Automation Techniques for Enterprise Application Testing

- Aditya Dada (aditya.dada@sun.com)

Speaker's Qualifications

- Aditya Dada
  - Member Technical Staff, Sun Java Systems Application Server Team, Software Quality Engineer dealing with
    - Application Client Container
    - Deployment
    - Automation
  - Sun Certified Java Developer
  - Sun Certified Business Component Developer

Presentation Goal

Present solutions to tackle issues with automated Enterprise Application Testing.

Presentation Agenda

- Problems with Testing Enterprise Software
  - Typical Problems
- Solution
  - Process
  - Automation
- Sample Implementation
- Important Pointers
- Takeaways
- Q&A

Problems with Testing Enterprise Software

- Common Problems:
  - X-platform & multi-configuration testing is **complex!**
  - Cross-platform and multi-configuration testing needs to be done.
  - Enterprise Applications require integrated components. Waiting means testing delays , raising costs.
  - Bugs need to be caught early
    - Especially when there are > 10000 classes in product code

Typical Problems

- We ensure the quality of Sun Java System Application Server for support on:
  - **Five** databases
  - Multiple versions of Linux, Solaris & Windows
  - Intel, SPARC & AMD architectures
  - Multiple browsers, loadbalancers, webtier modules.
- In all, they add up to over 1500 configurations!

Typical Problems

- Application Server consists of ...
  - Multiple Containers (EJB, Appclient, Web)
  - Multiple Services (Messaging, Transaction, Security)
  - Multiple Components(Connector, deployment, JDBC, JSF, Webservices, High-availability)
- Test applications are complex transactions
  - Test automation covers all the above!

Solutions

- The solution is 2-fold.
  - Software quality engineering process
    - Get requirements for developing, maintaining, executing and reporting tests.
    - Define process ***before*** automating.
  - Automation
    - To make process easy to follow.

Defining the Process

- Get requirements by answering following questions:
  - Which tests will be executed?
  - What will be the test environment?
  - When and how often is it run?
  - What are the guidelines for filing, tracking, and fixing defects?
  - Who will execute tests?
  - Who will monitor and analyse results?
  - Who will monitor issues like hangs, crashes?

Defining the process

- Will Release Eng. execute any set of tests?
- How will regression tests be run & maintained?
- Do we need Basic Acceptance Tests?
- What will be the check-in procedure for development team?
- Will other groups use the test base?
- Will other groups contribute to the test base?
- Will support be added in future for more configurations?

Status Check

- Problems with Testing Enterprise Software
  - Typical Problems
- Solution
  - Process
  - Automation
- Sample Implementation
- Important Pointers
- Takeaways
- Q&A

Defining Automation...

- Zone-in on automation requirements by getting answers to the following questions:
  - Do you need a platform-independent framework?
    - Will multiple platforms be supported in future?
    - Will there be any X-Platform testing?

Defining Automation...

- What services are required of the framework:
  - html/xml/text/other reporting?
  - Diff with previous weeks results?
  - Are there common functions/properties/targets that maybe isolated?
  - Will failed tests need to be re-run?
  - Will there be logging support required?

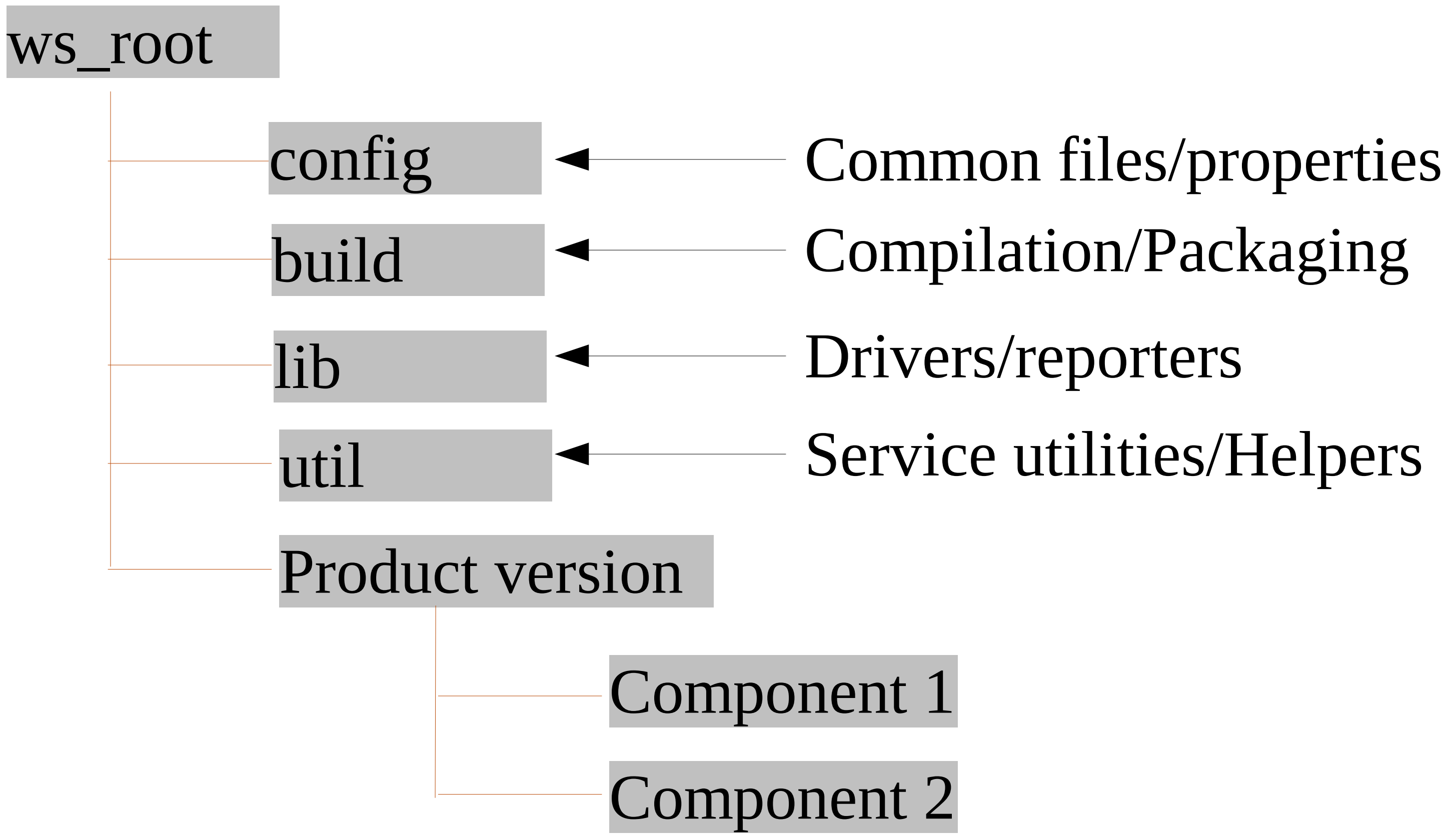
Defining Automation...

- Will the framework be extended in future?
- Is a modular design needed?
- Will workspace require checkpoints/tags.
  - Which source control should be used?
- Is a tree organization appropriate?
  - Will a flat-file driven automation solve any issues?
- Is easy Database switching support desirable?
- How many components will be supported?

Sample Implementation

- Our Implementation consists of:
  - ANT Based framework.
  - Common targets, functions, properties, components controlled from top.
  - Components are free to improve locally.
  - Tree hierarchy allows easy enabling/disabling test branches.
- Rules for Development & Release Engineering
- Certification on Multiple Configurations
- Base-Line Establishment

Our Implementation...



Workspace Implementation Example

```
<target name="deploy-common" depends="init-common">
 <property name="as.props"
  value="--user ${admin.user}
         --password ${admin.password}
         --host ${admin.host} --port ${admin.port}"/>

  <exec executable="${ASADMIN}" failonerror="false">
    <arg line="deploy"/>
    <arg line="${as.props}"/>
    <arg line="--name ${appname}App"/>
    <arg line="--retrieve ${assemble.dir}"/>
    <arg line="${assemble.dir}/${appname}App.ear"/>
  </exec>
</target>
```

This a snippet of a simple target to deploy an EAR file to the application server. In our e.g., this is common functionality used by most components. This is kept in a 'common.xml' file under the 'config' directory that contains all the common and shared code.

Rules for Development Team

- Provided acceptance tests (pulse/quick-look) are run on each module before any check-in
  - Basic functionality in the beginning & enhance later
  - Execution time < 15-20 mins.
- Provided regression tests (Smoke) are run before feature integration.
  - Add unit tests here.
  - Execution time ~ 30-45 minutes.

Rules for Release Engineering

- Release Eng. executes “Pulse” & “Smoke” tests before promotion ensuring a reasonable quality build.

- Basic Acceptance Tests help:
  - To test patches
  - To test special builds
  - To cut down execution time

Creating a Baseline

- For every major feature integration or build promotion:
  - Checkpoint / Tag the test workspace
  - Certify build on most supported configurations.
  - Use the baseline establishing as the minimum requirement for handoff to other teams.

Certification on Multiple Configurations

- Nightly certification on the following:
  - Multiple databases
  - Multiple platforms
  - Multiple Supported Configurations.
- Manual does not scale. Automation is the key!

Multiple Frameworks

- Frameworks are seldom 'One size fits all'.
  - Specialized frameworks/tools needed
    - Silk, Findbugs, Code Coverage tools..
  - Too many to manage?
    - Need to glue them.
      - ANT, Maven, Scripting, testNG, Junit...

(In)Flexible Frameworks

- Identify flexible frameworks as follows:
  - Can the framework accommodate new tools?
  - Can the framework easily accommodate new requirements:
    - Additional Databases, Architectures, OSes, X-Platform
  - Can the reporting be changed?
  - Can the framework be used as-is on another workspace/directory
  - Can the new framework allow processes to be implemented and followed?

Status Check

- Problems with Testing Enterprise Software
  - Typical Problems
- Solution
  - Process
  - Automation
- Sample Implementation
- Important Pointers
- Takeaways
- Q&A

Important Pointers...

- Important Pointers that aid Automation:
  - Follow standard programming guidelines & naming convention.
  - Avoid back-end resource specific details in component interfaces.
    - Security credentials for database connections,JMS Resource provider.
    - If unavoidable, ANT file token replacements targets are used for configurations.

Important Pointers...

- Ensure database portability of persistent applications.
  - Different DDL for each database.
  - Minimize dependencies on stored procedures.
  - Use easily portable datatypes for non cmp focussed applications.
  - XA and non-XA transaction
- Consequences of data-exchange between different system with different encoding.
  - LDAP (UTF-8),Oracle (UTF-16),JVM default client encoding

Important Pointers...

- Make no assumptions about client side and server side default encoding
  - Browser and WebServer
  - Java client application and database
  - Use standard settings e.g. Servlet 2.4 specification <locale-encoding-mapping-list/>
- Sources and binaries to stay in Sync:
  - Sources required to eliminate test case bugs, version inconsistency issues.
  - Promotes ease of enhancement by all
  - Reasonable payoff against execution time

Important Pointers ...

- Maintain integrity of individual components
  - Dev team should ensure integrity of their component while checking-in
- Maintain integrity of all components
  - Dev team should ensure integrity of all integrated components, during feature integrations
- Certify builds nightly
  - The release engineering should certify builds nightly for reasonable quality level

Important Pointers...

- Automate & Execute Basic Acceptance Tests (BAT) every night on multiple configurations.
- Baselining
  - Establish baseline on most supported configuration & Tag Workspace
  - Create auto-check targets.
- Follow proper check-in rules – review changes
- Rules once made, are difficult to change. Think through!

Takeaways

- First, get all requirements. Then define processes before automation.
- Flexible frameworks stand tests of time, products, teams, features and requirements.
  - Create flexible, extensible, simple frameworks
  - Special Tools do special work. Ensure their easy inclusion in your framework.
- Designing and making frameworks is expensive. Think through!

Q&A