How IT Audit Adds Value Interpreting, Reviewing and Relying on Automated Testing in an Agile Setting

Presentation by Vic D’Angelo and Peter Sword
Quick Hits on Concepts of Agile System Development

- Agile encapsulates many software development methodologies
  - Scrum
  - Kanban
  - Extreme Programming (XP)

- Key Attributes
  - Incremental development and delivery
  - Speed to market
  - Embrace change
  - Collaboration
The Basics of Automated Unit Testing

- By definition, automated Unit Testing examines small sections of code without human intervention
  - Unit test includes expected outcome for each scenario
  - Test logic compares actual outcome to expected
  - Unit tests can include negative testing
    - The test data bank can include scenarios that should trigger validations and exception processing
Variations of Automated Unit Testing

- **Test Driven Development (TDD)**
  - Developer writes test(s) before writing code
  - Tests are run and fail because no code is implemented for them to pass
  - Developer writes functional code
  - Test passes
  - Developer refactors code, ensuring tests still pass, and moves on
Variations of Automated Unit Testing

- **Behavioral Driven Development (BDD)**
  - Goal is to replicate the behavior of a user within the software
  - Tests are written like a sentence
  - Focuses on user behavior and specifications to drive software development
Variations of Automated Unit Testing

- Creation of Unit Tests post-development
  - Agile squads may create automated Unit Testing after they code the functionality
  - Creates technical debt
  - Makes it harder to create quality Unit Tests
  - Relies on manual regression testing until automated suite is adequate
Goals of Cyclical Automated Unit Testing

- In an Agile deployment pipeline, automated Unit Tests act as a form of regression testing
  - If one test fails, the entire move of code to production fails
  - As new code and functionality is added, Unit Tests are added to the regression suite
  - Functional tests include manual UAT and end-to-end prior to production
Management of Automated Unit Testing

- Unit Tests live within the source code, typically in separate test folders
- Configuration of the deployment pipeline and source code automatically detects additions and modifications to the test suite
- When a developer commits code to a branch, all automated tests execute during the build, including new and modified ones
- All automated Unit Tests run with each subsequent build
Agile Project Management Software Tools

There are many Agile Project Software Management tools in the market. A quick Google search yielded over 90 tools, including:

- Monday.com
- Workamajig
- Smartsheet
- Kintone
- Ravetree
- Workfront
- Monday.com
- Wrike

For the purpose of our example, we will be referencing work done using the Atlassian product suite, including Jira.
Example of Automated Unit Testing

• 2018 Audit – Compensation System 2.0
  – System Development Review (also known as System Development Life Cycle) focused on addition of agent profit share (APS) calculations and subsequent payouts via a new system, Compensation 2.0
  – Sales Agents get bonus based on book of business profitability
  – APS calculations require inbound data containing premium and loss in relation to the agent’s book of business
  – The APS agreement contains rules and logic which required coding into Compensation 2.0
    • The output was a determination of agent qualification and the bonus calculation
Automated Testing Review – Jira Example

Finance - Sales Comp2.0 / SC2-979

Exclude Ineligible Premium From OLTP

Details
- Type: Story
- Priority: Blocker
- Affects Version/s: None
- Component/s: None
- Labels: [APS] [SOC]
- Sprint: SC2 SOC Sprint 25
- Epic Link: APS R1 - New Agent Calculation in Prod Parallel
- Story Points: 5
- Acceptance Criteria: Items in email attachment (_FW: Reconciling Proposed_) are addressed

Description
- IM will add three new fields: Referral Source Code, State & Booked to Corporate Code.
- Comp 2.0 will need to store these fields SC2-4404.
- Comp 2.0 logic will exclude:
  - California Earthquake Authority premium based on Booked to Corporate code.
  - Written Premium involuntary business except in NC - VOLUMNTRY_INVOL_CONDITION 'Voluntary' OR 'VOLUMNTRY_INVOL_CONDITION' 'Involuntary' AND 'VOLUMNTRY_INVOL_CONDITION' 'Involuntary' AND 'VOLUMNTRY_INVOL_CONDITION' 'Involuntary' AND 'VOLUMNTRY_INVOL_CONDITION' 'Involuntary' AND 'VOLUMNTRY_INVOL_CONDITION' 'Involuntary'.
  - Earned Premium all involuntary is excluded.
  - Interstate business with a referral source code of Q34.
  - Exclude all umbrella business.

Attachments

Global Internal Audit
Story Description – Exclude Ineligible Premium

This story added three fields to a data feed and changed the eligible premium calculation logic to exclude certain types of policies

Description

IM will add three new fields: Referral Source Code, State & Booked to Corporate Code
Comp 2.0 will need to store these fields SC2-4404
Comp 2.0 logic will exclude:

- **California Earthquake Authority** premium based on Booked to Corporate code
- Written Premium **Involuntary** business except in NC - VLNTRY_INVOL_CD EQ 'Voluntary' OR (VLNTRY_INVOL_CD EQ 'Involuntary' AND CVRG_ST_ABBR_CD NE 'NC')
- Earned Premium all Involuntary is excluded
- Interstate business with a referral source code of 034
- Exclude all umbrella business
Jira Links to Commits

Using Jira “Smart Commits” feature, a Jira User Story has a “commit” link under the “Development” section.
Link to Commits

Development

7 commits

3 pull requests [Merged]

Latest 21/Mar/18 9:52 AM

Updated 31/May/18 5:39 PM

Create branch
Commits Screen
This screenshot shows the commits linked to the User Story

<table>
<thead>
<tr>
<th>Author</th>
<th>Commit</th>
<th>Message</th>
<th>Date</th>
<th>Files</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7acbe1b35a1</td>
<td>Merge pull request #17 in USCM-FINANCE/fin-salescomp-aps-premium-ingestion from…</td>
<td>21/Mar/18</td>
<td>7 files</td>
</tr>
<tr>
<td></td>
<td>6af0e68b9f</td>
<td>SC2-979 Finalizing unit tests</td>
<td>20/Mar/18</td>
<td>2 files</td>
</tr>
<tr>
<td></td>
<td>43996e00ed</td>
<td>SC2-979 Adding [green] tests back in; were lost in merge conflict</td>
<td>20/Mar/18</td>
<td>2 files</td>
</tr>
<tr>
<td></td>
<td>49a633ad410</td>
<td>SC2-979 Changing method parameter</td>
<td>20/Mar/18</td>
<td>1 file</td>
</tr>
<tr>
<td></td>
<td>b0e69c8763</td>
<td>SC2-979 Changing method parameter</td>
<td>20/Mar/18</td>
<td>2 files</td>
</tr>
<tr>
<td></td>
<td>36a9707714</td>
<td>SC2-979 Pushing failing test so team can collaborate</td>
<td>20/Mar/18</td>
<td>1 file</td>
</tr>
<tr>
<td></td>
<td>234b8fbae1</td>
<td>SC2-979 Add logic to exclude records and premiums based on criteria</td>
<td>19/Mar/18</td>
<td>6 files</td>
</tr>
<tr>
<td>Commit SHA</td>
<td>Message</td>
<td>Date</td>
<td>Files</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>-------------------------------------------------------------------------</td>
<td>------------</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>6afe6b9f</td>
<td>SC2-979 Finalizing unit tests</td>
<td>20/Mar/18</td>
<td>2 files</td>
<td></td>
</tr>
<tr>
<td>MODIFIED</td>
<td>PIFIN_APS_Premium_batch/src/test/groovy/com/limig/uscmin/comp2/aps/premium/oltp/processor/APSOLTPLoadProcessorTest...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43996ce00ed</td>
<td>SC2-979 Adding yellow tests back in; were lost in merge conflict</td>
<td>20/Mar/18</td>
<td>2 files</td>
<td></td>
</tr>
<tr>
<td>49a6d3d6410</td>
<td>SC2-979 Changing method parameter</td>
<td>20/Mar/18</td>
<td>1 file</td>
<td></td>
</tr>
<tr>
<td>b0e59c87f83</td>
<td>SC2-979 Changing method parameter</td>
<td>20/Mar/18</td>
<td>2 files</td>
<td></td>
</tr>
<tr>
<td>36ad9707714</td>
<td>SC2-979 Pushing failing test so team can collaborate</td>
<td>20/Mar/18</td>
<td>1 file</td>
<td></td>
</tr>
<tr>
<td>234b8fbae1</td>
<td>SC2-979 Add logic to exclude records and premiums based on criteria</td>
<td>19/Mar/18</td>
<td>6 files</td>
<td></td>
</tr>
<tr>
<td>MODIFIED</td>
<td>PIFIN_APS_Premium_batch/Fusionfile</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MODIFIED</td>
<td>PIFIN_APS_Premium_batch/src/main/java/com/limig/uscmin/comp2/aps/staging/premium/reader/PremiumStagingReader.java</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See more files in fin-salescomp-aps-premium-ingestion
Reviewing Automated Tests

- Sometimes the auditor can just review the final automated tests to see if they test all of the acceptance criteria.
- Many times this is difficult. The auditor may need to review the progress of the automated testing through a series of commits to understand the tests.
**First Commit – Logic Change Example**

- This screen shot of the source code shows the coding to ignore California Earthquake, Interstate, and Umbrella premium.
Code File Structure

The source code shows the file structure for the code, including the Unit Test folders:

- **PIFIN_APS_Premium_batch**
  - **src**
    - **main/java/com/lmig/uscfin/comp2/aps**
      - **premium**
        - **batch**
          - **Utils.java**
      - **model**
        - **StagingPremiumData.java**
      - **oltp/processor**
        - **APSOLTPLoadProcessor.java**
  - **staging/premium/reader**
    - **PremiumStagingReader.java**
  - **test/groovy/com/lmig/uscfin/comp2/aps/staging**
    - **PremiumStagingReaderTest.groovy**
  - **Fusionfile**
First Commit – Ignore CEA Business

This screenshot zoom of the source code shows the code to ignore premium from California Earthquake

```java
394 + /**
395 + * Ignore all policies sold through California Earthquake Authority
396 + * @param stagingRecord
397 + * The staging record
398 + * @param errorMessage
399 + * The error message buffer
400 + * @return
401 + * True if the record should be marked as ignored false otherwise
402 + */
403 + private void ignoreCEAPolicies(StagingPremiumData stagingRecord, PremiumData oltpPremiumData, StringBuffer errorMessage)
404 + {
405 +     if(stagingRecord.soldThroughCEA())
406 +     {
407 +         errorMessage.append("(Earthquake corporate codes are ignored)");
408 +     }
```
First Commit – Ignore Interstate Business

This screen shot zoom of the source code shows coding to ignore interstate premium

```java
/**
 * Ignore interstate business
 * @param stagingRecord
 * The staging record
 * @param errorMessage
 * The error message buffer
 */

private void ignoreInterstateBusiness(StagingPremiumData stagingRecord, PremiumData cltpPremiumData, StringBuffer errorMessage)
{
    if (stagingRecord.is Interstate())
    {
        errorMessage.append("(Interstate business is ignored");
    }
```
The only change in the Unit Testing was for the new fields added in the data feed.
The new fields added in the data feed were added to the Unit Testing
Unit Test Update to Ignore Umbrella Policy Premium

In the next to last commit, testing was added to validate that Umbrella policies are ignored.

```java
void "Records with an umbrella line of business are ignored"() {
    given: "a staging record with a LOB defining umbrella"
    stagingRecord.lineOfBusiness = "Umbrella"
    def stagingPremiumData = validStagingData
    stagingPremiumData.lob = "Umbrella"

    when: "the staging record is being mapped to the oltp record"
    boolean success = apsLossLoadProcessor.mapToOltpClaim(stagingRecord, oltpClaimRecord)
    boolean success = processor.mapToOltpPremium(stagingPremiumData, premiumData)

    then: "the record is marked as ignored"
    stagingRecord.recStatus == APSClaimsDAO.STG_REC_STATUS_IGNORE
    stagingPremiumData.recStatus == "IGNORE"
    success == false
```
Unit Test: Involuntary / Not NC Written Premium

```java
511   669     void "Records with an involuntary indicator and not in NC have written premium set to 0"() {
512       {
513       given: "a staging record with a booked to corporate code of CZA"
514       def stagingPremiumData = validStagingData
515       stagingPremiumData.vINonY1Ind = "1"
516       stagingPremiumData.state = "MA"
517
518       + and: "company code is blank"
519       stagingPremiumData.logyBrandCdVal = ""
520       + and: "QBE policy number"
521       stagingPremiumData.policyNum = "1234567890"
522       + and: "party search criteria for party with default company code (Safeco)"
523       def partySearchCriteria = new PartySearchCriteria()
524       partySearchCriteria.partyNum = stagingPremiumData.agentCdVal
525       partySearchCriteria.partyCompany = "Safeco"
526
527       + and: "party search returns party id for default company (Safeco) and party number"
528       def partyId=100L
529       processor.refDao = Stub(ReferenceDataImpl)
530       processor.refDao.getReferenceData(partySearchCriteria) >> [partyId]
531
532       when: "the staging record is being mapped to the oltp record"
533       boolean success = processor.mapToOltpPremium(stagingPremiumData, premiumData)
534
535 -       and: "appropriate values injected"
536 -       def refDao = Mock(ReferenceDataImpl)
537 -       refDao.getReferenceData(PartySearchCriteria) >> Collections.singletonList(1234L)
538 -       processor.refDao = refDao
539
540 -       then: "written premium is set to 0"
541       premiumData.writtenPremium == 0.0
542       System.out.println(stagingPremiumData.errorDescription)
543       success == true
```

Global Internal Audit
void "Records with an involuntary indicator and not in NC have written premium set to 0"()
{
    given: "a staging record with a booked to corporate code of CEA"
    def stagingPremiumData = validStagingData
    stagingPremiumData.vINonVInd = "I"
    stagingPremiumData.state = "MA"
    and: "company code is blank"
    stagingPremiumData.lgcyBrandCdVal = ""
    and: "QBE policy number"
    stagingPremiumData.policyNum = "123S567890"
    and: "party search criteria for party with default company code (Safeco)"
    def partySearchCriteria = new PartySearchCriteria()
    partySearchCriteria.partyNum = stagingPremiumData.agentCdVal
    partySearchCriteria.partyCompany = "Safeco"
    and: "party search returns party id for default company (Safeco) and party number"
    def partyId=100L
    processor.refDao = Stub(ReferenceDataImpl)
    processor.refDao.getReferenceData(partySearchCriteria) => [partyId]
Involuntary / Not NC Written Premium: Actions and Expected Outcome

```java
when: "the staging record is being mapped to the oltp record"
boolean success = processor.mapToOltpPremium(stagingPremiumData, premiumData)

and: "appropriate values injected"
def refDao = Mock(ReferenceDataImpl)
refDao.getReferenceData(PartySearchCriteria) >> Collections.singletonList(1234L)
processor.refDao = refDao

then: "written premium is set to 0"
premiumData.wrtnPremium == 0.0
System.out.println(stagingPremiumData.errorDescription)
success == true
```
Auditing Unit Testing – Detecting Missing Tests

- For processing logic with multiple conditions and multiple outcomes, the Unit Testing should test each condition and each outcome.

- The condition “Involuntary and Not in North Carolina, Written Premium = 0”, implies another condition that Written Premium for Involuntary Policies in North Carolina should be included.
  - If the Unit Test didn’t validate this condition the auditor may have an issue and recommend that they add it.
Auditing Unit Testing – Refactoring

Sometimes the development team will update the Unit Testing to refactor (improve) or add conditions that weren’t tested before.

```python
365 +     def "party search is called with the default company when staging company blank and policy is QBE"(){
366 +         given: "a valid StagingPremiumData object"
367 +         def stagingPremiumData = validStagingData
368 +         and: "company code is blank"
369 +         stagingPremiumData.lgcyBrandCdVal = ""
370 +     }
371 +     end: "QBE policy number"
372 +     stagingPremiumData.policyNum = "1234567890"
373 + +
374 +     and: "party search criteria for party with default company code (Safeco)"
375 +     def partySearchCriteria = new PartySearchCriteria()
376 +     partySearchCriteria.partyNum = stagingPremiumData.agentCdVal
377 +     partySearchCriteria.partyCompany = "Safeco"
```
Detecting Gaps in Automated Testing

The code in this story does not have any Unit Tests created
Detecting Gaps in Automated Testing

Some Unit Tests only validate “happy path” processing, and do not test validations and exception processing.
# Detecting Gaps in Automated Testing

```
15 - String line = "TMTD_RPTG 1020191152798 s1 PLSW480264 10212011Workers C"
15 + String line = "TMTD_RPTG 1020191152798 s1 PLSW480264 10212011Workers C"
16 16 when: "the mapRecord method is called"
17 18 stagingPremiumData stagingPremiumData = premiumStagingReader.mapRecord(line)
19 19 - then: "a populated stagingClaimRecord object is returned"
20 -
21 + then: "a populated stagingPremiumData object is returned"
22 22 stagingPremiumData.recordType == "D"
23 23 stagingPremiumData.srcSym == "TMTD_RPTG"
24 24 stagingPremiumData.bookMon == "10"
25 25 stagingPremiumData.bookYr == "2018"
26 26 stagingPremiumData.agentCdVal == "11302798"
27 27 stagingPremiumData.brandCdVal == "S1"
28 28 stagingPremiumData légèreBrandCdVal == ""
29 29 stagingPremiumData.plySrcSysIdVal == "PLD"
30 30 stagingPremiumData.policyNum == "W480264"
31 31 stagingPremiumData.plyEffDt == "10212016"
32 32 stagingPremiumData.lob == "Workers Compensation"
33 33 stagingPremiumData.lobLvl2 == "All Other Lines"
34 34 stagingPremiumData.lobLvl3 == "Property"
35 35 stagingPremiumData.lobCtrl == "RM Workers Compensation"
36 36 stagingPremiumData.lobCode == "REPMON"
37 37 stagingPremiumData.newRvlCode == "2"
38 38 stagingPremiumData.newRvlDesc == "RENEWAL"
39 39 stagingPremiumData.v1MonVsInd == "Y"
40 40 stagingPremiumData.earnedPremium == "00000800000.000000"
41 41 stagingPremiumData.vrtnPremium == "00000001000000.00"
42 + stagingPremiumData.refSourceCode == "343"
43 + stagingPremiumData.state == "No"
44 + stagingPremiumData.bookedToCorpCode == "Go America Indy Region"
```
Detecting Gaps – Reviewing for Unit Test Notes

The Unit Test code has a comment “TODO fix test cases that have been commented out”
Detecting Gaps – Reviewing for Unit Test Notes

The Unit Test code has a comment “TODO fix test cases that have been commented out”
Internal Audit’s Role in Adding Value

- When conducting System Development Reviews (SDR), reviewing automated Unit Testing adds assurance that each unit of code functions as expected and adds value by:
  - Giving the Product Owner and Stakeholders assurance of code coverage
  - Helping ensure quality of the product delivery
  - Assisting in evaluation of adequacy of regression testing
Potential Internal Audit Recommendations

- We may recommend testing controls such as edits and validations.
- If the recommendation is addressed in a later sprint, the Auditor can review the code and the Unit Test for assurance it was actually implemented.

<table>
<thead>
<tr>
<th>Findings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete automated unit testing was not always created for the APS functionality in the MVP release.</td>
</tr>
<tr>
<td>As a result, APS functionality in the MVP release may not have been adequately regression-tested.</td>
</tr>
</tbody>
</table>
Reviewing Automated Testing is a Valuable Struggle

- Sometimes the automated Unit Testing is the only documented testing for a User Story
- Review of the content in the Unit Tests to the acceptance criteria is necessary to provide assurance the User Story’s Acceptance Criteria is met and will continue to function as intended over time.