February 22-25, 2010

Source Control...$100
Regression Script...$500
Good Automated Release Steps...$Priceless

by

Jeffrey Wren

Principle Consulting Engineer

Paradigm Works

Introduction

- Release Management Defined
  - Release management is the set of steps taken to guarantee that one's source code, schematic, layout, etc, is ready for distribution to the customer.
  - The customer is defined as another team member, another division within the company, or a customer in the traditional sense.
- Critical to the success of every project
- Often times, it is overlooked and underestimated
- Poor release management can cost a project man months in lost productivity
- Often, this lost productivity is hidden, i.e. 15 minutes here, 30 minutes there, etc.

Overview

- The 3 problems of release management
- The 5 steps to a good release flow
  1. Create/Update Workspace
  2. Modify and Commit Changes
  3. Submit Changes
  4. Integrate Submits (Automate)
  5. Publish (Automate)
- Case Study
- Conclusions

Release Management Problems

- The User Problem
- The Release Manager Problem
- The Reproducibility Problem

The Typical User Workflow

- A user is defined as anyone who modifies the source files of the repository that make up the project. This can be a designer, a verification engineer, schematic entry tech, etc.

- Typical List of Workflow Steps For User
  1. Creates workspace from Source Control.
  2. Makes modifications
  3. Runs local tests or regression list to verify changes
  4. Update to latest changes on trunk
  5. Re-run tests to make sure everything still works
  6. Commits changes to source control.

The User Problem

- How does one even know that the workspace being created works, i.e. compiles and runs basic set of tests?
- Burden of acceptance testing is placed on user
- Steps are often manual and prone to user mistakes
  - User forgets to commit a file
  - User has environment variable set in local shell that is needed for things to work
- Race conditions between users
  - Example: While user is verifying changes before commit someone else makes conflicting commit

Release Manager Problem

- The release manager's focus is on the entire project
- The release manager must ensure that all changes from the team integrate together to make a deliverable to the customer
- Typical Workflow Steps for Release Manager
  1. Determine latest good code.
  2. Compile a workspace
  3. Run regression suit of tests
  4. Interpret results and resolve issues
  5. Label the files
  6. Notify project team
  7. Repeat
- Steps prone to human error if performed by hand

Release Manager Problem Cont.

- What set of changes are causing compile/test errors
- Who is responsible for the fix?
  - Are they available (Vacation, out sick)?
  - Different time zone communication and resolution delays
- Once problem is found, should it be fixed?
  - Is fix required before moving on?
  - Should changes be backed out?
  - Does fix causes other areas of code to break?
- Resolution can be a moving target

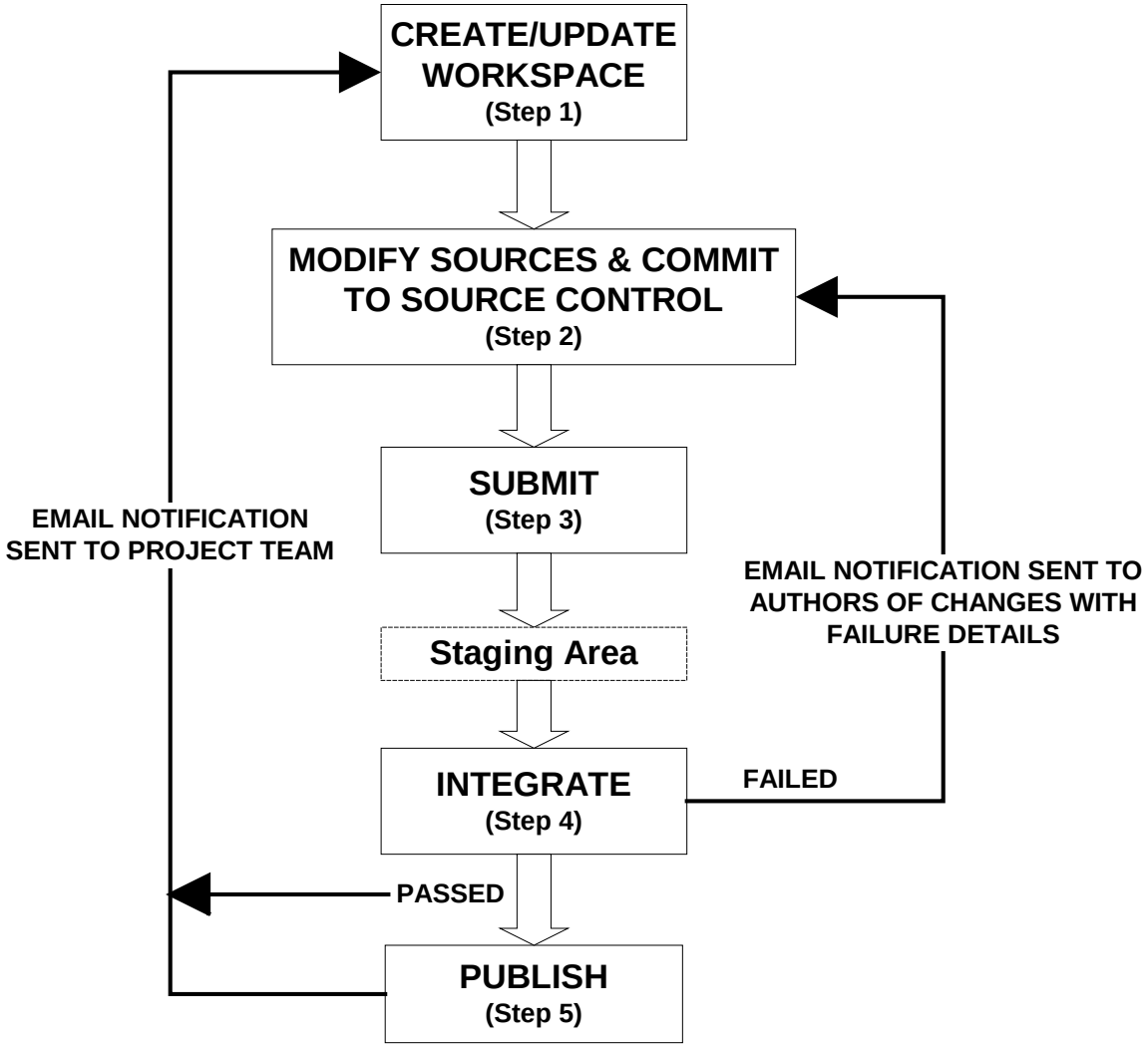
Reproducibility Problem

- Random verification environments based on seed
- Changes to the testbench can effect the random behavior
- Burden is on user to generate label that represents problem
  - Can be time consuming
  - Everything must be committed before labeling

What Makes Up a Good Release Flow

- Takes the burden of acceptance testing out of the hand of the user
- Makes the release manager's job easier so that they can work on other project related tasks
- Makes the development area reproducible at any incremental stage of the design with minimal effort

Five Steps For a Good Release Flow



Step 1: Create/Update Workspace

- Command line tool or script that creates a workspace to a known working label

- Working is defined to be a workspace that at a minimum when created will compile and execute a set of tests that are defined in the Integrate phase (Step 4) of the flow.

Create Workspace Example

```
% release_tool create_sandbox sandbox_name
```

Where:

- release_tool — The command line executable
- create_sandbox — The subcommand call to the release_tool that performs the create
- sandbox_name — A user argument that defines what the name of the sandbox should be

- Having the tool automatically keep track of the release labels removes the burden of the user keeping track and trying to figure out what is the latest good code.

Create Workspace Example Cont.

- For reproducibility purposes, the command should have an option that allows it to accept any previous release label.

```
% release_tool create_sandbox \
  -label PROJ-REL_1_2_3 sandbox_name
```

Update Workspace Example

- Ability to update an existing workspace to the latest Published (Step 5) release

```
% release_tool update_sandbox
```

- Option to update to the latest Integrated (Step 4) release

```
% release_tool update_sandbox –latest_accepted
```

Step 2: Modify Source Code

- User makes modifications to source code
- Commits changes to source control tool
- Two methodologies are available
  - Mainline/Trunk Approach
  - User Branch Approach

Mainline/Trunk Approach

- **Pros:**
  - Files can be locked so that two users cannot modify the file at the same time. This can be specifically useful for binary files.
  - Commits cannot be performed until changes made by another user are rectified. This can help to prevent merge conflicts.

- **Cons:**
  - Two users cannot work on the same file at the same time. More of an issue with distributed work teams.
  - Commits are performed on same branch so that if two users need to touch the same file for their change group to pass the Integrate (Step 4) but user one's changes don't work, user two changes can be blocked.

User Branch Approach

- **Pros:**
  - Two users can work on the same file at the same time.
  - Commits to the user branch do not affect anyone else. Therefore, commits can be done frequently.

- **Cons:**
  - Files cannot be locked. This can be an issue for binary files.
  - More merge conflicts when integration to the trunk is performed.

Step 3: Submit Changes

- A Submit is made up of 1 or more files
- Should contain all the files that are dependent on one another for the bug fix or functional change
- For Example:
  - Three files were modified to fix a bug. The changes to each are required for the fix to work
  - Do not submit each file individually. If there is a problem in the Integrate phase (Step 4), the individual submits will most likely fail, even if the changes are correct.

Submit Changes Cont…

- Depending on source control tool the Submit information can be:
  - Listing of each file along with appropriate source control version
  - Label that marks the file versions that have changed
- Information is stored in a staging area (A directory, or file)
- Staging area is location where Submits are stored until the Integrate (Step 4) is ready to process them

Step 4: Integrate the Submit Groups

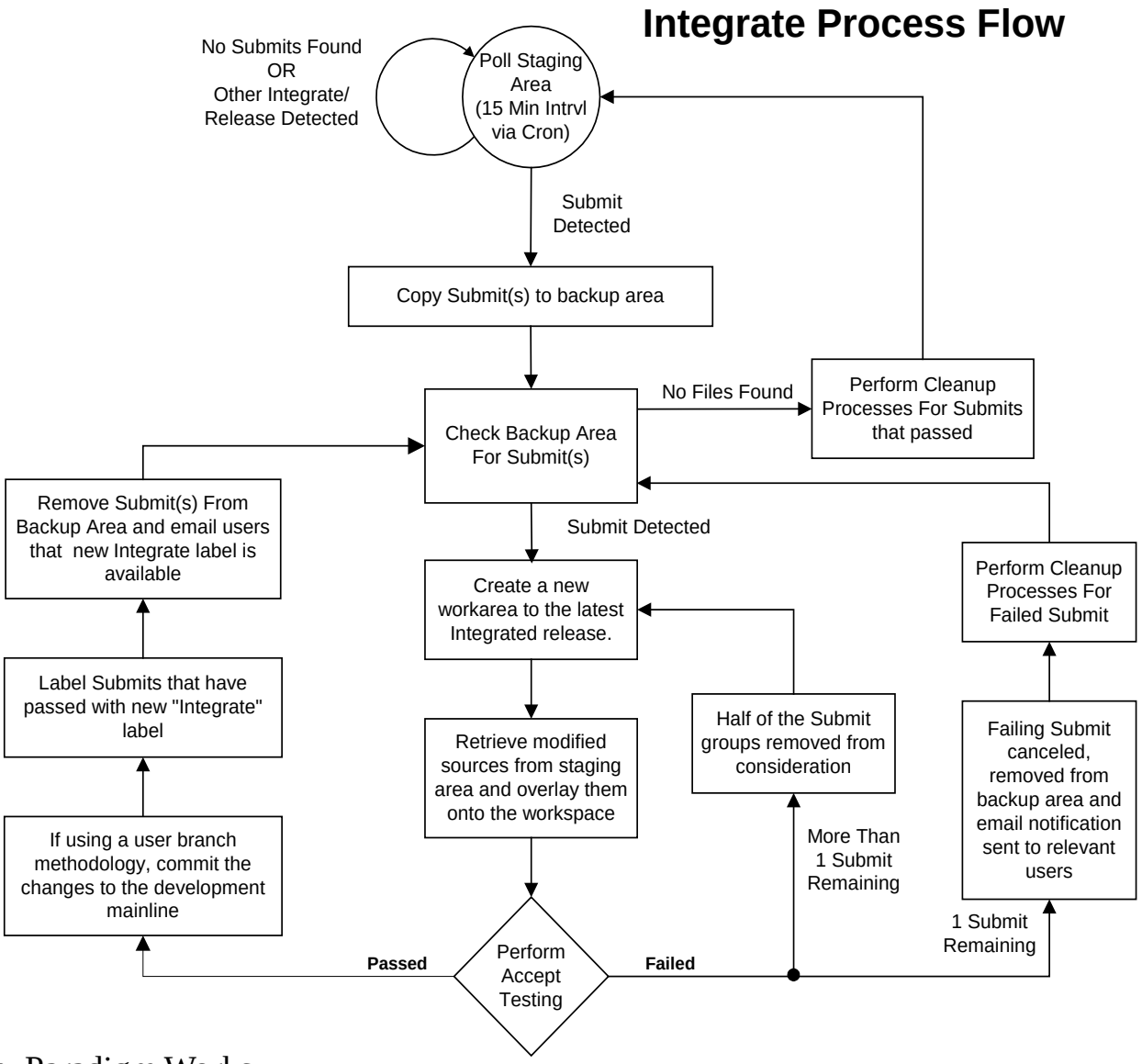
- Heart of Release Flow
- Keeps code in a known working state
- Made up of acceptance testing
  - Compile of each relevant testbench
  - Simple test from each testbench
- Everything must pass in order to be integrated into the release
- Can be automated with binary search algorithm
- If testing passes then email can be sent announcing new changes

Step 4: Integrate the Submit Groups Cont…

- If testing fails changes are rejected and email can be sent to relevant parties detailing how to reproduce

- Because the Integrate knows the state of the project at time of failure, reproducibility is made up of the following steps:
  - Create workspace to the Integrate label at time of failure
  - Overlay changes detailed in Submit group

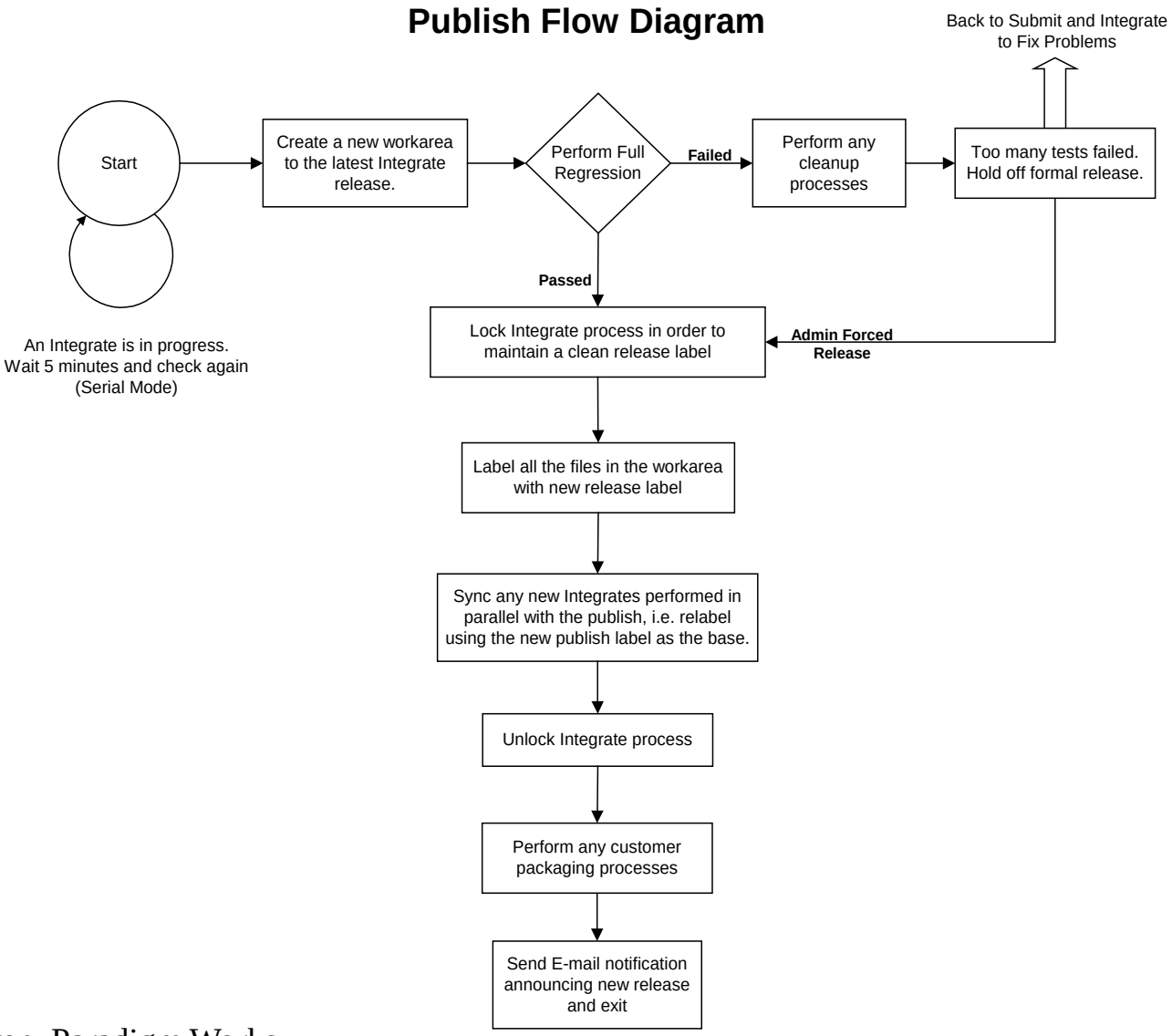
Integrate—Step by Step (Automate)



Step 5: Publish

- The full suite of testing is performed
  - Nightly Regression
  - Weekend Regression
  - Full Regression
- This is a blessing of the label applied during the Integrate
- If it testing passes then new label is applied representing the higher level of testing
- If it fails then no new label is applied and Commits, Submits and Integrates can continue until next Publish
- For larger projects, it is recommended that the ability to perform Integrates and Publishes concurently be established

Publish—Step by Step (Automate)



Case Study

- Company X utilized in-house solution called the Release Process Tool (RPT)
- Source control tool was ClearCase by IBM Rational
- It solved the problem of reproducibility
- Still had problems regarding user and release manager

Company X User Problem

- RTP tool acted as wrapper for commits and applied a label at time of commit. This label loosely represented a change group.
- Users encouraged to perform acceptance testing to make sure changes passed, but not enforced.
- **PROS:**
  - Broken code blocked from immediate release
- **CONS:**
  - Passed problem to release manager
    - Forgotten file commits and local env variables in user's shell made debug difficult
    - User could commit one file or many. Determining dependencies could be difficult if multiple commits represented a single change group.
  - Feedback on changes dependent on release manager

Company X Release Manager Problem

- RPT provided functions to manage which user commits were tested and promoted
- Required release manager to have detailed knowledge of all aspects of design, and verification
- The larger the team, the more difficult the task of debugging issues
- Many steps were manual
- Project releases were dependent on availability of release manager
  - Out sick
  - vacation

Company X Reproducibility Problem

- Applying labels at time of commit worked fairly well
- Allowed users to use a label or combination of labels to reproduce problem
- Worked best if commit included all the files representing a single change group

Company X's RPT Tool vs. 5 Step Flow

- Acceptance Testing
  - **RPT**
    - Burden placed on user, but not enforced.
    - Result was the release manager would need to perform debug if something broke
  - **5 Step Flow**
    - Burden taken away from the user.
    - Changes only integrated into release if acceptance testing passed.
    - The release manager no longer needed to debug inter-dependencies between multiple user changes
  - **Potential Savings**
    - 1 man hr per commit

Company X's RPT Tool vs. 5 Step Flow

- Feedback to user if changes successfully integrated into release
  - **RPT**
    - Dependent on release manager: Typical 1 day, Worst case 1 week
  - **5 Step Flow**
    - Dependent on time to perform acceptance testing and binary search for failing Submit.
    - For Company X this was a minimum of 2 hrs, worst case 1 day.
  - **Potential Savings**
    - 2 man hrs
    - Based on fact user gets quick feedback, saving time on issues not propogating to other areas of the project.

Company X's RPT Tool vs. 5 Step Flow

- Release manager time to perform release
  - **RPT**
    - Typical: 0.5 day
    - Worst Case: 3 days
  - **5 Step Flow**
    - Automated
    - Release manager is free to perform other tasks
  - **Potential Savings**
    - 4 man hrs per release

Company X's RPT Tool vs. 5 Step Flow

- Nightly Regressions
  - **RPT**
    - Often would crash because a user didn't properly submit the needed changes or due to incompatibility between user changes.
    - A loss of feedback as to the state of the project of 1 day
  - **5 Step Flow**
    - Automated acceptance testing executed throughout the day guaranteed that the nightly regressions would execute cleanly.
    - This gave the development team consistent feedback as to the state of the project
  - **Potential Savings**
    - 1 day of project productivity per week.
    - Based on estimate that nightly regressions would fail at least once a week.

Company X's RPT Tool vs. 5 Step Flow

- Time between Publishes
  - **RPT**
    - Dependent on release manager. Best case, 1 a day.
  - **5 Step Flow (Automated)**
    - Consistent incremental Submit releases based on acceptance testing
    - Daily release based on nightly regression
    - Weekly release based on full weekend regression.
  - **Potential Savings**
    - Invaluable.
    - Helps keep project on schedule.

Conclusion

- The 3 major problems for release management to solve were presented
  - The user problem
  - The release manager problem
  - The reproducibility problem
- A 5 robust step solution was detailed that solves the problems and which can be automated
- Case study was provided showing how Company X made significant performance increases by moving to an automated 5 step flow
- The larger the team, the greater the cost reward
- Improvements could very well be $Priceless if time to market is reduced allowing a company to beat their competition to market

Resources

- Free Open Source Software Tool ReleaseWorks®
  - Utilized by Company X to implement 5 step flow
  - Available on SourceForge
    - http://releaseworks.sourceforge.net
  - Currently supports
    - ClearCase
    - CVS
    - SVN (comming soon)