MODEL BASED TESTING
OF SOFTWARE APPLICATIONS

An Overview

MODEL BASED TESTING OF SOFTWARE APPLICATIONS

- 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

MODEL BASED TESTING OF SOFTWARE APPLICATIONS

- 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

The list is near endless

MODEL BASED TESTING OF SOFTWARE APPLICATIONS

- 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

MODEL BASED TESTING OF SOFTWARE APPLICATIONS

- 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)

MODEL BASED TESTING OF SOFTWARE APPLICATIONS

- 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%

MODEL BASED TESTING OF SOFTWARE APPLICATIONS

- 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 OF SOFTWARE APPLICATIONS

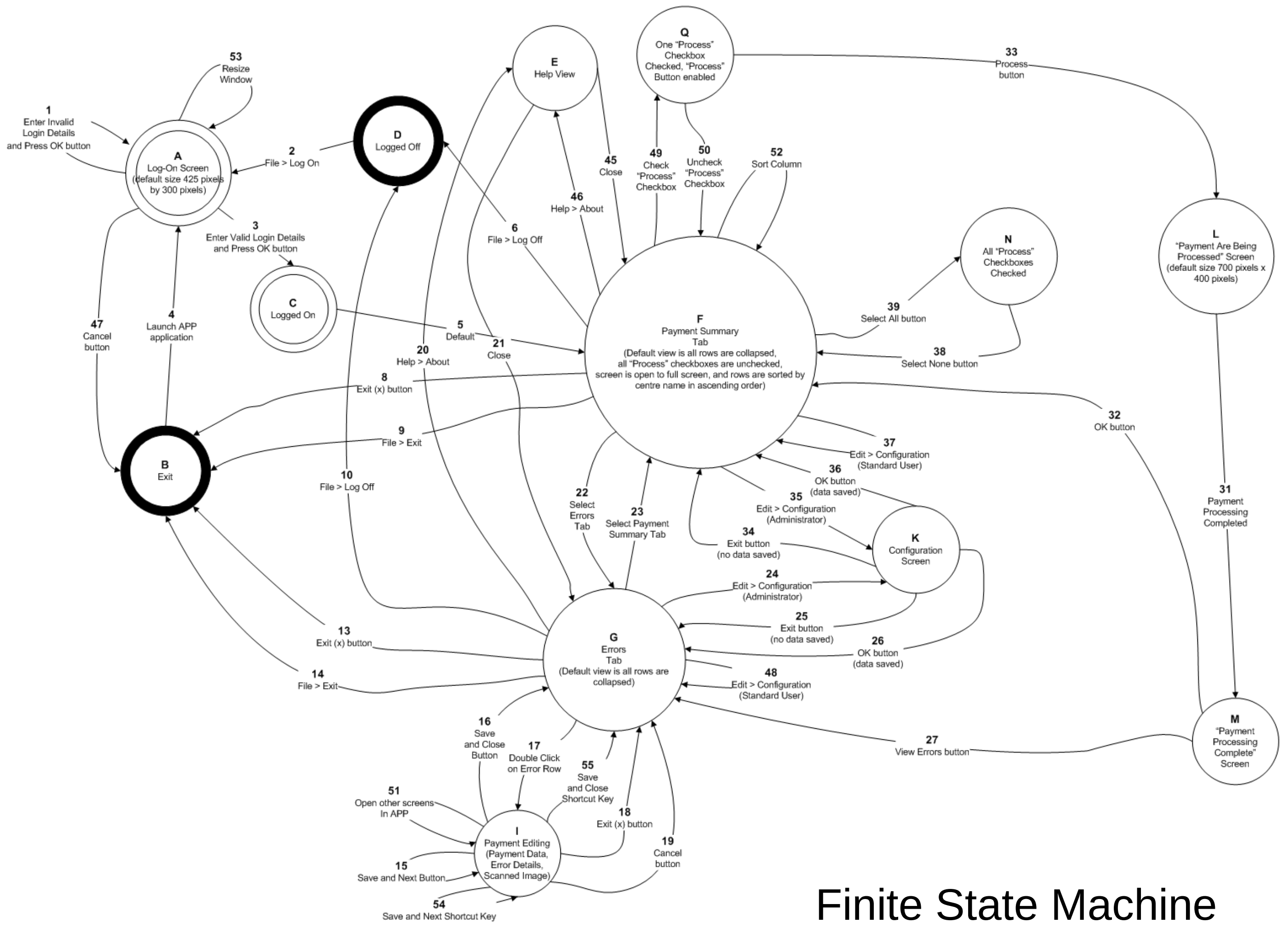
- Model Based Testing ensures that there is a very tight coupling between the generated test sequences and the originating requirements
- Models are an **abstraction** or **simplification** of the behavior of the application to be tested focused to resolving a particular issue(s)
- Many modeling approaches are available

MODEL BASED TESTING OF SOFTWARE APPLICATIONS

- 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

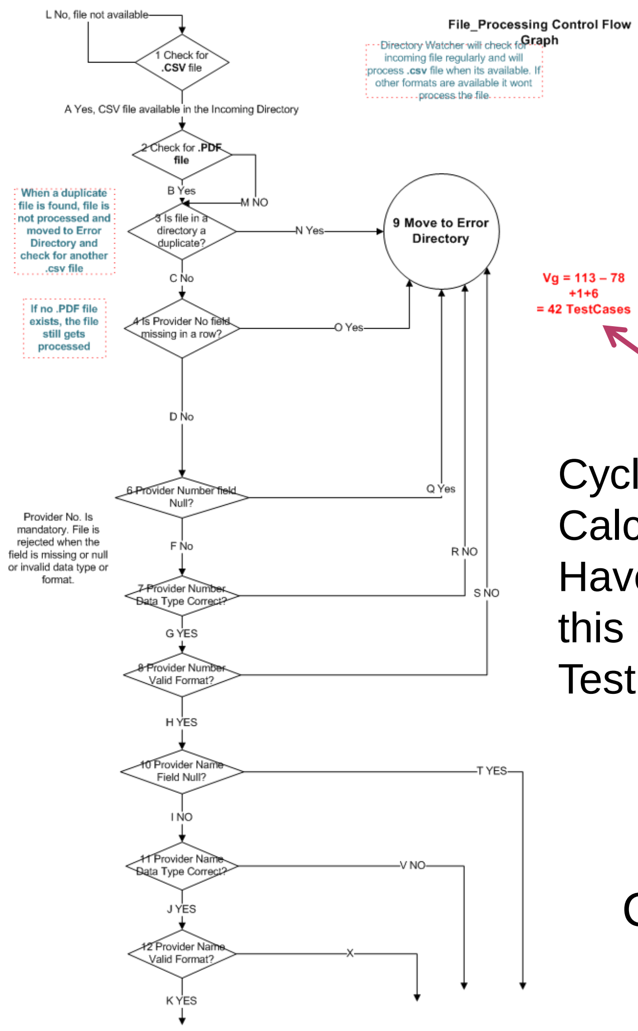
Many More Besides

Model Based Testing of Software Applications


A Log-On Screen (default size 425 pixels by 300 pixels)
B Exit
C Logged On
D Logged Off
E Help View
F Payment Summary Tab (Default view is all rows are collapsed, all "Process" checkboxes are unchecked, screen is open to full screen, and rows are sorted by centre name in ascending order)
G Errors Tab (Default view is all rows are collapsed)
I Payment Editing (Payment Data, Error Details, Scanned Image)
K Configuration Screen
L "Payment Are Being Processed" Screen (default size 700 pixels x 400 pixels)
M "Payment Processing Complete" Screen
N All "Process" Checkboxes Checked
Q One "Process" Checkbox Checked, "Process" Button enabled
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
5 Default
6 File > Log Off
8 Exit (x) button
9 File > Exit
10 File > Log Off
13 Exit (x) button
14 File > Exit
15 Save and Next Button
16 Save and Close Button
17 Double Click on Error Row
18 Exit (x) button
19 Cancel button
20 Help > About
21 Close
22 Select Errors Tab
23 Select Payment Summary Tab
24 Edit > Configuration (Administrator)
25 Exit button (no data saved)
26 OK button (data saved)
27 View Errors button
31 Payment Processing Completed
32 OK button
33 Process button
34 Exit button (no data saved)
35 Edit > Configuration (Administrator)
36 OK button (data saved)
37 Edit > Configuration (Standard User)
38 Select None button
39 Select All button
45 Close
46 Help > About
47 Cancel button
48 Edit > Configuration (Standard User)
49 Check "Process" Checkbox
50 Uncheck "Process" Checkbox
51 Open other screens In APP
52 Sort Column
53 Resize Window
54 Save and Next Shortcut Key
55 Save and Close Shortcut Key

Finite State Machine

MODEL BASED TESTING OF SOFTWARE APPLICATIONS


File_Processing Control Flow Graph
Directory Watcher will check for incoming file regularly and will process .csv file when its available. If other formats are available it wont process the file.
L No, file not available
1 Check for .CSV file
A Yes, CSV file available in the Incoming Directory
2 Check for .PDF file
B Yes
M NO
When a duplicate file is found, file is not processed and moved to Error Directory and check for another .csv file
3 Is file in a directory a duplicate?
N Yes
9 Move to Error Directory
C No
If no .PDF file exists, the file still gets processed
4 Is Provider No field missing in a row?
O Yes
D No
6 Provider Number field Null?
Q Yes
Provider No. Is mandatory. File is rejected when the field is missing or null or invalid data type or format.
F No
R NO
7 Provider Number Data Type Correct?
S NO
G YES
8 Provider Number Valid Format?
H YES
10 Provider Name Field Null?
T YES
I NO
11 Provider Name Data Type Correct?
V NO
J YES
12 Provider Name Valid Format?
X
K YES

Vg = 113 – 78 +1+6 = 42 TestCases

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

Control Flow Graph

MODEL BASED TESTING OF SOFTWARE APPLICATIONS

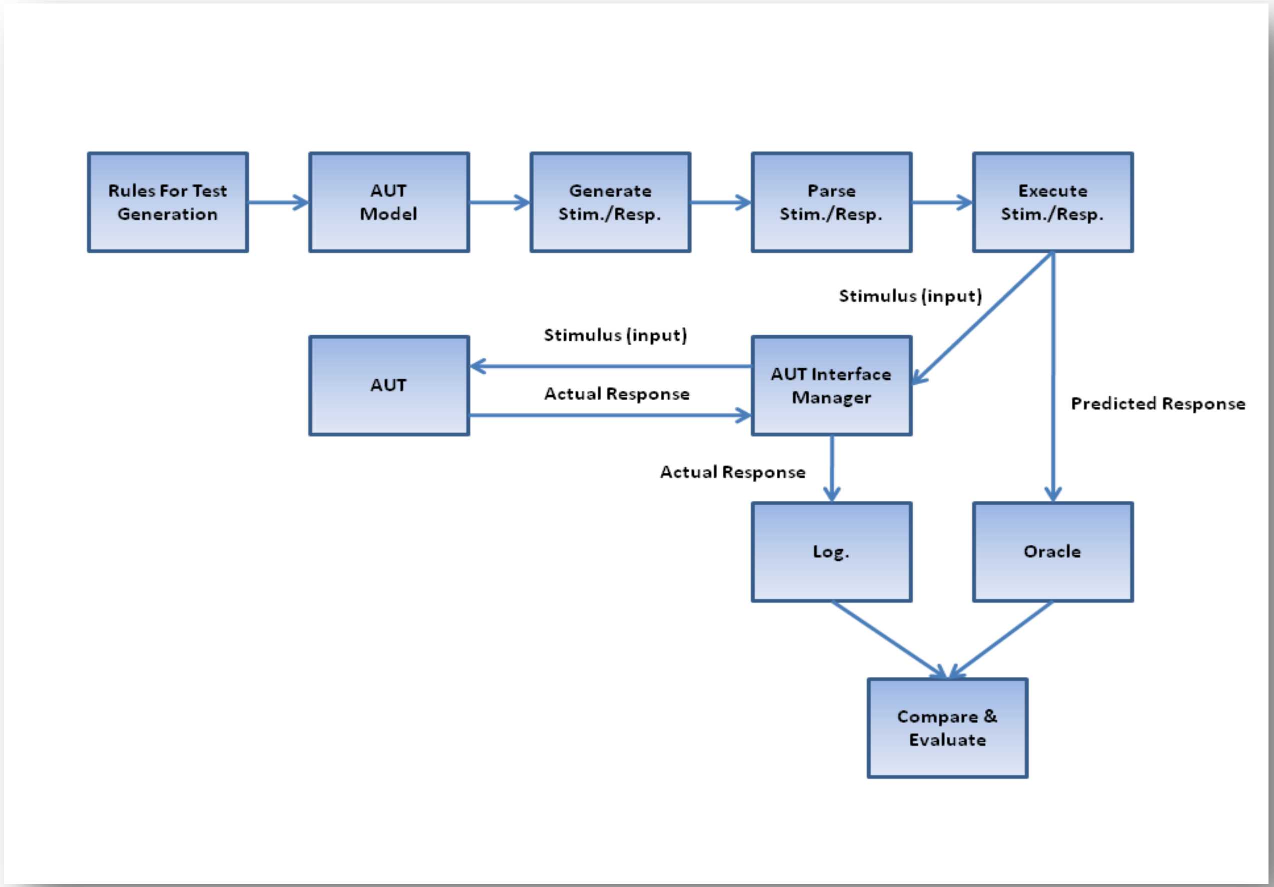
- If we could harness the potential of model based testing with some form of automation then testing would be in a more powerful place to deal with the issues presented by advanced and advancing Software Applications.

MODEL BASED TESTING OF SOFTWARE APPLICATIONS

- Generating models in a machine readable and executable format gives rise to the potential for comprehensive test automation on a massive scale

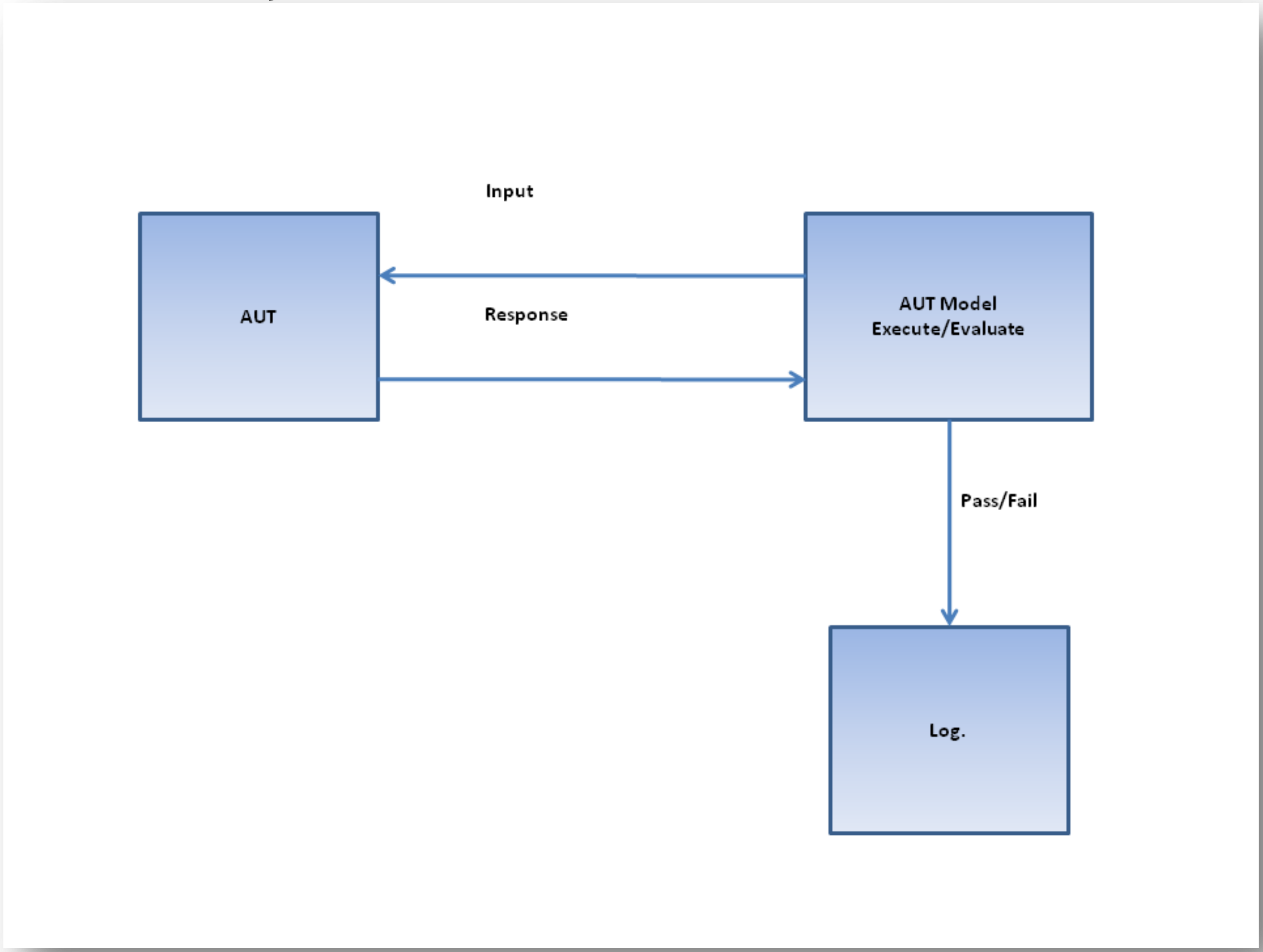
MODEL BASED TESTING OF SOFTWARE APPLICATIONS

- One Model Based Testing architecture - is Offline with Oracle:



MODEL BASED TESTING OF SOFTWARE APPLICATIONS

- On-The-Fly Architecture



MODEL BASED TESTING OF SOFTWARE APPLICATIONS

- 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 inJava, C#.Net, Selenium and SQL methods, Finite State Machine approach – low on coding high on output

MODEL BASED TESTING OF SOFTWARE APPLICATIONS

- 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 – **<u>this is fundamental</u>**
  - Ability to abstract detail away without removing the substance of the problem
  - Ability to design and code – models consume both design and code

MODEL BASED TESTING OF SOFTWARE APPLICATIONS

- 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)

MODEL BASED TESTING OF SOFTWARE APPLICATIONS

- 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 reconcil requirements covered, states covered, transitions covered
  - Management needs to be on-board and supportive, without this support only failure awaits

MODEL BASED TESTING OF SOFTWARE APPLICATIONS

- 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

MODEL BASED TESTING OF SOFTWARE APPLICATIONS

- To start building models to create your Model Based Testing you start with Finite State Machine representation of your application area of focus

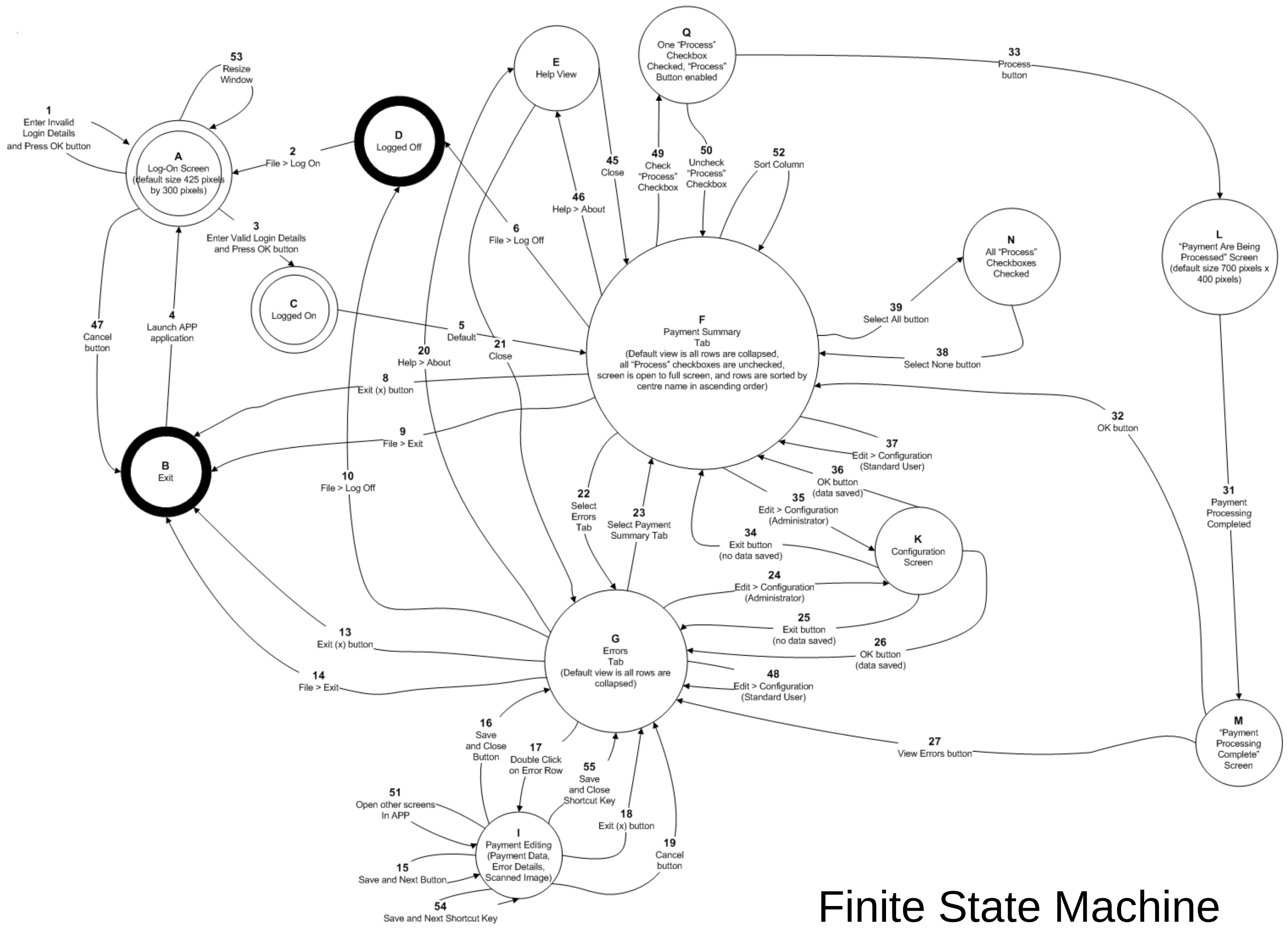
MODEL BASED TESTING OF SOFTWARE APPLICATIONS

- 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

MODEL BASED TESTING OF SOFTWARE APPLICATIONS

- 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.

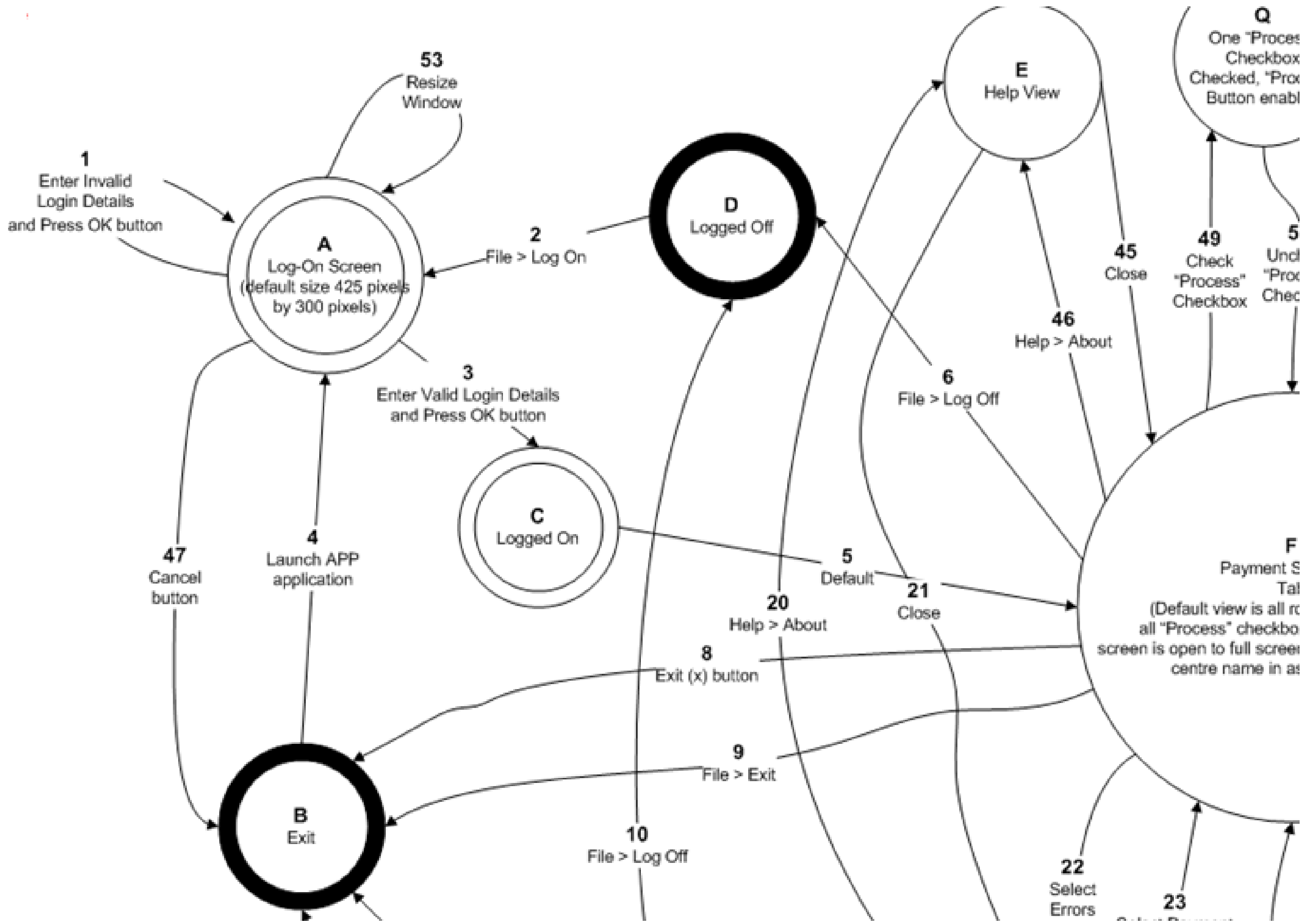
Model Based Testing of Software Applications



A Log-On Screen (default size 425 pixels by 300 pixels)
B Exit
C Logged On
D Logged Off
E Help View
F Payment Summary Tab (Default view is all rows are collapsed, all "Process" checkboxes are unchecked, screen is open to full screen, and rows are sorted by centre name in ascending order)
G Errors Tab (Default view is all rows are collapsed)
I Payment Editing (Payment Data, Error Details, Scanned Image)
K Configuration Screen
L "Payment Are Being Processed" Screen (default size 700 pixels x 400 pixels)
M "Payment Processing Complete" Screen
N All "Process" Checkboxes Checked
Q One "Process" Checkbox Checked, "Process" Button enabled
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
5 Default
6 File > Log Off
8 Exit (x) button
9 File > Exit
10 File > Log Off
13 Exit (x) button
14 File > Exit
15 Save and Next Button
16 Save and Close Button
17 Double Click on Error Row
18 Exit (x) button
19 Cancel button
20 Help > About
21 Close
22 Select Errors Tab
23 Select Payment Summary Tab
24 Edit > Configuration (Administrator)
25 Exit button (no data saved)
26 OK button (data saved)
27 View Errors button
31 Payment Processing Completed
32 OK button
33 Process button
34 Exit button (no data saved)
35 Edit > Configuration (Administrator)
36 OK button (data saved)
37 Edit > Configuration (Standard User)
38 Select None button
39 Select All button
45 Close
46 Help > About
47 Cancel button
48 Edit > Configuration (Standard User)
49 Check "Process" Checkbox
50 Uncheck "Process" Checkbox
51 Open other screens In APP
52 Sort Column
53 Resize Window
54 Save and Next Shortcut Key
55 Save and Close Shortcut Key

Finite State Machine

Model Based Testing of Software Applications



53
Resize Window
1
Enter Invalid Login Details and Press OK button
A
Log-On Screen (default size 425 pixels by 300 pixels)
2
File > Log On
D
Logged Off
3
Enter Valid Login Details and Press OK button
C
Logged On
47
Cancel button
4
Launch APP application
5
Default
B
Exit
E
Help View
Q
One "Proces Checkbox Checked, "Proc Button enabl
45
Close
49
Check "Process" Checkbox
5
Uncl "Proc Chec
46
Help > About
6
File > Log Off
F
Payment S Tal (Default view is all ro all "Process" checkbo screen is open to full scree centre name in as
20
Help > About
21
Close
8
Exit (x) button
9
File > Exit
10
File > Log Off
22
Select Errors
23

MODEL BASED TESTING OF SOFTWARE APPLICATIONS

- 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

MODEL BASED TESTING OF SOFTWARE APPLICATIONS

- 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

MODEL BASED TESTING OF SOFTWARE APPLICATIONS

- 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

MODEL BASED TESTING OF SOFTWARE APPLICATIONS

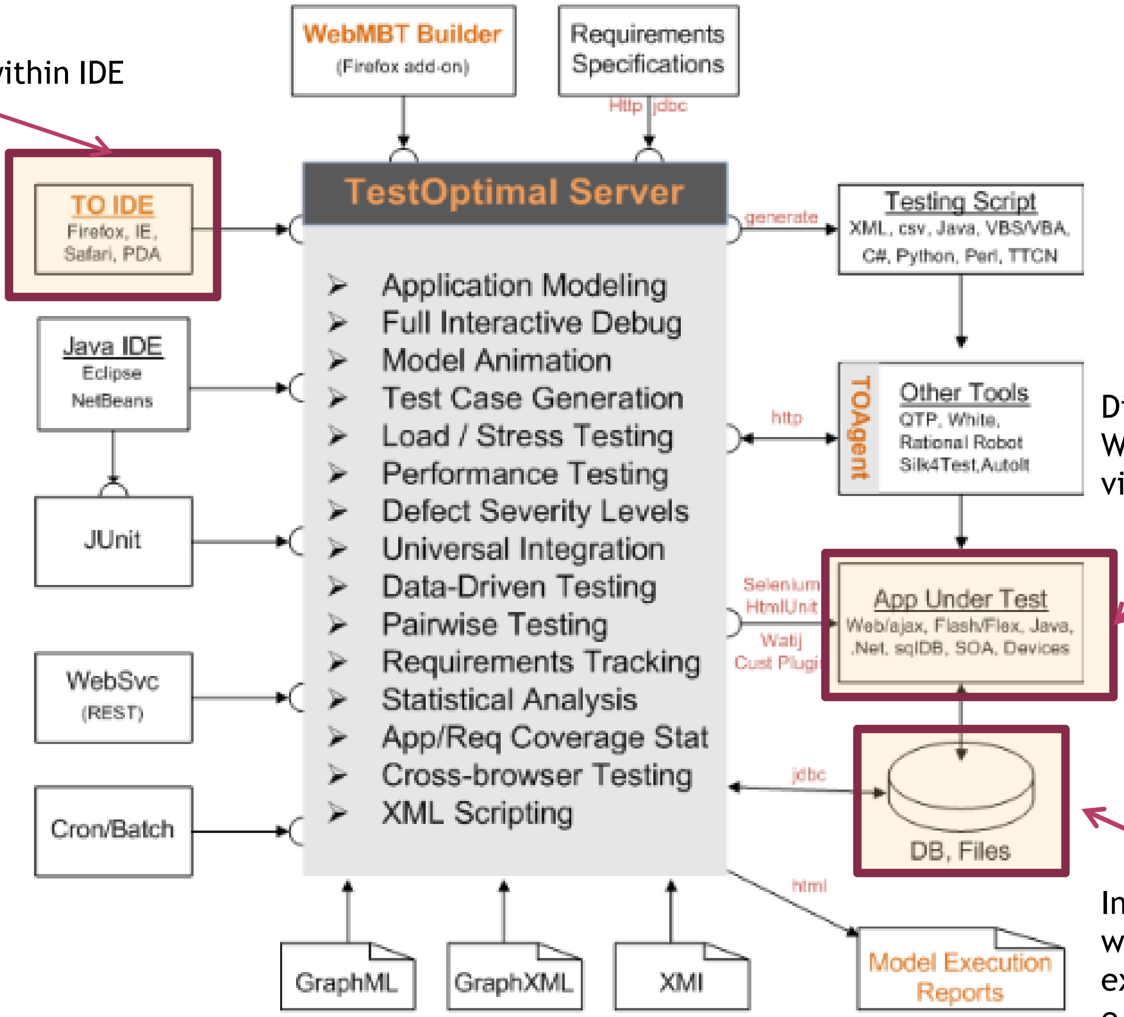
- We need to capture or integrate our model within a modelling framework that will permit the model to:
  - exist in a machine readable format
  - provide for use of graph traversal algorithms to generate test sequences
  - Provide an execution and reconciliation bench

TestOptimal from TestOptimal LLC

www.testoptimal.com

MODEL BASED TESTING OF SOFTWARE APPLICATIONS

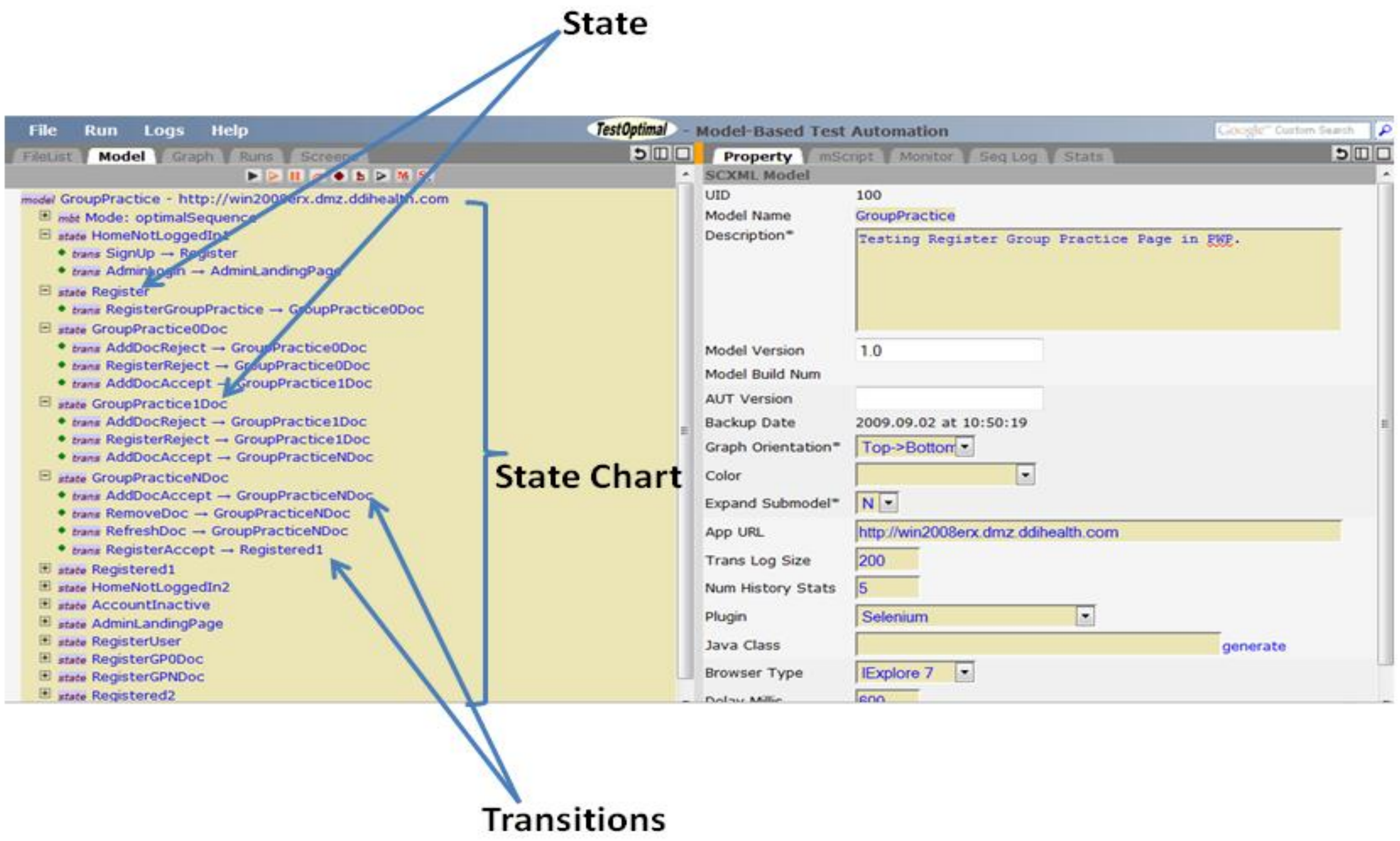
Build models within IDE



Directly interface
With AUT
via Selenium

Interface to and
work with
external files;
e.g. external
param. fles

MODEL BASED TESTING OF SOFTWARE APPLICATIONS


State
File Run Logs Help
TestOptimal - Model-Based Test Automation
FileList Model Graph Runs Screens
Property mScript Monitor Seq Log Stats
State Chart
Transitions

```
model GroupPractice - http://win2008erx.dmz.ddihealth.com
  mbt Mode: optimalSequence
  state HomeNotLoggedIn1
    trans SignUp → Register
    trans AdminLogin → AdminLandingPage
  state Register
    trans RegisterGroupPractice → GroupPractice0Doc
  state GroupPractice0Doc
    trans AddDocReject → GroupPractice0Doc
    trans RegisterReject → GroupPractice0Doc
    trans AddDocAccept → GroupPractice1Doc
  state GroupPractice1Doc
    trans AddDocReject → GroupPractice1Doc
    trans RegisterReject → GroupPractice1Doc
    trans AddDocAccept → GroupPracticeNDoc
  state GroupPracticeNDoc
    trans AddDocAccept → GroupPracticeNDoc
    trans RemoveDoc → GroupPracticeNDoc
    trans RefreshDoc → GroupPracticeNDoc
    trans RegisterAccept → Registered1
  state Registered1
  state HomeNotLoggedIn2
  state AccountInactive
  state AdminLandingPage
  state RegisterUser
  state RegisterGP0Doc
  state RegisterGPNDoc
  state Registered2
```

SCXML Model

UID	100
Model Name	GroupPractice
Description*	Testing Register Group Practice Page in RWP.
Model Version	1.0
Model Build Num	
AUT Version	
Backup Date	2009.09.02 at 10:50:19
Graph Orientation*	Top->Bottom
Color	
Expand Submodel*	N
App URL	http://win2008erx.dmz.ddihealth.com
Trans Log Size	200
Num History Stats	5
Plugin	Selenium
Java Class	generate
Browser Type	IExplore 7

TestOptimal IDE

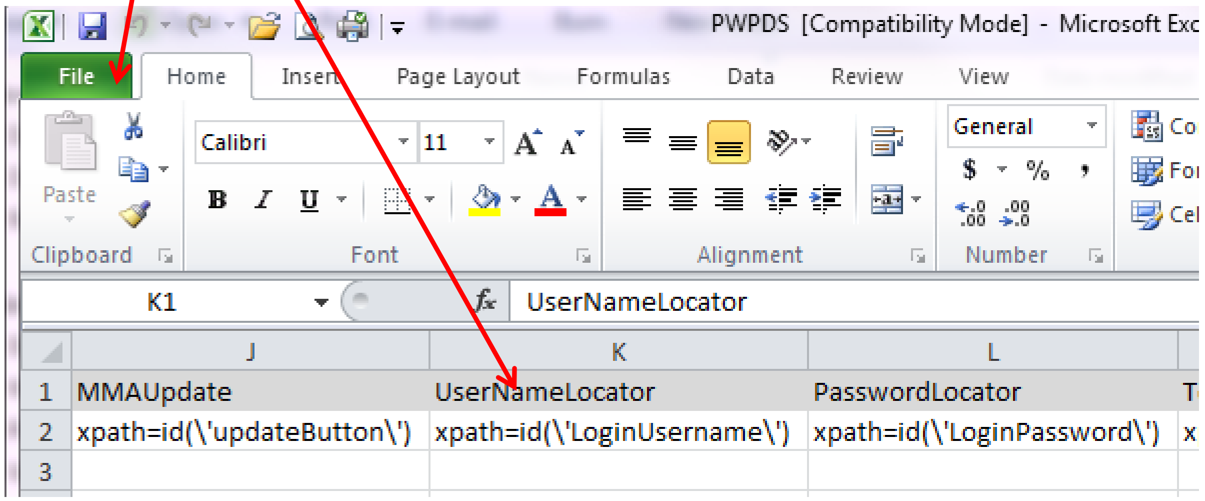
MODEL BASED TESTING OF SOFTWARE APPLICATIONS

- Example of a parameterized model script

```
<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="$setCheckBox('$getData('ElementNamesDS','TermsOfUseCheckBox')','true')"/>
      <action code="$click('$getData('ElementNamesDS', 'LoginButton')')"/>
      <action code="$sleep('$getData('TextDS','SleepTime')')"/>
      <action code="$sleep('$getData('TextDS','SleepTime')')"/>
    </script>
    <script type="verify"/>
  </transition>
</state>
```

Abstract value

Concrete value



K1 f_x UserNameLocator

	J	K	L	T
1	MMAUpdate	UserNameLocator	PasswordLocator	T
2	xpath=id(\'updateButton\')	xpath=id(\'LoginUsername\')	xpath=id(\'LoginPassword\')	x
3				

MODEL BASED TESTING OF SOFTWARE APPLICATIONS

- 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

MODEL BASED TESTING OF SOFTWARE APPLICATIONS

- 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:

MODEL BASED TESTING OF SOFTWARE APPLICATIONS

- 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

MODEL BASED TESTING OF SOFTWARE APPLICATIONS

- 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?