from Qtronic (Eclipse® based tool to automate the design of functional tests for software and systems)**

- **TestOptimal - browser and Eclipse® based, XML style scripting language supported by built in Java, C#.Net, Selenium and SQL methods, Finite State Machine approach** - low on coding high on output
Regardless of which approach or framework you adopt Model Based Testing requires some unique skill sets:

- Understanding of Finite State Machines as a form of formal requirements modelling and test derivation - **this is fundamental**
- Ability to abstract detail away without removing the substance of the problem
- Ability to design and code - models consume both design and code
Generating models doesn’t come for free

- Modelling/coding commences with requirements analysis, continues during and keeps pace with application development and launches almost immediately upon build delivery
- While generating models/code no “tests” appear, traditional handcrafting looks to lead in this regard (a big mistake to believe this)
- When models are complete the number of tests that can be generated are only limited by the constraints we place on the model
- The speed of generating (and executing tests when coupled to a framework) is phenomenal
- Ability to update tests is rapid by comparison to traditional means (typically under an hour for full regeneration - ready for re-execution)
You will quickly come to appreciate that:

- Model Based Testing is more about software development for testing than about individual test creation. This is important to recognise.
- You cannot view model based testing as just another exercise in testing. You must manage and control your activities and deliverables just as you would manage or control software development and artefacts for in deed you are developing software.
- You must not reconcile model based test output with numbers of test cases you may however reconcile requirements covered, states covered, transitions covered
- Management needs to be on-board and supportive, without this support only failure awaits
To setup a Model Based Testing environment
Think about:
- The people
- The skills to service the framework you adopt
- The projects
- The circumstances that you deploy Model Based Testing On
- Again this is not a standard testing exercise, this is a software development exercise for the purpose of highly adaptive, highly responsive and exceptionally comprehensive testing
To start building models to create your Model Based Testing you start with Finite State Machine representation of your application area of focus.
Finite State Machines (FSM) what are they?:

- In FSM representation we consider the Application Under Test (AUT) in terms of its States (however we decide to visualize them) and those actions (triggers - the transitions) that cause State change.
- Consider a State as an outward representation of an AUT’s behavior - depicted in the case of a web application for example by a page or tab of features within these pages.
- The “F” (Finite) in FSM, merely reflects the limited (non-infinite) number of States that represent the totality of the AUT or in the case of a web application perhaps the limited number of pages/tabs/screens.
A few simple rules to follow to construct an FSM for an application

- Take one view (or perspective) of the application to start
- Each page/screen of the application can be viewed or equated with a State/sub-state of the application
- Every action that alters or changes the page of the application in a way that you care about results in a State change and each such action or trigger/event can be equated with a Transition for the purposes of the model.
Finite State Machine
MODEL BASED TESTING OF SOFTWARE APPLICATIONS

A Log-On Screen (default size 425 pixels by 300 pixels)

1 Enter Invalid Login Details and Press OK button

2 File > Log On

3 Enter Valid Login Details and Press OK button

4 Launch APP application

53 Resize Window

5 Logged On

47 Cancel button

4 Log Off

B Exit

C Logged On

D Logged Off

E Help View

45 Close

46 Help > About

49 Check "Process" Checkbox

One "Process Checkbox Checked, "Prox Button enabl

Und "Prox Chec

F Payment S

G (Default view is all "Process" checkbox screen is open to full screen centre name in at

Q

Select Errors

22

23

5
Begin Modeling from a purely abstract point of view:

- Early in model development **ignore** the AUT (unless it is legacy)- you DON’T need to have access to the actual AUT, you can build models directly from requirements

- Do NOT build one large amorphous model to represent your application. To do so is to invite disaster and it breaks with the concept of abstraction

- Break down the application by logically grouping closely related or interdependent features and model those

- Don’t be afraid to have small models, they best describe discrete behavior - small **IS** good. Many small **IS** better
Abstract models:

- Are purely a representation of the AUT derived from requirements (or other knowledge)
- Utilize “abstract” names within the model code (script) to represent the actual AUT elements that you wish to interact with
- Never use hard coded values for any parameters within your models, always parameterize these values out and retain actual values in external files
- Make sure requirements are traced through to the individual models you build, you need to know what your models are covering when executing
Concretizing models is the next step. At some point you will gain access to the AUT, at this point you are in a position to begin concretizing your models which means:

- You can begin to derive “concrete” or real values for each and every element implemented within the AUT that you care about.
- You associate the “concrete” values with the “abstract” values you incorporated in your models to this time.
- You provide your models with a real path to the AUT such that your models can reach and interact with the AUT.
We need to capture or integrate our model within a modelling framework that will permit the model to:

- exist in a machine readable format
- provide for use of graph traversal algorithms to generate test sequences
- Provide an execution and reconciliation bench

TestOptimal from TestOptimal LLC

www.testoptimal.com
MODEL BASED TESTING OF SOFTWARE APPLICATIONS

Build models within IDE

Directly interface With AUT via Selenium

Interface to and work with external files; e.g. external param. files
Example of a parameterized model script

```xml
<state id="NotLoggedIn">
  <script type="onentry"/>
  <script type="onexit"/>
  <transition event="UserLogin">
    <script type="prep"/>
    <script type="action">
      <log message="Logging into Home from Login"/>
      <action code="$type($getData('ElementNamesDS', 'UserNameLocator'), $getData('GoodLoginDS', 'UserName'))"/>
      <action code="$type($getData('ElementNamesDS', 'PasswordLocator'), $getData('GoodLoginDS', 'Password'))"/>
      <action code="$getCheckBox($getData('ElementName DS', 'TermsOfUseCheckBox'), 'true')"/>
      <action code="$click($getData('ElementNamesDS', 'LoginButton'))"/>
      <action code="$sleep($getData('TextDS', 'SleepTime'))"/>
      <action code="$sleep($getData('TextDS', 'SleepTime'))"/>
    </script>
  </transition>
</state>
```
In order to acquire the identifiers for the AUT elements that must necessarily deal with you will require to use FireFox and the following Add-ons:

- DOM Inspector
- FireBug
- FireXpath
- Xpather
- Xpath Checker
- UI Spy
Additional support to make modelling of applications possible is required in the form of:

- Explicitly created/declared element ids. No non-specific element id’s, for example id_255 or a3425h9989098876788 etc. This sort of non-descriptive element ID tends to cause problems:
Everything you need to interact with, identify, provide input for, read out from needs to have an identifier, this may be at least one of the following:

- Explicit handle
- Windows automation ID
- Element ID
- Xpath
- Attribute
- Link (href)

Without at least a stable form of one of the latter for each of your elements or attributes of interest there is no way to programatically work with the AUT.
Repurposing is one of the great benefits of modeling especially from within an framework such as TestOptimal: All models can with minimal effort be employed for:

- Load testing. You can imagine that launching a model on multiple threads can provide a constant load to your app or web server.
- Stress testing by utilising for example “searching”, updating, purchasing, copying models (as many as you have created) to push your:
  - Db server
  - App server
MODEL BASED TESTING OF SOFTWARE APPLICATIONS

- Questions?