

*10,000 Classes,
5 Databases,
3 Operating Systems:
Techniques for
Multiplatform Enterprise
Java™ Technology Testing*

- Sailaja Rao (sailaja.rao@sun.com)
 - Aditya Dada (aditya.dada@sun.com)
-
-

Speaker's Qualifications

- Sailaja Rao
 - Member Technical Staff, Sun Java Systems Application Server Team, Software Quality Engineer,
 - EJB – CMP Expert
 - Code Coverage Expert
- Aditya Dada
 - Member Technical Staff, Sun Java Systems Application Server Team, Software Quality Engineer
 - Deployment Expert
 - Automation Expert

Overall Presentation Goal

Present process for tackling Quality issues with Enterprise applications and implementation framework that eases automation and maintenance of Enterprise Applications tests.

Presentation Agenda

- Problems with Testing Enterprise Software
- Solution
- Implementation of the Solution
 - Prerequisites
 - Process
 - Implementation of process
- Important Pointers
- Conclusion
- Q&A

Problems with Testing Enterprise Software

- General Problems:
 - Bugs need to be caught early in the product development cycle especially when there are over 10,000 classes in the product code
 - Cross-platform (usually 3-4) and multi-configuration testing (~5 databases on different platforms) needs to be done.
- Specifically:
 - Cross-platform and multi-configuration testing is very complex in itself.
 - Enterprise Applications require integrated components to work together. Waiting can cause delay in testing, raising costs.

Our Problem...

- We ensure the quality of Sun Java System Application Server for:
 - Support on 5 databases
 - Support on multiple versions of Linux, Solaris & Windows
 - Support for Intel, SPARC & AMD architectures
 - Support for multiple browsers, loadbalancers, webtier modules.
- In all, they add up to over 1500 configurations!

Our Problem...

- Application Server consists of ...
 - EJB Container
 - Web Tier
 - Java Message Service
 - Java Transaction Service
 - Java Connector Architecture
 - Java Server pages....and many more.
- Test applications cover all above!

Typical Problems...

```
public boolean reserveFlight(int flightID, int seats){  
  
    // BMP Bean1 lookup  
    // CMP Bean2 lookup  
  
    // lookup QueueConnection Factory  
    // send map message  
  
    // lookup connector  
    // create connection factory  
    // update tables.  
}
```

The above code is a snippet from one test application that needs to be run on 3 different platforms and 5 different databases.

Solutions

- The solution is 2-fold.
 - There needs to be a software quality engineering process that is easy to follow and easy to enforce
 - There needs to be sufficient automation that can reduce the complexity of following the process in place.

Implementation of Solution

- Implementation Needs certain pre-requisites
 - All teams involved should be using the following across the board:
 - A common test base
 - A common build framework and workspace structure for the testbase
 - A common set of guidelines for filing bugs/defects/change requests.
 - A common set of guidelines for reacting to bugs/defects/change requests

Software Quality Process (Implementation)

- Software Quality Processes:
 - The development team should maintain integrity of their own component while doing check-ins
 - The development team should maintain the integrity of all the components working together, while doing major feature integrations
 - The release engineering should certify nightly builds for a certain quality level agreed upon by different teams

Software Quality Process (Implementation)

- Software Quality Processes (contd.)
 - Automate & Execute Basic Acceptance Tests (BAT) every night on multiple configurations.
 - Establish baseline on most supported configuration & Tag Workspace

Implementation of Quality Process

- Workspace & Build Framework Design
- Rules for the Development Team
- Rules for the Release Engineering
- Certification on Multiple Configurations
- Base-Line Establishment

Workspace and Build Framework Design Considerations

- The workspace should have a platform independent build-framework
- Common configuration contains all the common targets, functions and properties
- The build framework should provide certain services like reporting, common utilities, easy setup etc.
- The workspace will support multiple components and their owners
- Simple structure and framework required.

Workspace and Build Framework Design Considerations

- The workspace & the framework should be easily extensible.
- A modular design preferable for different components.
- The workspace will require checkpoints.
 - Source control should support tagging or checkpointing.

Workspace Implementation

- Workspace Design
 - wsroot
 - wsroot/config <=Common Files
 - wsroot/build <=Compilation/Packaging
 - wsroot/util <=Libraries/Helpers
 - wsroot/<components>
 - wsroot/ejb/cmp
 - wsroot/web/servlet
- Different <wsroot> for different teams, components.

Workspace Sample Implementation

- Our Sample Implementation consists of:
 - ANT Based framework.
 - Common targets, functions, properties, components controlled from top.
 - Components have freedom to improve upon the existing services
 - Tree hierarchy allows easy enabling/disabling tests.

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 is 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

- Provide set of integrity tests that are run before every check-in
 - Use helloworlds in the beginning & enhance later
 - Keep execution time < 15-20 mins. Name it “Pulse” or “Quick Look”...
- Provide a set of regression tests that are a bit more thorough.
 - Add unit tests here.
 - Keep execution time around 30-45 minutes. Call them “Smoke” tests.

Rules for Release Engineering

- RE should execute “Pulse” & “Smoke” tests before promotion ensuring a reasonable quality build.
- Introduce Basic Acceptance Tests as needed.

Certification on Multiple Configurations

- Nightly certification on the following:
 - Multiple databases
 - Multiple platforms
 - Multiple Supported Configurations.
- Can be overkill if done manually. Automation is the key.

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.

Rules for writing test applications

- Follow blueprints enterprise programming guidelines and it's naming convention.
 - Application name, component name, Webservice name, references
 - Close SQL statements before closing database connections
- 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.

Database Portability

- Differences in multiple JDBC drivers vendor implementation
 - `java.sql.Timestamp`, `java.util.Date`, `java.sql.Date` mapping of BLOB and CLOB most problematic.
- 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

Locale Limitations

- Consequences of data-exchange between different system with different encoding.
 - LDAP (UTF-8), Oracle (UTF-16), JVM default client encoding
- 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/>`

Some Important Pointers

- 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
- Enterprise Application Deployment issues:
 - With J2EE 1.4, stand-alone modules are allowed to be deployed (e.g. EJB-JAR, WAR, RAR) along with EAR. The build framework can easily be setup to take care of packaging configurations and issues.

References

- ANT (<http://ant.apache.org>)
- Blueprints for coding Enterprise Applications (<http://java.sun.com/blueprints/code/namingconventions.html>)

Takeaways from the session

- Setting up a process that is agreed to by all is the first step.
- Automation is the key to following through with the process.
- Look for repetitive patterns in your build process. Those are potential candidates for being automated.
- Identify functionality that can be reused across components like compile, deploy, create resources etc. These are candidates for being shared by everyone

Takeaways

- Identify functionality local to a component, but reused across different applications. These are candidates to be shared by component developers.
- Maintain control of the build-framework. The common functionality can be a maintenance nightmare if proper check-in rules are not followed.
- Never rush the initial setup of the framework and workspace. Rules that everyone is used to are difficult to change.

Q&A