AUTOMATION

OF

FIRMWARE CODE REVIEW

PACKET GENERATION PROCESS

_____

A Master's Thesis

Presented

To the Faculty of

California Polytechnic State University

San Luis Obispo

_____

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

In

Biomedical Engineering

_____

By

Udaykumar Rasiklal Shah

March 2011

© 2011

Udaykumar Rasiklal Shah

ALL RIGHTS RESERVED

COMMITTEE MEMBERSHIP

TITLE:	AUTOMATION OF FIRMWARE CODE REVIEW PACKET GENERATION PROCESS
AUTHOR:	Udaykumar Rasiklal Shah
DATE SUBMITTED:	March 2011
COMMITTEE CHAIR:	Lanny Griffin, Chair Department of Biomedical Engineering
COMMITTEE MEMBER:	Robert Crockett, Director General Engineering Program
COMMITTEE MEMBER:	Daniel Walsh, Professor Department of Materials Engineering

ABSTRACT

AUTOMATION OF FIRMWARE CODE REVIEW

PACKET GENERATION PROCESS

Udaykumar Rasiklal Shah

Peer code review is one of the best ways to immediately improve software quality, save cost and prevent rework. It improves software quality by finding defects earlier in the development lifecycle before software gets to customers.

Problems were experienced for generating a consistent code review packet due to lack of an automated process. It takes anywhere from four hours to more than a day based on midsize to the large size features. Errors were introduced due to the manual steps involved in the legacy process.

The automation of generating the code review packet was implemented using Perl programming and Beyond Compare diff tool in Clear Case version control environment to output the code packet in the PDF format.

Automation in generating the code review packet reduced the packet generation time by more than 75%. It translated the manual process to a simple, automated process which helped save time and rework. It removed all inconsistencies and made it easier for reviewers to understand and review the packet.

ACKNOWLEDGMENTS

I am very happy to dedicate this thesis, to give my deepest love and appreciation to my lovely parents, wife, son and other family members. It was impossible to achieve this goal without their continuous support and blessings.

I would like to express my deepest appreciation to my committee chair Dr. Lanny Griffin and my committee members Dr. Robert Crockett and Dr. Dan Walsh, for their mentorship, guidance and encouragement.

I would like to thank St Jude Medical for giving me a great opportunity to work on a very exciting project towards my Master's Thesis. Special appreciation goes to George Huang, my manager in the Firmware Development group and my colleagues from the same group who have been supportive all along my project. They have provided me with the legacy data based on their past experience and ideas for the improvements.

Last but not least I would like to thank all my friends and colleagues outside the Firmware Development group, who have directly or indirectly guided me in reaching this milestone.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1 Importance of The Quality Code Review

A Technical Review is defined as a documented examination of Software Development requirements, design, code, integration and verification documents in order to evaluate whether specifications have been met and to identify issues [1].

Thorough code review happens behind every quality delivery of the implementation code in the biomedical device products. It has been proven that code review uncovers defects, ensures maintainable code, and helps mentor new hires. Detailed code review ensures the quality of the code and adds more confidence to the development work completed.

Usually code gets reviewed by different firmware piers like systems requirements team, firmware requirements team, firmware verification, integration test team and by SMEs from the firmware team itself. All of them come from different backgrounds and different disciplines. It becomes really challenging for them to review the code if the code review packet is not standardized. It is very important that all the code reviews follow the same sequence, pattern and format. This will make the reviewers' job much

easier and they could focus more on the content of the code review material itself rather than trying to figure what needs to be reviewed and where to find the needed information.

1.2 Case Study

Below is the case study showing the importance of the detailed code review compared with the risk of the code delivered unreviewed [1]. As shown in Figure 1 “Cost Before Code Review” below, total bugs left after the development work were about 463. But after the verification testing, the bugs were down to 321. This is still very risky for the medical device industry as it causes big risks for the patients. It could cause life threatening behavior due to so many bugs left uncovered.

Compared to case above, if the same code goes through the detailed code review as shown in Figure 2 “Cost After Code Review”, it found 283 defects in the code review itself. It is a big achievement in terms of ensuring the high quality delivery of code. Not only that, looking at the total costs to fix the bugs before and after the code review, it saved more than $150k with the case when the code was thoroughly reviewed. Specially in the medical device products, even a single bug left uncovered could cost many lives and millions in losses. It is not acceptable to deliver medical devices with any bugs at all.

Also, it is desired to have the 100% code coverage for each of the features implemented in the medical device product software [8]. By doing the formal code review, we could ensure the code coverage as well.

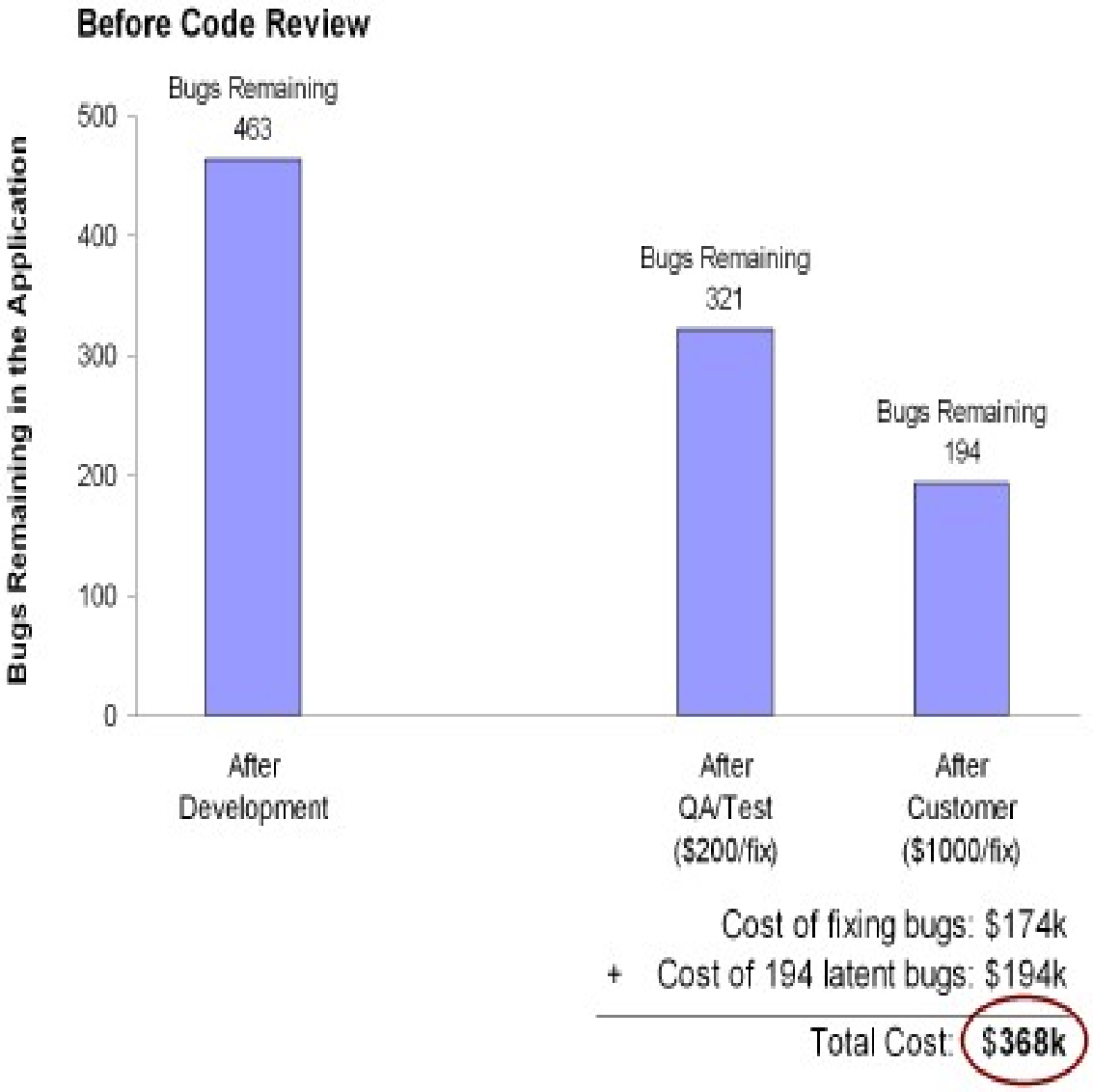

Figure 1 Cost Before Code Review

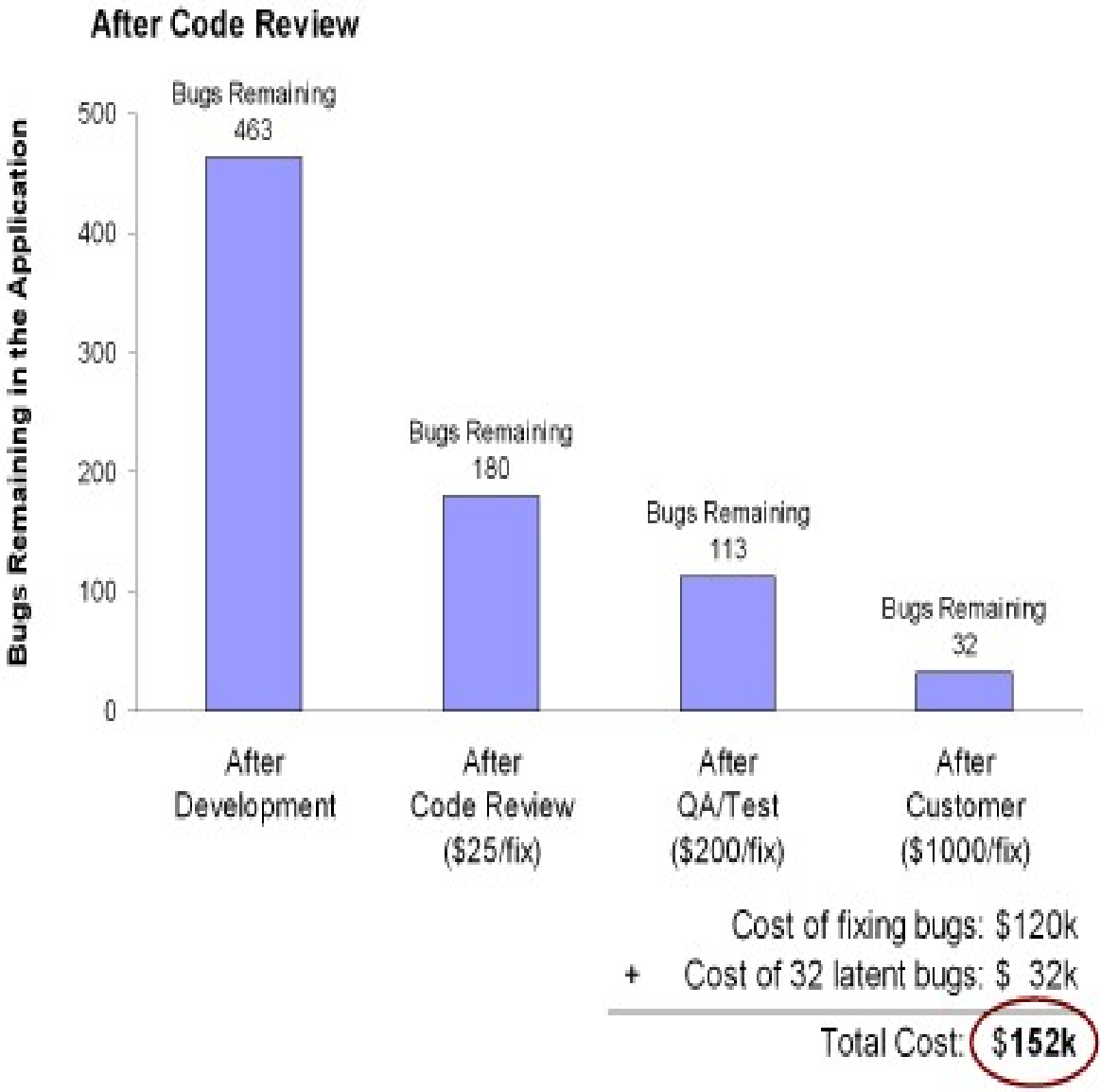

Figure 2 Cost After Code Review

1.3 Common Problems

A problem seen for generating the consistent code review packet is the lack of processes defined to be followed by the entire firmware development team to generate the code review packet. It is very time consuming with the current resources to generate the code review packet which could take from four hours to an entire day and sometimes even more than a day based on midsize to the big size features. There are lots of manual steps involved to generate the code review packet which is prone to errors.

Any errors and inconsistency in generating the review packet make it harder to review it and cause delay in understanding the review material. Inconsistencies in the way the code review packets are generated cause delays in reviewing the feature implementation, and thus impact the schedule of software deliveries.

In order to overcome such problems and to achieve high quality code delivery, it is most important to come up with a simple, automated, integrated process to generate the code review packet. It is necessary to define a common style and format of the code review packet for all of the firmware projects and across all the development sites. Also, in order to deliver the high quality code within the defined schedule, it becomes obvious to reduce the time to generate the code review packet by more than 50% of the time it takes today.

CHAPTER II

OBJECTIVES

2.1 Project Objective

The main objective of the code review packet generation tool is to reduce the code review packet generation process time by more than 50% and to minimize the manual steps by replacing them with the automated process. The 50% time reduction relates to the time it takes for generating the code review packet using the process defined for the current AFM\HV\LV firmware projects which is anywhere from four hours to longer than a day based on small to big size firmware features.

2.2 Project Requirements

Following are the main objectives of the Code Review Packet Generation Tool:

1. A primary goal is to reduce the code review packet generation time by more than 50% of the actual time it takes using the existing process defined for the AFM\HV\LV projects.
2. To enforce firmware processes by integrating them with the automated process of generating the code review packet.
3. Common style and format of the code review packet for all firmware projects and across all CRMD sites.
4. To come up with a simple, automated, integrated process to generate the code review packet.
5. Depending on the size of the feature, to save from 2-6 hours of developers' time for generating each of code review packet.
6. To get rid of manual steps involved in the code review packet generating process which are prone to mistakes [8].
7. By automating the process, minimize the commonly made mistakes like missing the changed data and newly added files, no line numbers, missing file names, etc.
8. To remove dissimilarities in the code review packet generation process and output across all CRMD sites and firmware projects.
9. To make the code review generation process easy to follow and the review packet easy to read, understand and review.

10. Define a clear and automated process to generate the code review packet for the AFM\HV\LV platforms.

11. To develop a Code Review Packet Generation Tool that is compatible with the existing tool set like Clear Case for the version control [7], and Beyond Compare to generate the diff report for the changed files.

12. To generate the code review packet in a PDF format for everyone to be able to access and review it easily.

2.3 Schdule & High Level Deliverables

Based on the project scope and main objectives described in the previous sections, this project has been allocated eight months from gathering the user requirements to the delivery of the Code Review Packet Generation tool. Below are the high level activities associated with this project:

Project Activity & Deliverables	**Duration**
User feedback and requirements gathering	1 month
Collecting actual data based on the current Unity based projects	1 month
Design	1.5 months
Implementation	2 months
Testing	1 month
User Manual	0.5 month
Training and Tool Demo	0.5 month
Delivery of the tool	0.5 month

Table 1 Project Schedule and High Level Deliverables

CHAPTER III

MATERIALS & METHODS

In order to resolve the issues with the code review packet generation process today, detailed analysis of the existing process needs to be performed to identify the areas for the improvements. Also, detailed study of the various Clear Case commands [7], comparison tools, Beyond Compare diff reports, user interviews and feedback need to be performed to come up with the right solution.

3.1 User Feedback and Data Collection

In order to understand the user concerns and issues, user interviews were conducted to collect the data based on ongoing firmware projects. Based on the user feedback from the Atrial Fibrillation Monitoring project, the followng data was collected:

Firmware feature Name	Time To Generate The Code Review Packet
Stored Electrogram (SEGM)	16 hours
Monitor Enable Disable (MED)	6 hours
Diagnostics (DGNS)	8 hours
Programming (Prog)	8 hours

Table 2 User Feedback and Data From AF Monitor Project

Based on the data collected in Table 2 above, each developer spent on an average four to six hours to generate the code review packet. It became very clear that an automated process is needed to cut down the code review packet generation time to less than half of what it takes today.

Firmware developers were completely unsatified with lots of manual steps involved and the lack of a unified process to be followed across the projects in Cardiac Rhythm Management Division.

3.2 Design Analysis & Implementation Technologies

The following Figure 3 “Technical Code Review Flow Diagram” is the flow diagram showing the high level steps involved in the technical code review process [9]. Given there are many critical steps involved and so much dependencies for the future deliveries, it is very important that the code review packet is prepared accurately, timely and in the standardized error-free format.

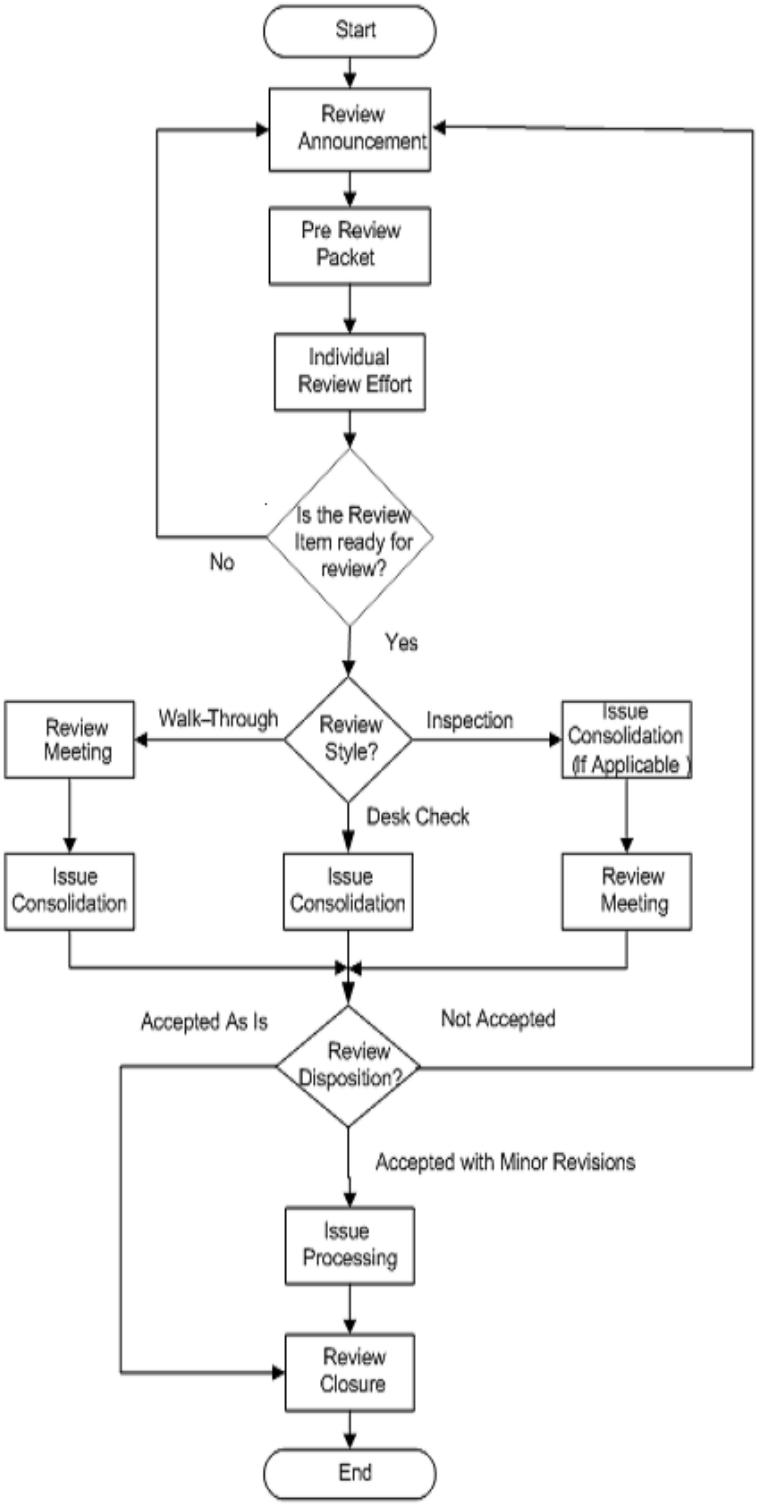

Figure 3 Technical Code Review Flow Diagram

The most important technology factor was in choosing the diff tool which gives the diff report which is easy to understand and review. The diff reports from both ClearCase and Beyond Compare were compared and taken to the customers' feedback. Based on the feedback and diff reports analysis, it was decided to use Beyond Compare to generate the diff report. All the output generated by Beyond Compare diff report is exported to the .txt format which will be further procesed to be converted to the .PDF format.

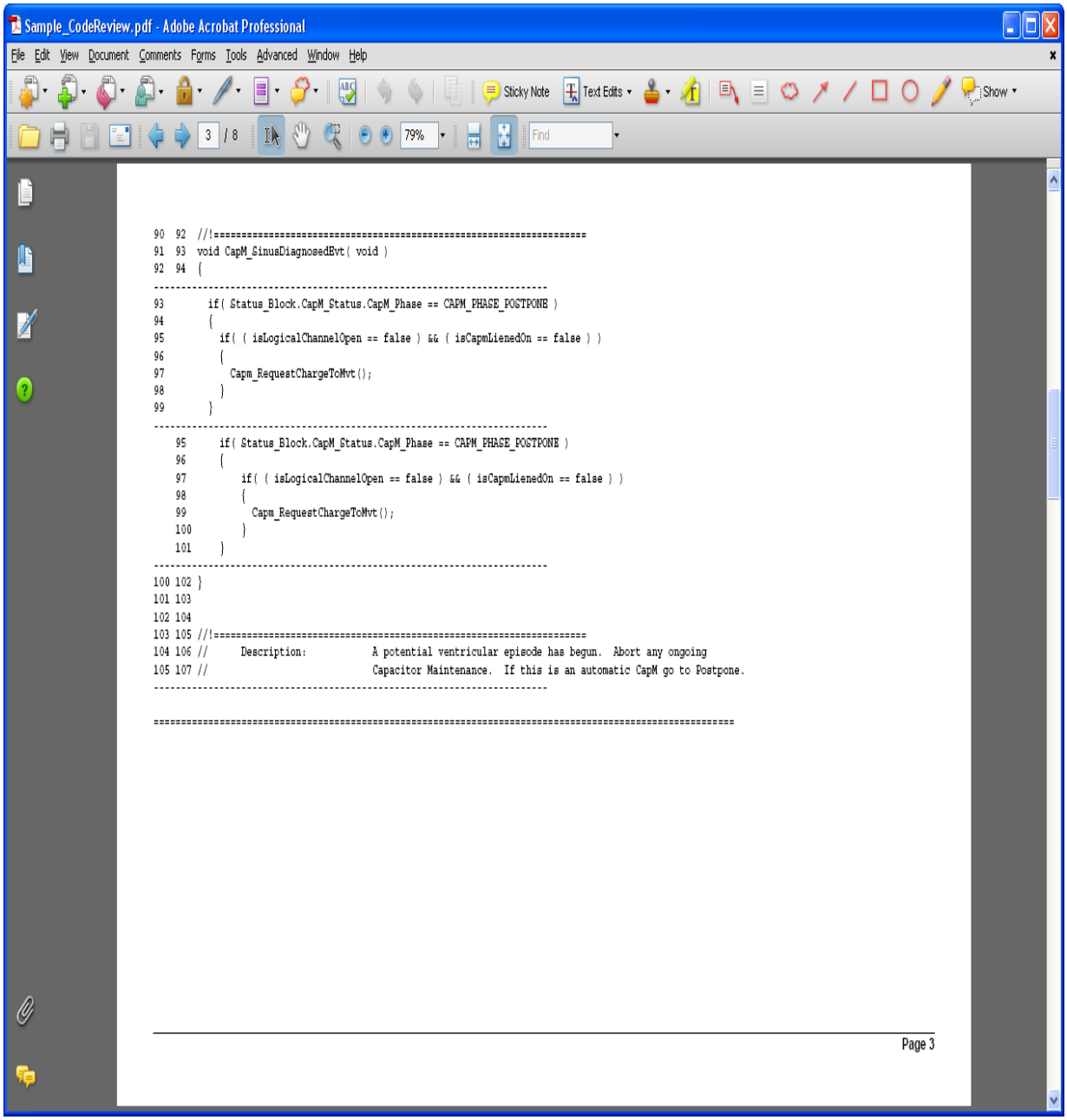

Figure 4 Beyond Compare Diff Report Sample

Several programming languages like Perl [5][6], C [3], C++ [4] and programming algorithms [2] were evaluated to develop the code review packet generation tool. Given Perl is the most robust, easy to implement and to maintain [5], it was decided to use the

Perl programming language to develop this tool. Perl has all the support to develop the features to fulfill the requirements for this tool. Perl also helps in converting lots of manual steps to the automated process in ClearCase [7].

3.3 Design To Make The Tool User Configurable

It is very important to design the tool to make it user configurable; to give users the freedom and output they need. Below are the configuration options.

A. Diff type: There are two types of diff reports which could be output using Beyond Compare. One type of diff report is the “context text only” which is easy to use if there are only few changes in the given source file. The other type of output is the diff report with “full source” where the output will include the complete source file along with the changes. The second type is more important when the changes are spread across several sections in the source file; just the context diff report will not be sufficient for reviewers to understand the changes. Users will have freedom to choose between these two types of the output reports.

B. Email option: User will have an option to automatically email the generated code review packet PDF file once it is produced or skip the emailing part and grab it from the local directory.

C. Unit tests: Unit tests must be executed once changes are made to the source files irrespective of how small\big the changes are. User will have an option to select among the projects to run the unit tests. Users can select different platforms to run the unit tests.

3.4 Technologies To Implement Code Review Packet Generation Tool

There are several technological areas to be investigated to come up with the right solution to implement the objective and requirements for the code review packet generation tool [9]. Clear Case is one of the most important among them as all the source files for the new\changed features are version controlled under Clear Case. Clear Case commands related to retrieving the baselines associated with the most recent and previous versions to produce the diff report, child stream vs parent stream and other related information would be executed to retrive the necessary information and data.

One important factor was to use the open source technologies to minimize the cost involved in developing, distributing and using the code review packet generation tool at the customers' terminals. Perl will be used to develop the tool and an open source PDF API for Perl will be used to convert the diff report to the PDF format [6]. With this API, the PDF document can be printed in the landscape mode to prevent line wrapping problems. The font size could be set and the page# could be displayed as well. With the Perl in build functions, the output could be set to sort by feature name, followed by the .h, .c and unit tests files to make it simpler for the reviewers to follow.

CHAPTER IV

RESULTS

4.1 Results

The author is very happy to report achievement of the main objective of this project to reduce the code review packet generation process time by more than 50% and to minimize manual steps by replacing them with automated process. The 50% time reduction related to the time it took for generating the code review packet using the process defined for the Unity projects which is anywhere from four hours to longer than a day based on small to big size features. As shown in Figure 5 below, it took from six hours to 16 hours for the midsize to the large size features using the manual process. When the code review packets were generated using the newly implemented automated tool, it took just 30 minutes to one hour for these features which reduces the packet generation time to well below the original objective of 50% reduction in time.

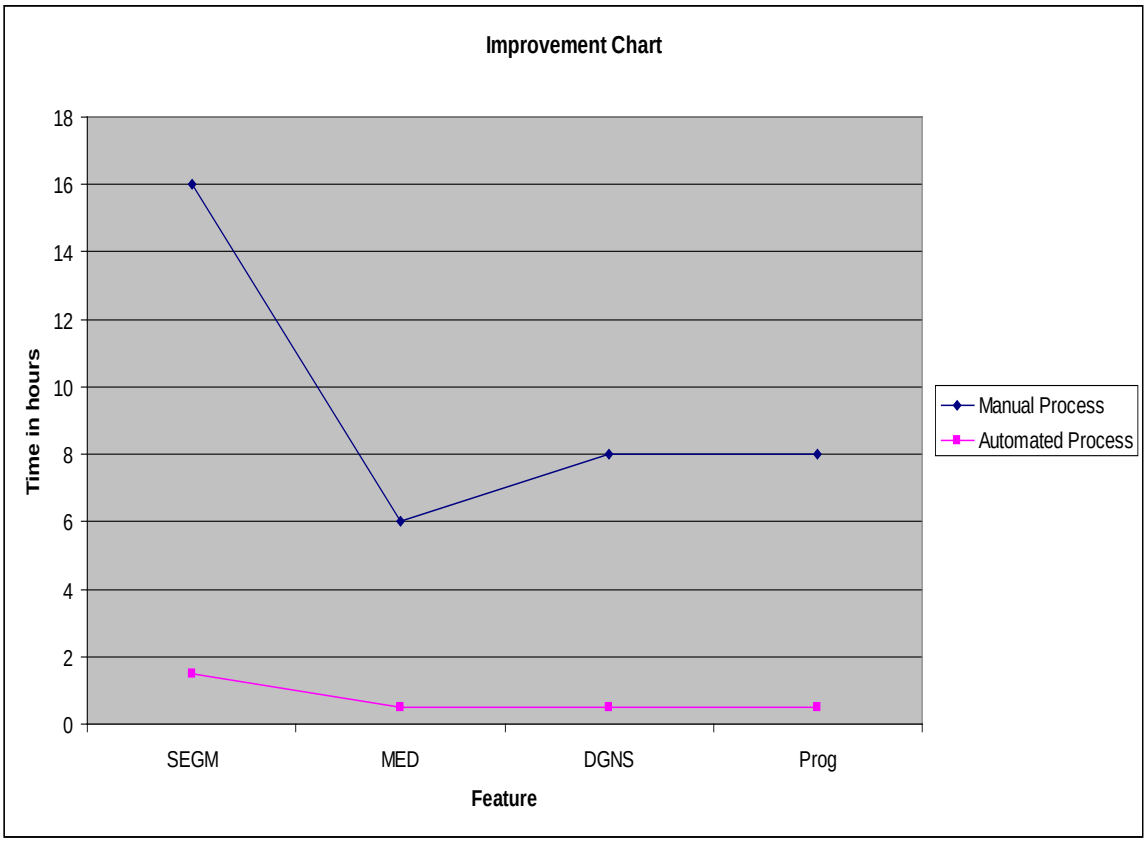

Figure 5 Comparison Between Old Process and New Process

Not only achieving the time reduction objectives, the author was also successful in reducing most of the manual steps involved by automating them and making them part of the packet generation tool. It just became a single step process to generate the code review packet generation tool.

Figure 6 below shows the change set which includes all of the files changes\added, their child and parent versions and locations where the changed files are located. This is printed on the first page of the report for reviewers to get a quick idea of the number of changes, features changed, location and their version details.

Figure 7 shows diff report generated using the Beyond Compare along with the original file data on the left and the changed file on the right hand side. It also prints the line numbers to be referred especially during phone meetings.

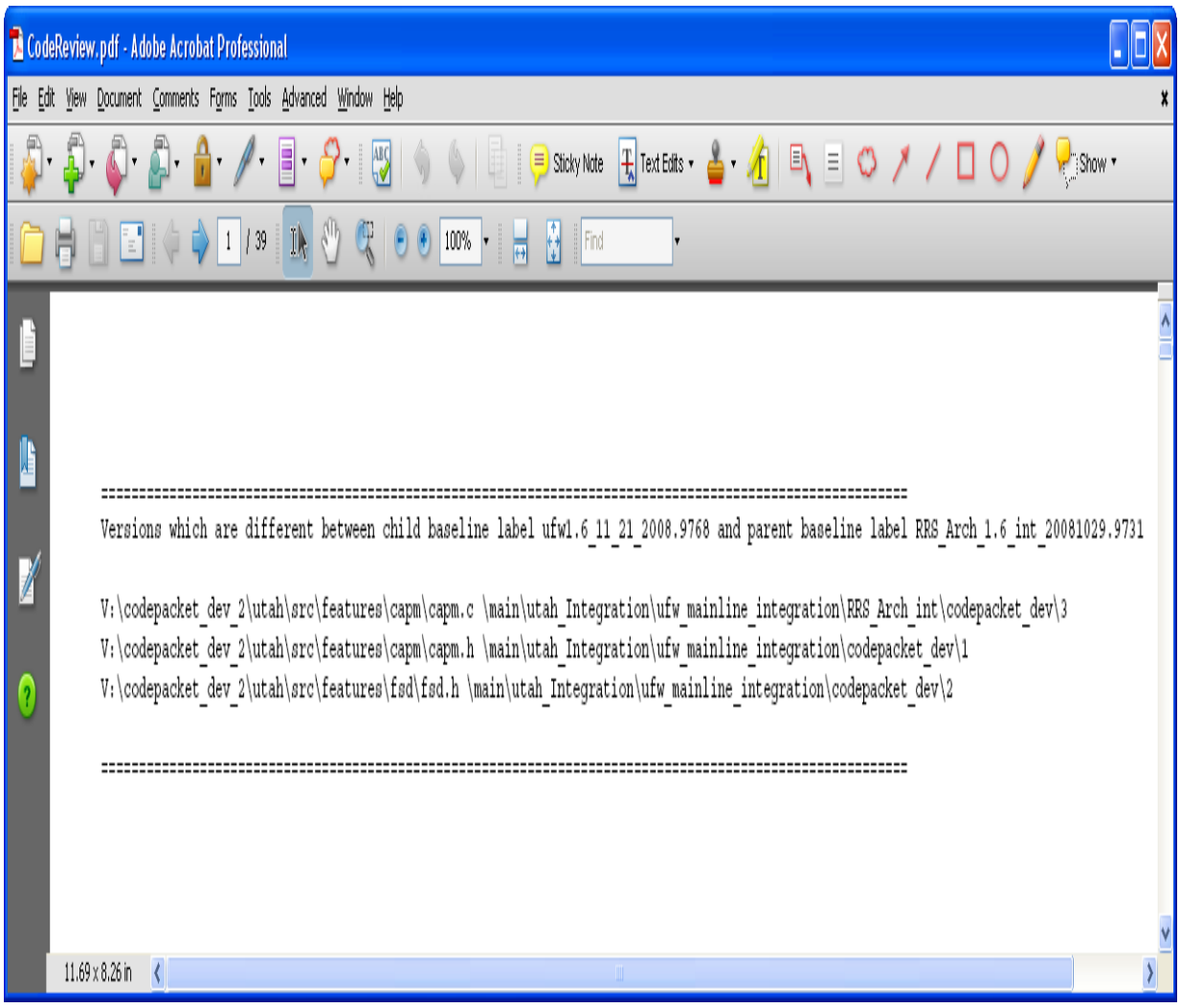

Figure 6 High Level Changed Set

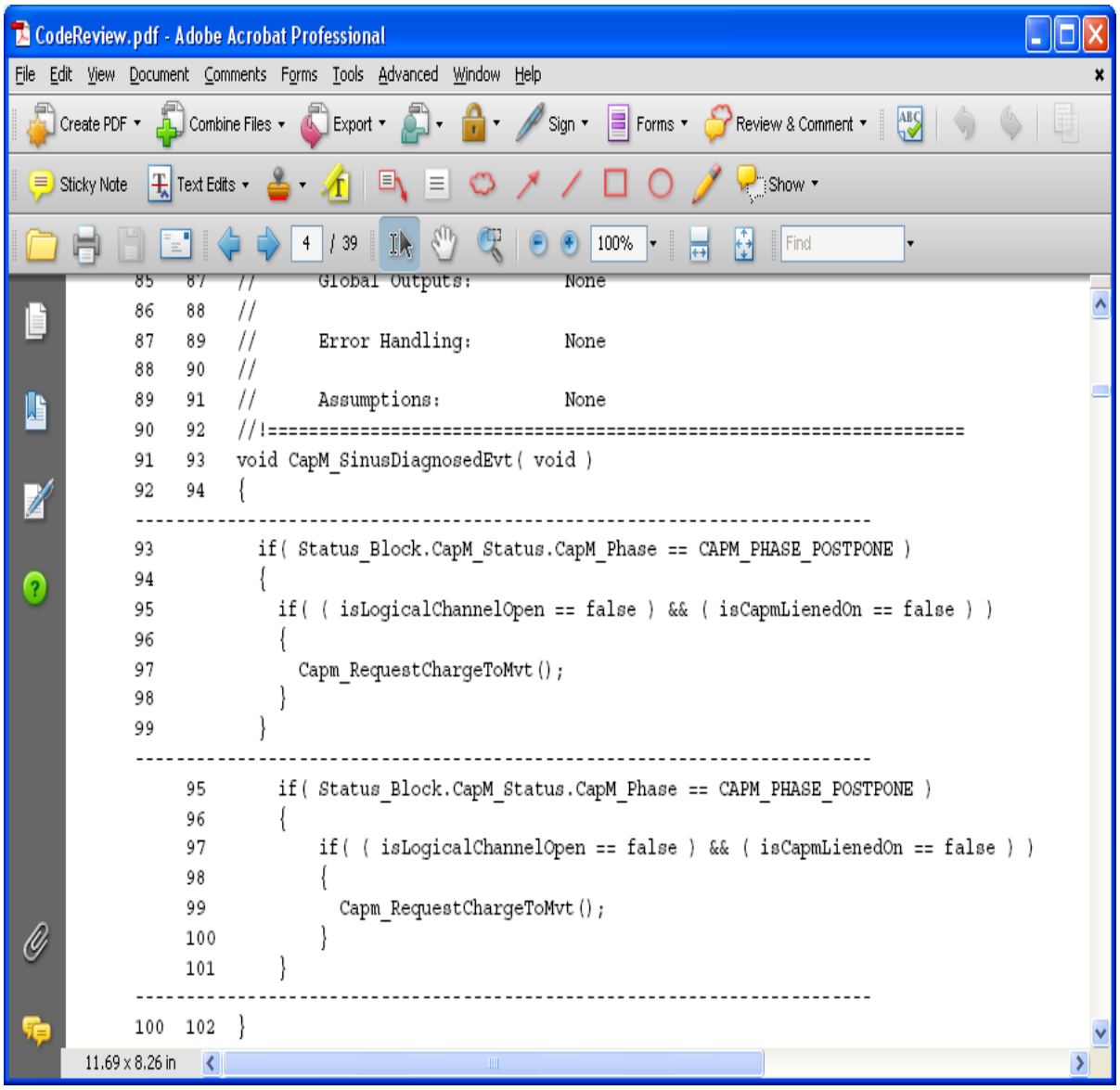

Figure 7 Diff Report - Includes Line #, Version #

Figure 8 shows the unit tests results run for the modified feature. Once any feature gets added\modified, the unit tests will automatically be run by the code review packet generation tool to make sure unit tests are updated along with the original code changes. Running unit tests for each of the features modified helps catch the bugs early.

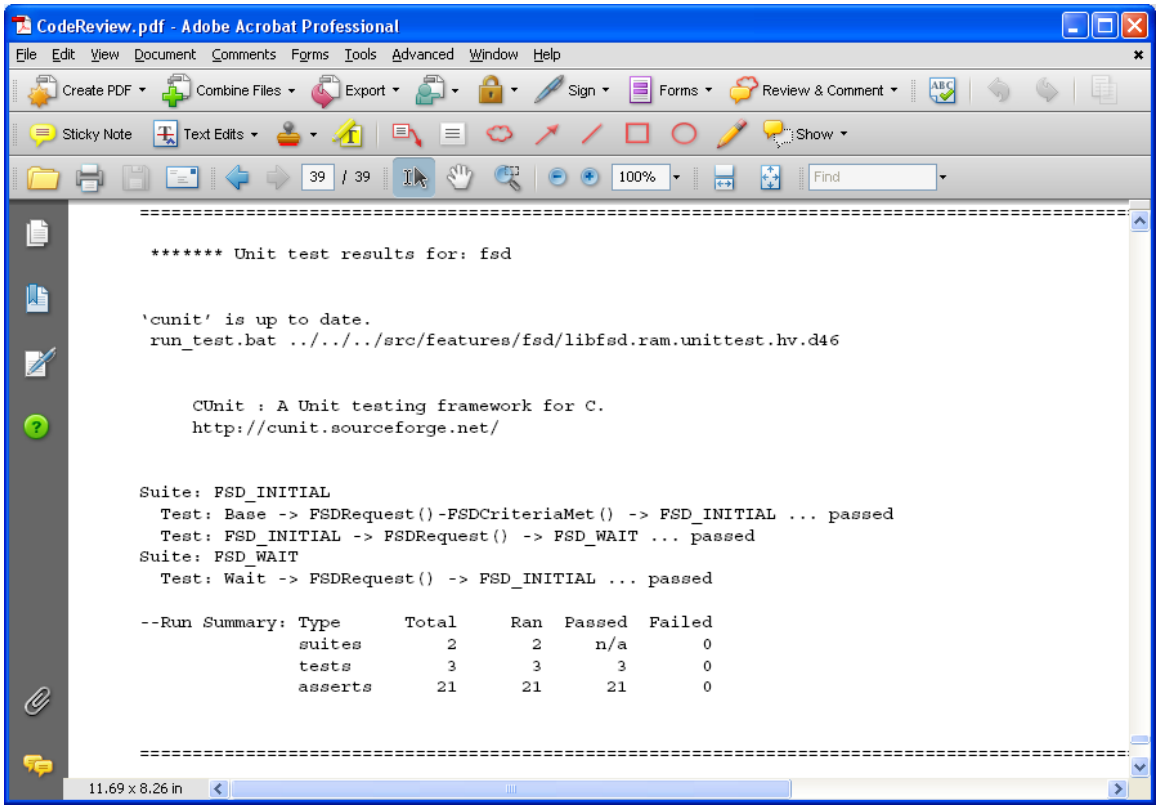

Figure 8 Unit Tests Result

4.2 How To Use The Tool & User Guide

1. Please make sure to have the steps below completed before generating the code review packet.

    a. Check in all the checked out files to the Clear Case

    b. Add the newly added files including the code coverage file to the source control

    c. Rebase your working stream

    d. Make a baseline for your working stream.

        How: Option available by right clicking your stream in the Clear Case Project Explorer (no need to recommend a baseline)

    e. DO NOT deliver the changes to the parent stream yet

    f. Code review packet generation tool has to be run from the view where you want to generate the review packet

2. Open up the command prompt window and go to:

    \your_working_view\ tools\code2text\

3. Just type >Code2Txt.bat followed by the arguments as mentioned in the screen shot below.

    >Code2Txt.bat [diff_type] [email_option] [unit_test_option]

        [diff_type]

-a diff with the complete file

-c context diff only

default -a

[email_option]

-n No email will be sent. Review packet will be saved in %temp%

dir

default Review PDF will be emailed out to %USERNAME%

[unit_test_option]

-hv HV unit test under all_targets

-lv LV unit test under all_targets

-afm AFM unit test under Regression_test

default unit test under current_targets

4. For help with arguments, please type in: >Code2Txt.bat -h

4.3 How To Integrate The Tool In To The Specific Project

The steps below at the project level Integration Stream need to be performed so all of the development streams under the same project could get it for free just by rebasing to the project integration stream.

Alternatively the following steps could be followed at the development level streams as well.

1. Copy Code2Txt.bat file to the source directory
2. Recommend a baseline if previous step was done at the project/feature level integration stream so child streams could rebase to get this tool.

CHAPTER V

DISCUSSION

5.1 Voice Of Customers

The code review packet generation tool was demoed to the CRMD firmware development technical leadership teams to present the features implemented and to seek their feedback before delivering the tool to the entire team. The tool was highly appreciated. It was taken as a big positive for the quality delivery of the firmware features. The leadership team was really happy seeing the time reduction in generating the code review packet. They were glad to see the manual process translated into a one step automated process.

The code review packet generation tool was also demoed to the all three CRMD sites including Sunnyvale, Sylmar and Sweden. Since all of the developers will be using this tool, it helped making it a standard process to be followed across the entire department for the code reviews. Even the other pier groups involved in reviewing the firmware code changes were trained on how to interpret the information, and where to find the necessary changes to catch the bugs sooner.

Overall, the customers were really happy having this tool available to replace the old time-taking error-prone process with a single step automated process which helps reduce the time to generate the code review packet by more than 50%.

5.2 Strengths and Weaknesses

Like every other product development, the code review packet generation tool has its own strengths and weaknesses.

Below is the list of the major strengths:

1. A simple, automated, integrated process to generate the code review packet.
2. Enforce the firmware processes by integrating them to the automated way of generating the code review packet.
3. Common style and format of the code review packet for all the firmware projects and across all CRMD sites.
4. Reduce the code review packet generation time by more than 50% of the actual time it takes with the current manual process.
5. Depending on the size of the feature, this tool could save from two to six hours of developers' time for generating the code review packet per feature.
6. Get rid of manual steps involved in the code review packet generating process which are prone to mistakes.
7. By automating the process, minimize the commonly made mistakes such as missing data and files, no line#, unable to find which side is original and which side is modified, missing file names, etc.
8. Make code review packet\material easy to read, understand and review.

9. Remove nconsistencies which could cause delay in reviewing the feature implementation, and thus impact the schedule of feature delivery.

10. User configurable support to generate the context only diff report or the complete diff report.

11. Make everything executable off the ClearCase so we don't need separate installation of Perl and Beyond Compare on end users' computers.

12. Generate the complete changes set for the given child development stream

13. Print the documents as per the order in the change set

14. Source files for the features followed by their unit test and codecoverage files

15. Compares child stream's baseline with the parent stream's baseline and find all changes made in the selected child stream.

16. Line numbers are printed for all the source files.

17. File name and page number show up in the file header.

Even though the list of the strengths outweigh the weaknesses, it is very important to note the weaknesses for future improvements. Below is the list of the weaknesses:

1. There are few hard rules in terms of certain steps to be performed in the Clear Case; for example, check-in all checked-out files before generating the diff report, make a baseline and recommend in order to diff with the parent stream.
2. The automated diff comparisons are only made between the most recent and the immediate previous baseline. There is no way for users to specify the comparison between the most recent version with the older versions.
3. Coverity reports are neither run nor included in the PDF report.
4. Code review check lists are not included in the code review packet.

CHAPTER VI

CONCLUSION & FUTURE WORK

6.1 Conclusion

Quality is the key for the companies like St Jude Medical developing and delivering biomedical devices including pacemakers, ICDs and other heart related products. It is really important to have each and every step involved in the product development process error-free, robust and reviewed by the key reviewers with the greatest depth to prevent any future recall issues. The Code Review Packet Generation tool helps in removing all of the manual steps involved in generating the code review packet to review either the newly developed feature or changes to the exisitng feature. By automating this process, it removes all of the errors that could happen when performed manually. Not only that, it reduces the time to generate the packet by more than 50% of the time it takes manually.

The code review packet generation tool helped achieve reducing the code review packet generation process time by more than 75% and in minimizing the manual steps by replacing them with the automated process. The 75% time reduction relates to the time it takes for generating the code review packet using the process defined for the Unity

projects which is any where from four hrs to longer than a day based on small to big size firmware features.

The Code Review Packet Generation tool helps generate the review packet in common style and format for all firmware projects and across all CRMD sites. It translated the old process to a simple, automated, integrated process to generate the code review packet. It helps in removing all of manual steps involved in the code review packet generating process.

The Code Review Packet Generation Tool is compatible with the existing tool set like Clear Case for the version control, and Beyond Compare to create the difference among the changed files.

6.2 Future Opportunities

Even though all of the requirements gathered through the voice of customers and feedback from users are implemented, there are always opportunities to enhance this tool and take it to the next level by automating more steps involved upon any level of code changes.

The author sees the followng being the immediate future opportunities to enhance this tool further:

1. Explore automatically generating the unit tests results specific to the project without specifying the platform and save them to the output PDF
2. Explore automatically generating the code coverage report and integrate in to the code review packet generation process
3. Provide users an option to upload the file directly to the given network location or the Share Point location
4. Add the code review check list to the master PDF report before adding the diff for the changed files

REFERENCES

[1] Jason Cohen, Steven Teleki and Eric Brown, *Best Kept Secrets of Peer Code Review*. Austin, TX: Smart Bear Inc., 2006.

[2] Thomas H. Cormen, Charles E. Leiserson and Ronald L. Rivest, *Introduction To Algorithms*. *Eastern Economy Edition.*New Delhi: Prentice-Hall of India Private Limited, 1998.

[3] E Balagurusamy, *Programming In ANSI C. Second edition.* New Delhi: Tata McGraw-Hill, 1997.

[4] Deitel & Deitel, *C++ How To Program, Second Edition,* New Jersey, NJ: Prentice Hall, 1997

[5] Tom Christiansen and Nathn Torkington, *Perl Cookbook: Solutions and Examples For Perl Programmers. 1st Edition*, Sebastopol, CA: O'Reilly & Associates, Inc., 1999.

[6] Nathan Patwardhan, Ellen Siever and Stephen Spainhour, *PERL In A Nutshell, 2nd Edition,* Sebastopol, CA: O'Reilly & Associates, Inc., 2002.

[7] Tom Milligan, *"Using Perl With Rational ClearCase Automation Library"* 2004. Retrieved Nov10, 2010 from World Wide Web: http://www.ibm.com/developerworks/rational/library/4711.html.

[8] E. Dustin, J. Raska and J. Paul, *Automated Software Testing: Introduction, Management and Performance.* New York, NY: Addison-Wesley, 1999.

[9] R. S. Pressman, *Software Engineering: A Practitioner's Approach, 4th edition.* Berkeley, CA: Osborne/McGraw-Hill, 1997.

Appendices

Appendix A Acronyms

AFM:	Atrial Fibrillation Monitoring
CRM:	Cardiac Rhythm Management
CRMD:	Cardiac Rhythm Management Division
HV:	High Voltage cardiac devices such as Implantable Cardioverter Defibrillator (ICD)
LV:	Low Voltage cardiac devices such as Pacemaker
PDF:	Portable Data Format
SME:	Subject Matter Expert
SRS:	Software Requirement Specifications

Appendix B Definitions

Baseline

A baseline specifies one version of each element in a component and represents a version of the entire component at a particular stage of its development. As work on a component progresses, the project manager periodically creates new baselines and gives each one a promotion level that indicates its maturity or readiness for use by the team.

Beyond Compare

Beyond Compare is a utility for comparing files and folders. It can help find and reconcile differences in source code, folders, images and data.

Code Coverage

Code coverage is a measure used in software testing. It describes the degree to which the source code of a program has been tested. It is a form of testing that inspects the code directly and is therefore a form of white box testing.

Feature

Feature is defined as a collection of set of requirements belonging to the same domain of the clinical functionalities.

Firmware

Computer programming instructions that are stored in a read-only memory unit rather than being implemented through software.

Source Code\Code

Code contains variable declarations, instructions, functions, loops, and other statements that tell the program how to function.

Unit Test

Unit testing is a method by which individual units of source code are tested to determine if they are fit for use. A unit is the smallest testable part of an application.

Appendix C Manual Process To Generate Code Review Packet

User guide to generate Firmware Code Review Packets:

Step 1: Print all the files to be reviewed to PDF format:

You can print a document to a PDF file if you have Adobe Acrobat Standard installed on your machine.

Before you print your file select Adobe PDF as your printer selection. This will direct the printing output to a newly created PDF file. Follow these steps:

1. Open the code file you wish to print on the editor of your choice.

   For Visual Studio:

   Recommendation: Make sure that the files include a header with the name of the file. This will facilitate browsing for a specific file on the packet.

2. Select File → Page Setup…
3. On the Page Setup make sure that the Page Header box is selected.

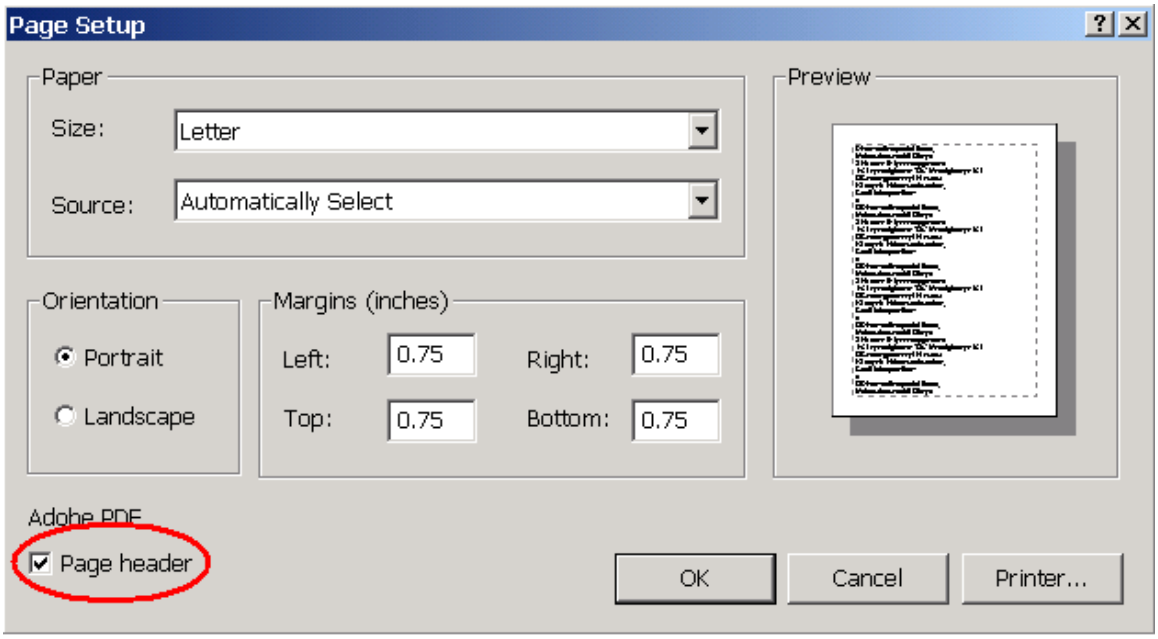

Figure 9 Code Review Packet Page Setup

4. Close the Dialog Box
5. Select File → Print…
6. When the printing menu appears click on include line numbers on the bottom left corner.

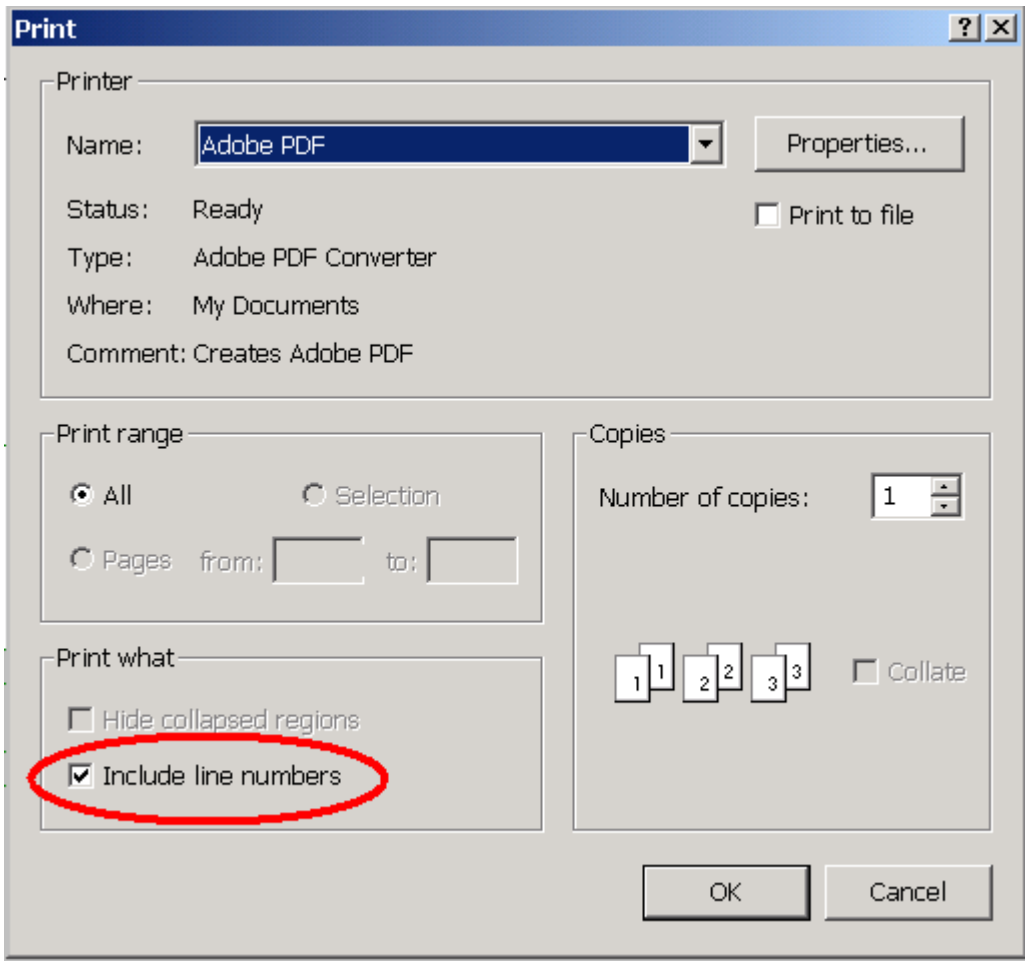

Figure 10 Code Review Packet Print Setup

7. In the name selection box select Adobe PDF and click on ok.
8. Enter the name for the PDF file to be created and save the file to a known location.

Step 2: Combine all the PDF files into a single PDF file

1. Open Adobe Acrobat Standard
2. Select File → Create PDF → From Multiple Files…
3. Click on the browse button to add your files to combine

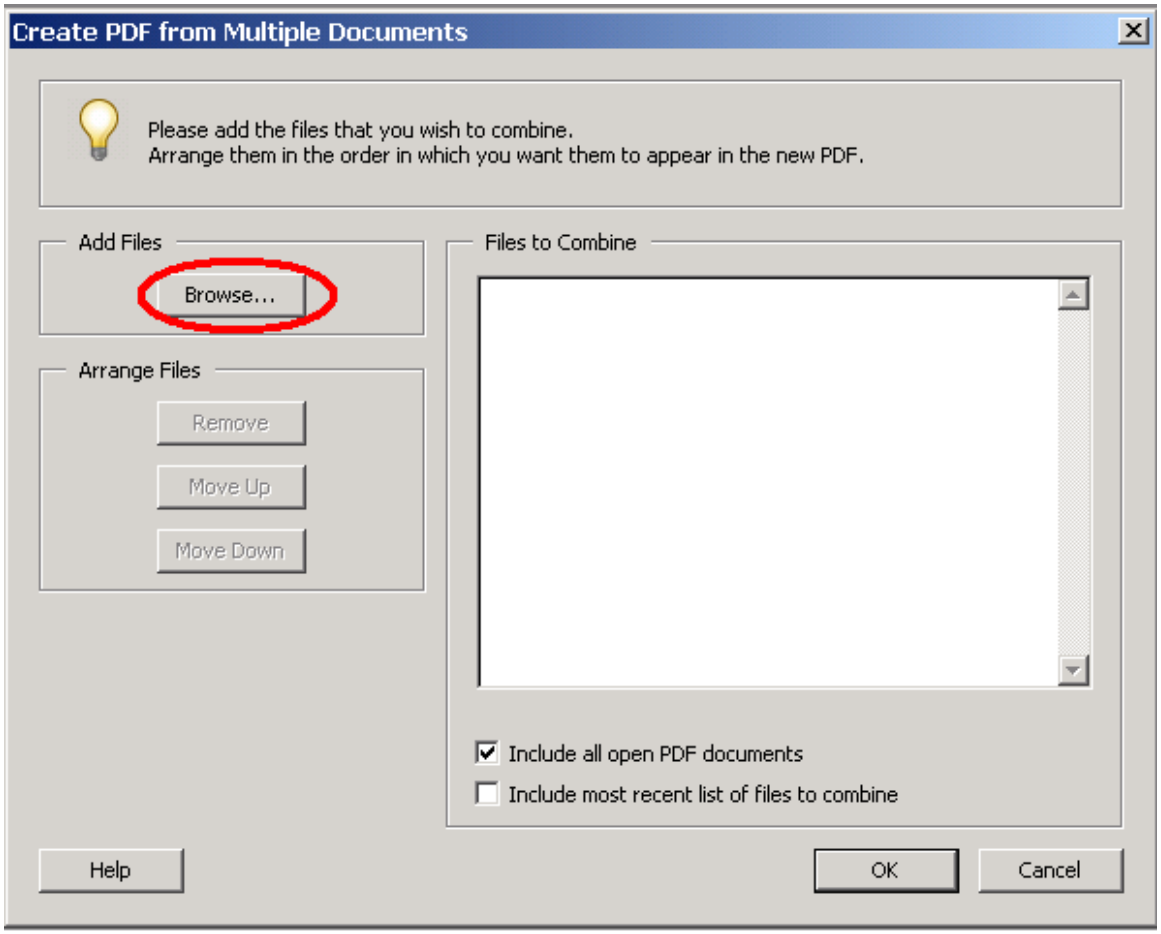

Figure 11 Code Review Packer Merge PDFs

4. The list of the files should appear on the list box, you can organize the order of the files by clicking on the Move Up and Move Down buttons.
5. Click ok to generate the packet.

Step 3: Add a footer that displays the packet page number

In Adobe Acrobat Standard with the final document open:

1. Select: Document --> Add Headers & Footers...
2. Select the footer tab.

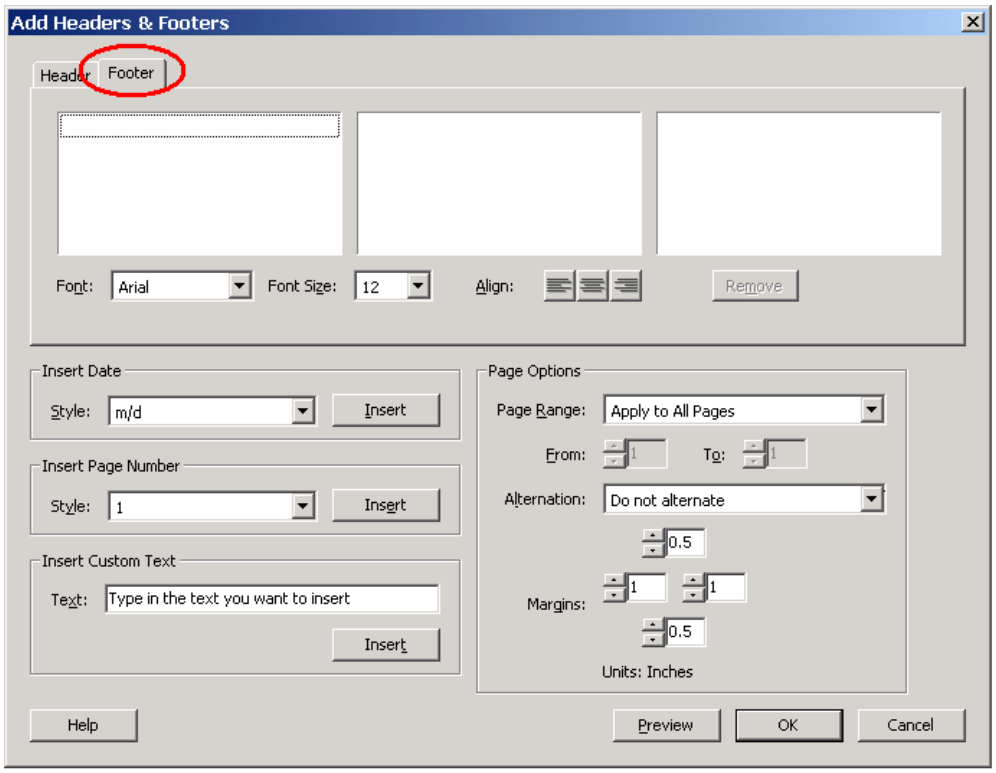

Figure 12 Code Review Packet Header Footer Details

3. In the “insert page number” section select the style and click “insert”. The page number will appear on the leftmost box on the top.

4. Center the page number by selecting it from the top box and clicking the centered text button.

Click on OK to add the page numbers to all the files.