Exploring Cross-Platform Testing at Microsoft

Jean Hartmann

PNSQC Test Conference
October 10, 2011
Background

- Mobile platforms are very restricted development platforms compared to PCs and Macs
- Wanted to leverage existing:
  - test code
  - test infrastructure
- Initially focus on API-based scenario testing rather than UI-based testing
Motivation

- Growing number of platforms and devices
- Existing Office test code contains platform-specific references (not platform-agnostic)
- Porting or writing tests from scratch is too costly and slow
- Heterogeneous testing infrastructure across supported (client) platforms
Goals - Holistic Approach...

- Promote shared test code strategy
- Promote unified automated toolset
- Develop best practices and guidelines for teams
Test Strategy : Today

- Focus: Scenario/End-to-End testing
- Tests coded against libraries that interact via user interface and object model
- Need ways to abstract platform-specific details
- Maximize shared test code

```java
///Class WordDesktopApplication
/// is the base class that defines tasks available
/// for the Desktop Word application.

class WordDesktopApplication
{
    /// Gets the Word canvas.
    public WordCanvas Canvas
    {
        get
        {
            // return canvas...
        }
    }
    // Other Desktop Word Application
    // members.
}
```
Test Strategy: Tomorrow

- Leverage design patterns
- Identify common/unique application properties across target platforms
- Express differences as abstract interfaces
- New or existing tests now coded against new interfaces
- Changes made to underlying implementations without changes to test cases

```csharp
/// IWordApplication is the base interface that declares all tasks available for the Word application.

public interface IWordApplication
{
    /// Gets the Word canvas.
    /// </summary>
    WordCanvas Canvas { get; }
    // Other common Word Application members.
}

public interface IWordDesktopApplication : IWordApplication {..}
public interface IWordBrowserApplication : IWordApplication {..}

public class WordDesktopApplication : IWordDesktopApplication
{
    /// Gets the Word canvas.

    public WordCanvas Canvas
    {
        get
        {
            // return canvas...
        }
    }
    // Other Desktop Word Application members.
}
```
Test Infrastructure : Today

- Complete PC-based testing environment
- Test Case Management System (Oasys)
  - 1 Controller
  - 1..N Test Client(s) (OAClient)
- Test Execution Harness (Motif)

Workflow
1. Tester specifies required machine resources, WinOSs and target apps to be installed (configs)
2. Test code and associated tasklibs (test libs) are executed within Motif environment against target apps
Test Infrastructure: Tomorrow

- Each test client is now ‘paired’ to a device host
  - Device host is PC or Mac
- Each device host is tethered via USB to one or more target devices
  - Needed for installs and logging
  - Accessed via standardized APIs
- Test client communicates with target device via Wifi (TCP-based protocol)
Updated Workflow: Infrastructure

- Device hosts and devices are booted, tethered and registered with Oasys (‘machine pool’)
- Tester requests a test run, specifying device type, OS and app to be installed
- Oasys ‘pairs’ a test client with device host/device
- Application under test is installed on device and launched, ready to be tested
Updated Workflow: Test Code

1. Testers have coded their tests in C# against abstract tasklib interfaces
2. Testers specify a scenario file (includes locations of tasklib implementations for specified devices)
3. Test code and associated tasklibs (test libs) are executed within Motif environment
   - Abstract interfaces replaced by calls to product APIs
   - Calls then routed to a proxy and marshaled (by ref) to device/application (‘remoting’)
   - Upon test or product failure, logs are available
Possible Testing Scenarios

- **Single-client scenarios**
  - Validating a native Office application on given device
  - Example: Load-Open scenario

- **Multi-client scenarios**
  - Validating concurrent features of a native Office application involving multiple endpoints
  - Example: Onenote co-authoring scenario
Example: Interoperability Testing

- Divisional need to ensure file compatibility
- Aka: ‘file fuzzing’
- Before: platform-specific tests run by each team
- After: a shared test case across platforms/teams

- Future: functional test sets representing ‘core’ product
Benefits

- Increased test portability, sharing and reuse
- Lower learning curves for testers
- Lower test infrastructure costs
- Quicker integration and testing cycles for new platforms and devices
Conclusions

- Insights into cross-platform testing challenges
- Ways to abstract test code and test libraries
- Ways to extend existing test infrastructure
- Discussed a practical example
Q&A