Speeding Up Cross Platform Testing – A Case Study

Jeffrey.Weston@Microsoft.com
About me

- Jeffrey Weston
- Senior Software Engineer in Test @ Microsoft
- 9 Years in the Office for Mac Group
Overview

1. Making **Office for Mac** with Ported Code
2. How Porting code lead to “random” bugs
3. Using **Air Crash investigation techniques** to understand where bugs come from
4. The **bug patterns** we found
5. Recommendations to improve our **Testing Efficiency**
How we make Office for Mac

Windows Office

C++

C++

New Windows Office features are merged into existing Office for Mac code

Office for Mac

C++

C++

C++

We also add custom features and UI that Apple customers expect
But adding new files or just replacing existing ones can break the build and cause compile errors.

Developers must manually fix up the code to properly integrate it into Office for Mac.
This updating of code happens to lot of files...

it can take **weeks** to **months** to complete...

all the while the build does **not compile**

and is **untestable**!
Finally a build is produced...

The bugs are seemingly in **random** places across the product!
So the only real option is to...

**RUN ALL THE TESTS!**
But that’s expensive...

time consuming...

boring...

with indeterminate results...

What to do?
TV has the answer!

- *Seconds from Disaster* (National Geographic Channel)
- *Nova: Crash of Flight 447* (PBS)

Air France Flight 447, Crashed in Atlantic Ocean
Crash in July 2000

Crashes are unpredictable  
Complex systems behaving in the real world  
Teams of investigators work hard to find the cause(s)  
Using Root Cause Analysis  
Propose recommendations  
Prevent the same crash from ever happening again.  
Saving Lives.
**Bugs** like Plane Crashes are not Random

- Occur based on initial conditions and a series of events.
- Use root cause analysis to uncover conditions and events on various bugs.
- Look for patterns in the bugs.
- Propose recommendations to prevent the bugs or find them quicker.
Chain of Events

- Pilot loses control and plane crashes
  - Leaking fuel ignites
    - Fuel tank damaged from blown piece of tire.
      - On take off, plane runs over debris from DC-30.
- Pictures do not appear in document
  - Code gets executed
    - Macintosh code deleted in a conflict
      - Multiple conflicts in Merged File
Logs

Airplane Black Box

Used to understand how the Pilots reacted during the incident.

Source Control History

Work backwards from the change that fixed the bug to change that introduced the bug.
Contextual Data

The Air France investigators asked a lot of contextual questions.

from

What was the position of the Cockpit Chair at take off?

to

Could the management hierarchy of Air France have contributed to the crash?

Code Context

- Was the source file ported?
- Did it have Mac specific code?
- Does it interop with Mac specific Code?

```java
NewAssert(cach.isEmpty == !True);
if (FInvalid(cache)
    err = errNone;
else
    err = displayLine(theText, CLIPPED);
NewAssert(err == errNone).
```

- Could it be found by regression automation?
- Could the area be targeted for testing in advance?
Learning from the developer how they personally performed the merge, such as the specific tools they used and the commands they executed.
Using the source control history to revert the source files back to the pre-merge state.
Executing that part of the merge myself.
A Conflict

1. Source Branches

2. Edits on Windows side

3. Edits on Mac side

4. Merge edits results in conflict
>>> ORIGINAL ++++++++++++++++++++++++++++++++++++
  Assert(cache.isEmpty == fTrue);
  Iserr = displayLine(theText, CLIPPED);
  Assert(err == errNone);

==== MAC DEV ++++++++++++++++++++++++++++++++++++
  Assert(cache.isEmpty == fTrue);
  err = displayLineMac(theText, CLIPPED);
  Assert(err == errNone);
}

Mac specific API call was deleted.

==== WINDOWS DEV ++++++++++++++++++++++++++++++++++++
  NewAssert(cache.isEmpty == fTrue);

    if (FInvalid(cache)
      err = errNone;
    else
      err = displayLine(theText, CLIPPED);
    NewAssert(err == errNone);

<<<< END ++++++++++++++++++++++++++++++++++++


Chain of Events
1. Ported code is brought over to Mac
2. Code is changed for Mac
3. File gets merged again
4. Conflict occurs
5. Developer chooses Windows Code block
6. Mac code is removed.

Logs (i.e. Source Control)
1. Identify the change that fixed the bug.
2. Find the original check-in that introduced the bug

Contextual Data
1. Mac specific code was not obvious.
2. Often chooses WINDOWS changes.

Reconstruction of merge
• 60 other conflicts in that file.

Continue Collecting Data for each bug to find patterns.
1. Once upon a time...
2. A source file was copied to the Mac then modified.
3. That original source file was also modified by Windows developers on their side.
4. Mac Developer merged the new Windows changes into the Mac source file.
5. This produced over 60 conflicts, all which had to be manually resolved.
6. When resolving, the developer kept the Windows changes most of the time.
7. Due to the tedious process, the developer accidently resolved this one conflict incorrectly.
8. The Mac specific API was removed and a bug was introduced.
9. Later during the testing process the bug was found after executing many, many test cases.
Bug Patterns Found

• Incorrect conflict resolutions.
• Mistakes when integrating Office for Windows changes into Macintosh code.
• Mistakes integrating Office for Windows changes into older legacy code.
• Bugs in Office Windows ported over to Mac.
### Designing Recommendations

<table>
<thead>
<tr>
<th>Stage</th>
<th>How to prevent bugs from getting checked-in?</th>
<th>How to find bugs more efficiently?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Design Phase (Pre-Merge)</strong></td>
<td>Prepare the code to reduce the number of conflicts before the merge starts.</td>
<td>Mark up unmarked Mac code so developers can clearly see the Mac code.</td>
</tr>
<tr>
<td><strong>Implementing (During Merge)</strong></td>
<td>Review each conflict resolution specifically looking for dropped Mac code.</td>
<td>Have Test Code Review the conflicts after check-in, looking for dropped Mac code.</td>
</tr>
<tr>
<td><strong>Testing Phase (Post Merge)</strong></td>
<td>N/A</td>
<td>Use code coverage analysis to determine whether conflicts resolutions were covered.</td>
</tr>
</tbody>
</table>

Come up with Recommendations to break the chain in as many places as possible.

---

**Stage**

**How to prevent bugs from getting checked-in?**

**How to find bugs more efficiently?**

**Design Phase**

(Pre-Merge)

Prepare the code to reduce the number of conflicts before the merge starts.

Mark up unmarked Mac code so developers can clearly see the Mac code.

**Implementing**

(During Merge)

Review each conflict resolution specifically looking for dropped Mac code.

Have Test Code Review the conflicts after check-in, looking for dropped Mac code.

**Testing Phase**

(Post Merge)

N/A

Use code coverage analysis to determine whether conflicts resolutions were covered.
Results for Mac Office

1. Implemented scripts to reduce conflicts by 50%.
2. Reduced bugs by training Developers to be aware of these patterns.
3. Developers changed their check-in procedures to make conflicts easier to code review.
Summary

- Faced a prospect of random bugs and expensive ways of testing for them.

- Using the Root Cause Analysis of Air Crash Investigation techniques
  1. Piece together the Chain of Events
  2. Through logs, contextual data and reconstruction
  3. Repeat on a sampling of bugs
  4. Look for similarities to uncover patterns

- Come up with Recommendations to break the chain in as many places as possible.