Effective Solution for Defect Management across Product Releases and Builds

by Ramanjaneyulu Malisetti, Principal Software Engineer, CA Technologies; and Naveen Arora, Associate Engineering Services Architect, CA Technologies

Executive Summary

The market turbulence of the recent years has foreshadowed a new phase of software development to reduce the IT cost and improve the delivery efficiency. Diversified development tools and lack of integration continues to roil traditional software development models. How can the development teams respond quickly and nimbly to the changing environment without getting tied up in knots?

In today’s knowledge age, a single unified development toolkit that is able to both debug and profile is the key to results, irrespective of the chosen platform. This paper examines the challenges and rewards of software development teams. Software development teams work with tools from multiple vendors for various activities of the development. Some examples include: source control management (SCM) and bug tracking using Rational Team Concert (RTC), build and continuous integration management using Maven and Teamcity, Quality Control from HP and an artifact repository using IVY. Development teams seldom find a holistic solution in a single tool set to meet all their development needs.

Due to the diverse set of tools, and lack of their out-of-the-box integration, development teams have to follow manual or semi-automated processes to achieve the following tasks:

- Configuring merging changes across streams (branches).
- Test case coverage for issues fixed to ensure they are included in all future product releases.
- Reporting code coverage metrics related to the changes done per defect fixed and maintaining its consistency across all releases.
- Integrated view of test results across builds of multiple product source branches for every change set related to fixing bug(s).

This paper describes a robust system that can be built using products like CA Release Automation for the efficient build and deployment of software. This addresses the defects that are common across various build/release images as a roadblock, and discusses a solution for this problem.
Introduction

Technology dynamics in today’s market is changing rapidly. The fast paced change in the technology and its adoption across various software development organizations is driving software architects, engineers, and business analysts to shift the paradigm of product development methodology from traditional Waterfall to Agile.

One of the agile manifestos of Agile focusses is on customer collaboration over contract negotiation. This shift in the mindset of product development is helping organizations to understand the customer’s requirements in a better way. Although involving the customer at each step makes the product development lifecycle shorter (from months to weeks), it increases the complexity and challenges which development teams face.

- Agile enables continuous delivery, which can multiply the number of releases which need to be actively maintained and supported.
- When customers report defects from a released version of the product, the defects may also apply to previous or subsequence versions of the product. The onus falls on the product teams to diligently identify the impacted releases and back port or forward port the fixes.

No one wants to release product versions (containing either new features or hot fixes) that get customer attention on for features not working in their environment. It is important that development teams use a system that works with the product development lifecycle tools comprised of Source Control Management, Defect Management, Test Case Management and Build Management.

The proposed solution primarily identifies and automates the process of merging change sets (patches) to all the branches of the product source code whenever there is a check-in (patch) to the related branch of a product source. The system should also track the source files and test cases coverage by tagging defect numbers across the source branches corresponding to releases. The tagging with defect numbers enables to queries to identify the branches that do not have the change sets and test cases corresponding to a defect fix.
The following diagram depicts an overview of the proposed solution:

1. **SCM**: a Source Control Management system used for managing product source for various releases.
2. **Build System**: This tool is responsible for checking out source per release from SCM and builds the product periodically or on demand.
3. **Defect Management System**: This tool is used for managing defects reported by customers against various releases of a product.
4. **Quality Center**: This tool is used to manage test cases and their run status for the released versions of a product.
5. **Patch Integrator (PI)**: This is a proposed system component that can seamlessly integrate with the disparate tool sets that are generally available in the development environment to identify and report the missing test cases and defect fixes in branches. The PI notifies users on the missing change sets related to defects across the branches of product source.
6. **Automator**: It is part of the Patch Integrator that runs as a daemon process and iterates over its database to trigger pending actions.

**How to Implement Patch Integration (PI)**

1. PI is implemented and configured to interact with SCM, the Build System, the Test Case Management and the Defect Management System.
2. PI monitors for change-sets (patches) created on the branch of a product source for a particular defect.
3. For every change-set created on the branch:
   a. Get the defect number.
   b. Get the list of files and changes in each file.
   c. For all other branches check whether they have got the change-set tagged with this defect.
      i. If “Yes” then whether check each file in the previous step contains all the changes.
      ii. If “No” i.e., if branch doesn’t contain a change-set, then create a change-set and apply a cross-merge.
      iii. If the cross-merge is not successful then notify the change-set owner to manually resolve changes.
      iv. On a successful cross-merge, update the PI database and trigger a build.
   d. For defect numbers tagged in a change-set, query a Test Case Management tool like Quality Center to get test case coverage. If there are test cases, update PI database accordingly.
   e. For the last successful build in the Build System, query the code coverage report for files contained in change-set and update the PI database.

Implementation Considerations

- Change-sets created in the SCM tool contain additional information such as the associated defect IDs.
- The SCM tool provides an API to read file information for a given change set.
- Test Cases created in the test management tool are tagged with defect information such as the defect ID.
- The Build System provides information about code coverage for a given build number at various levels such as the class level and method level.

Algorithmic Flow of PI

Build change sets are created for a defect on an unstable branch. Next we run the test automation suite and compute the code coverage (a “smoke build”). If all test cases pass, we merge the change sets to a stable branch and run the build on the stable branch (the final build).

Upon completion of the final build, the Patch Integrator (PI) plugin is triggered to identify the need for cross merging this change set to other maintenance branches and discovers missing test cases, code coverage and updates the PI database with details like the product branch version, fixed defects and related test cases. The following sequence of actions takes place in the PI for all source streams of a product.

1. PI checks whether this call is originated from itself, if yes, following steps are skipped. Otherwise,
2. PI checks with the Defect Management System, as to whether the given defect ID is a bug or not. If defect is not a bug, then the following steps are skipped. Otherwise,

3. PI gets other related maintenance branch’s details from configuration.

4. Based on change sets ID, the PI queries SCM to get a list of source and resource files related to change sets and perform following for each configured branch.

5. Check whether the PI database has these defect(s) fixed. If not:
   a. Determine whether this branch has similar source files and similar code snippets (assuming same the package structure is retained across branches). If not, break.
   b. Send notification to the Configuration Manager that auto-merging can take place in the specific branch. Provide options such as “Proceed,” “Defer,” “Ignore.”
   c. Wait for acknowledgement from the Configuration Manager. If the Configuration Manager responds with the “Proceed” option, then auto-merge change sets into the branch. If auto-merge is successful, then repeat above steps a, b and c for the remaining maintenance branches.
   d. If auto-merge is unsuccessful then send notification to the Configuration Manager for manual merging of the change set(s) and promote the change sets to the smoke build.
   e. If the Configuration Manager responds with the “Defer” option for step 5b, then update the PI database with defect details and action status “Defer.”
   f. If the Configuration Manager responds with “Ignore” option for step 5b, then update PI database with defect details and Action status “Ignore.”

6. Check whether the PI database has test cases related to the defect(s), if not
   a. Notify the Configuration Manager with a message that the defect is fixed but is missing test cases. Provide him an option to “Remind Later” or “Ignore.”

7. Check whether the PI database has enough code coverage for the source files. If not, notify the Configuration Manager with a message that the defect is fixed but there are not enough test cases. Provide him the option to “Remind Later” or “Ignore.”

The Automator checks each “Deferred” row related to an each product stream.
- If auto-merge is in “Deferred” state, then repeat steps 5b to 7.
- If test cases are in “Deferred” state, then repeat step 6a.
- If code coverage is insufficient, then repeat step 7.
Benefits

Having Application release automation with Continuous Delivery and a Patch Integrator in place, some of the challenges can be mitigated:

- Proactive and auto identification of impacted branches of the product source whenever a change set is checked-in to a specific product source branch for addressing a defect found in the released version of the product.
- Automation of cross-merging a change set across product source branches and notifying the change set owner if the process needs manual intervention.
- An integrated view of test case coverage for the change set across product source branches, their delivered releases/builds and code coverage.
- Unified tagging across tools for easy identification and system level analysis of source files, test cases and defect identifiers. For instance, source files and test cases are tagged with a defect number across branches and versions of test cases.
- Tagging test cases also helps to mark test cases which are sensitive to the build environment, so that their failures can be ignored by the build system and promote the build to the next phase. Currently, builds need to be re-run after the environment cleanup which leads to resource consumption and delays in delivering the build.
Conclusion

Implementing a continuous integration or continuous delivery system requires a thorough design of the build and integration process. This paper primarily concentrated on the challenges of porting fixes across impacted product releases by the development teams amidst a continuous delivery process and its algorithmic implementation. The proposed solution will help in effectively track, identify, merge and manage the fixes across releases, thereby taking the continuous delivery process to next level. The system will also address concerns about defects that may span multiple builds and account for practices that will enable automated delivery of fixes across build images with minimal overhead.
CA Technologies (NASDAQ: CA) helps customers succeed in a future where every business—from apparel to energy—is being rewritten by software. From planning to development to management to security, at CA we create software that fuels transformation for companies in the application economy. With CA software at the center of their IT strategy, organizations can leverage the technology that changes the way we live—from the data center to the mobile device. Learn more about CA Technologies at www.ca.com.

© 2014 CA. All trademarks, trade names, service marks and logos referenced herein belong to their respective companies. ITIL® is a Registered Trade Mark of AXELOS Limited. The statements and opinions expressed in this document are those of the author(s) and are not necessarily those of CA. CA and the authors assume no responsibility for consequences resulting from the publication of or use of this document, and are not responsible for, and expressly disclaim liability for, damages of any kind.