Improvement of

Microsoft Office Build System Feedback Mechanism

NUI MAYNOOTH

Ollscoil na hÉireann Má Nuad

Department of computer science

Miao Li

January 2010

Supervisor: Rosemary Monahan

A thesis submitted in partial fulfillment of the requirements for the M.Sc. in Software Engineering

Declaration

I hereby certify that this material, which I now submit for assessment on the program of study leading to the award of Master of Science in Software Engineering, is entirely my own work and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: ______________________ Date: __________________

Table of Content

Abstract

Microsoft Office becomes increasingly large and complex, as well as its build system. The complex and unique build system for Office is constantly perfecting.

Because the variety of software products the build system and process also very different. But all the company has the same goal: to make quality software. For large and with long release lifecycle project the build process is even more important.

My project is especially under Microsoft Office Build System. And improve the feedback mechanism to report build break to developers, which can reduce builder workload, saving time, and sent useful message to the correspond developer. Whether a build system can give effective feedback is essential for improving the software quality.

Internship and Project Overview

About Microsoft

Microsoft was founded in 1975. The company has now become largely successful. As of 2008, Microsoft has global annual revenue of US$ 60.42 billion and nearly 90,000 employees in 105 countries. Microsoft first opened its doors in Ireland in 1985, located at Sandyford in Dublin almost 2,000 employees. Microsoft's operations in Ireland include software development and testing, localization, operations, finance, IT, HR and sales & marketing, both here in Ireland and across Europe, Middle East and Africa. More information about Microsoft Ireland at the official website [1].

Microsoft's most profit products are Microsoft Window operating system and Microsoft Office suit of productivity software. According to recent Forrester Research, as of June 2009, some version of Microsoft Office is used in 80% of enterprises and the latest Office versions hold roughly 80% of those installations. Some certain functions Microsoft had to offer may be not better than some companies' standalone applications, but Microsoft Office is a package of many different applications and they become more powerful when work together. Microsoft also positions Office as a development platform for line-of-business software under the Office Business Applications brand. More information about Microsoft Office products at Office Online [Office]

"One of the jokes in the IT industry is that the biggest competition for Microsoft Office is previous versions of Microsoft Office," Jason Hiner TechRepublic editor-in-chief. Some of Microsoft's competitors such as IBM and Google also has launched its own product to compete with the Office, like IBM's Lotus Symphony tend to be more low-priced, simple, easy integrated with other applications and support a standard file format OpenDocument Format. "In the past, large-scale application software, when comparing who has more functions the greater the advantage of whom," Suarez-Potts said. "Now, this design concept has become obsolete. The future is small, portable applications in the world, they adopted the same format and design of logic seamlessly combine together to accomplish a task. Open or proprietary -- - This is not a problem“ Even though Microsoft is a giant, but the response is not slow, in Office 2010 a big new feature is Office Web applications, lightweight online versions of Word, Excel, PowerPoint and OneNote to compete with similar cloud-based suites like Google Applications.

Internship Introduction

My internship is carried on Ireland Office Setup And Release team (OSAR); the OSAR team has built up a strong working relationship with Redmond team over the last 10 years.

The work of the OSAR is split into three separate areas therefore most team members play an active role in more than one of these three areas:

- build operation
- build development
- setup infrastructure and authoring

I mainly work on build operation as a build operator also known as builder, well builder also refers the tools for build software. We sharing build work between Dublin and Redmond build team. The main goal of build shared operations is to reduce the build turnaround time while still maintaining quality. This is achieved by taking full advantage of the eight hour time zone difference between the two locations. For two remote team work together, effective communication is very important. Therefore, we use tools such as SharePoint forums, Email, Conference calls, Office Communicator and Live Meeting etc.

My daily work includes the following points:

- open bugs for build break and system failures accurately contain detailed failure data
- actively push for the timely resolution of all build issues by self-investigation and resolution of problems and also escalation of larger issues to the appropriate parties
- report all issues that may have delayed a build and identify trends in the build process
- provide precise and accurate handoff communications highlighting particularly any issues that occurred with the build during Ireland hours

The above points are also the keys to success on build operations. We are babysits of builds, always concerned the status of builds, solve any problem or notify the person who can fix it.

I am also work on setup, troubleshoot and maintain Terminal Servers in our lab. The Terminal Servers are hosting builds for testing by other teams. Specific work includes patch the server, report network failure, provide access to these Terminal Servers by design and implement a web page use SharePoint.

The Build Development work that has been taken by our team mainly includes:

- build tools development
    - ➢ update to build break logging automation

    - efficiency improvements to build lab automation tools
    - improvements to the SNAP build environment
- build feature development: to improve efficiency in the overall SNAP build system
- build break root cause resolution: try to discover the root cause of frequently occurring build breaks
- code reviews: providing support of code review and being a source of advice for developers across the organization

Build development mainly to improve the build system and solve any issues during build. Office build team ensures the quality and the completion on time of the builds. In the end, ensure quality of the Office products.

Project Overview

My project is closely related to my daily work, as I mentioned earlier, with regard to the work of open bugs for build break and push for timely resolution of build issues. Simply to say, this work is related with build status feedback. Builders get build system feedback by using some of the tools and developers get feedback from builders, if their source code changes caused build breaks. The principle of this work is timely sent the right information to the right person.

My project is improving the tool which used in our team for open bug and sent build break feedback to developers in the following aspects:

- increase automation of open bug process which reduce workload for builders and the time to send feedback
- provide error message of build break to developers that can save their time for searching for these messages, and reduce the network access limitation to improve accessibility to these messages
- sent feedback to all related people, enable all related people get feedback as soon as possible

It is necessary to introduce the relevant background for better understanding of my project. In Chapter 1 Background, I will introduce the position of software build in the software development context. Explain the software build system used for Microsoft Office product, and further narrow the scope to discuss my project related area.

The Microsoft Office Build System is unique, and my project is to solve these specific problems in this particular environment, so, in Chapter 2 Identify Problems I will introduce our workflow for open bug and some relative tools. Based on my work experience and the interview with other team members, I identity six problems exists in the open bug process.

In Chapter 3 Finding Solutions I provide detailed solution for three of them, because of the priority of the problem and considering the limitation of the existing system. To solve the rest of the problem require redesign the tool. However, that could be the future work.

After having the solutions, I begin to realize them in the real work environment. I will introduce the implement process and core source codes in Chapter 4 Implementation. I used a lot of existing code to complete my functions, but I will not distinguish new code, because the point is the new features.

Next, I will present the results of by testing the new features in Chapter 5 Testing. And evaluate my contribution.

In Chapter 6 Conclusion, give conclusion on my thesis and an overall judgment on Office build system feedback mechanism. I will discuss some new ideas for the future research.

Chapter 1: Background

1.1 Introduction

This chapter will introduce many aspects to help to understand build system, and introduce a type of build system – continuous integration build system; because Microsoft Office Build System is the application of CI. Next, we discuss the feedback mechanism of build system, by looking at some existing methods and tools.

1.2 Building Software

Building Software is the process that translates the source code into binaries and assembly required files to create a standalone software application. This process is very different due to a wide range of differences in software application, such as development environment and size.

1.2.1 Defining a build

When we build software we call the compiler and assembler, convert high-level languages (source code) into machine language (binary) that computer hardware understand, and software build refers the result of this process. Look at figure 1.2.1-1 it gives an example of how source code converts to executable code.

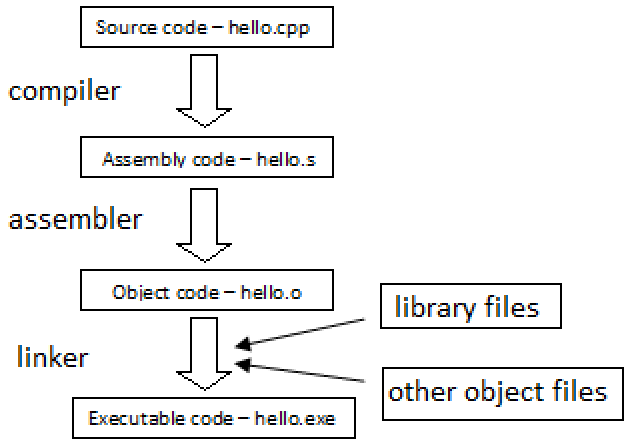


Figure 1.2.1-1 example of complication process

More precisely, there are two stages of the build process and each stage has a result. The two different results are builds and releases. The first stage is to compile the source code into object code and libraries and the result is build; the next stage package all object code and libraries the result is release.

For a simple project consists only a single file the build process is the compile of that file, the compiler is the build tools. The build process only take seconds, and you get the build result by the compiler, and most likely your IDE (Integrated Development Environment) [2] can do this for you, such as Eclipse for Java [3] or Visual Studio for C# [4]. After your project builds successfully you can perform unit test on it [5]. Most IDE support unit test as well.

However, imagine how you could build hundreds of projects together and need to manage project dependencies, and how you could run unit tests every time when source code changes. You need to configure the dependencies and automate the build process with unit tests.

1.2.2 Build automation

The software build process can be automated by using build tools and scripts and this act was called Build Automation. It is primary focus on automating the calls to the compilers and linker. As the build process grew more complex this becomes more essential. In order to achieve build automation, one essential element is Source Control System.

Source Control System

Source Code refers to files written in high-level languages need to be compiled, however Sources refers to all the files involved in building a product, not only the source code files, also include non-compile file like library file, documents and configuration files etc.

Source control system also known as version control system is simply just a database where sources stored in and with a front-end application with graphic user interface or just command prompt which is faster. It enables multi-user share and edit same file at same time. User 'check out' files from central repository to his development machine, make changes and 'check-in' changes back to repository, the first developer always success, it may have conflict when the second developer 'check in' his changes and it is his responsibility to take care with the merge. "The current state of the system referred to as the 'mainline', the copy on the developer's machine is called 'working copy'[CI]"

Each time a change checked in repository it is a new version of the sources. The system record and track all versions by identify changes using number and each revision with a timestamp and the person making the change. To keep simple, source control system keep the history of all changes.

Branching

"Branching is the Source Control System operation of creating an independent line of development for one or more files."[book1] Branching is copy some files to a separate place, the copies and original files can be modified as desired. Changes in one branch are not reflected in the other one, except explicit operations performed to merge changes from one branch into another, this merge operation is called Integration. Merge from original branch to the copy branch is Forward Integration**;** merge from the copy Branch to original branch is called Reverse Integration.

Integration Hell

The integration process can take very long time and becomes nightmare of development teams. "The consequence was that people were exhausted when they finally integrated, increasing the probability of errors. Also, the delay before integrating increased the probability of conflicts with the changes from other pairs" [6]

Integration is always an event for parallel development that multi-version of projects developing by multi-developer. And a good practice to solve this problem is to integrate changes continuously.

1.2.3 Continuous integration

"If you would like to run frequent integration builds so that it becomes a non-event on your project—including compilation, rebuilding your database, executing automated tests and inspections, deploying software, and receiving feedback—Continuous Integration (CI) [7]".

Figure 1.2.3-1 shows the steps in a CI scenario and the components of CI system.

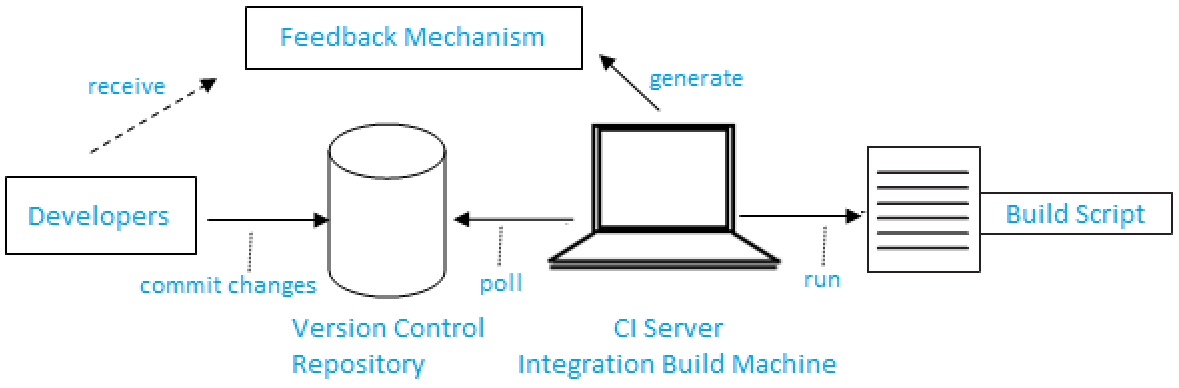


Figure 1.2.3-1 the components of a CI system

First, developer commits changes to the version control repository and CI Server is polling repository for the changes. Then CI Server runs Build Script to build the latest source code. The last, CI Server generates feedback of build status and sends to specified project members.

CI Server

The CI Server can start an Integration build by schedule on a regular frequency or manually trigger whenever a change is applied to the repository.

Build script

The basic part of build automation, use scripts because majority of builds are done at command line. It is scripts use to compile, test and deploy software.

Integration Build Machine

It’s a separate machine hosts the CI server, the responsibility is to integrate sources.

Feedback Mechanism

“One of the key purposes of CI is to produce feedback on an integration build, because you want to know as soon as possible if there was a problem with the latest build [8]”. By receiving feedback promptly, you can fix the problem quickly. Feedback is at the heart of CI build system.

For continuous integration we need continuous feedback that “sent the right information to the right people at the right time and in the right way [9]”.

- the right information: build status and testing results
- the right people: “everyone need to receive some type of feedback on the project, but not necessarily every item every time[9]“, it depends on the role in the team, such as
  - Project Manager need high-level information relates to time, cost, quality and scope
  - Architect needs the status of all builds because they look at the entire system;
  - Developers only need receive information related on their own tasks, the status on the code they just check-in;
  - Testers need information on the test results;
  - The right time: as soon as possible, real time is the best.
- The right way: there are main ways enable continuous feedback, such as e-mail, text message, visual devices and sound etc.

    - E-mail is the most common form of feedback, require e-mail server and client. The disadvantage is people don’t always have immediate access to e-mail;
    - Text message require a mobile phone and tool for sent message. It is good because it almost enable people receive and view feedback anytime, anyplace. But message will be very short;
    - Visual devices such as Ambient Orb, it can be customized to display lots of different color to show different build status. Requires a script capable of sending HTTP Get message, a build script and network connection. But cost, and cannot show detailed information;
    - Play different sound through your computer for different build status add a bit of fun, but easy to miss and short of information.
- Additional Feedback Devices
    - Browser plug-in: it enable your browser can indicate build status
    - Instant messenger: such as AIM, Yahoo and MSN.
    - RSS: Really Simple Syndication file updated for every build, it can provide as much information as e-mail does.
    - Widgets: various widgets can be created for windows and Mac.

No matter use what way for feedback, the purpose is for someone to take action as quickly as possible according the information.

1.2.4 Where we are in software engineering

Building software is a process belongs to Software Configuration Management (SCM). SCM is an integral part of the software development process cross all phases of the life cycle and is a series of measures to control and standardized the products and their process. It's goal is to record evolution of software products, to ensure that software developers have access to accurate product configuration at any phase of software lifecycle, by manage vast number of elements include source code, documentation, change requests, over the lifetime of a large software system, especially for distributed development projects.

Wayne Babich describes SCM as “the art of identifying, organizing, and controlling modifications to the software being built by a programming team. It maximizes productivity by minimizing mistakes [10]”. People always view coding as the real work for software development. “SCM can’t achieve star status, but it is essential to project success.

1.3 Microsoft SNAP Build

SNAP stands for Shiny New Automation Process. SNAP build system is the Microsoft own implementation of Continuous Integration System. The idea is the same to integrate frequently, but not for every change nor for each change. We integrate several changes together and these changes were approved by triage which a process to evaluate the changes. SNAP system is used by several teams in Microsoft such as Office and Windows. But each team has customized the system to build their products.

1.3.1 Microsoft Office Build System

It is based on SNAP build system and customized and developed by Office team. The build system is continuous improving. Here is only a brief introduction of some features of some elements, and how they work together. Figure 1.3.1-1 illustrates some components and how they communicate. All the components are used within Microsoft and some of them only for Office Build Team.

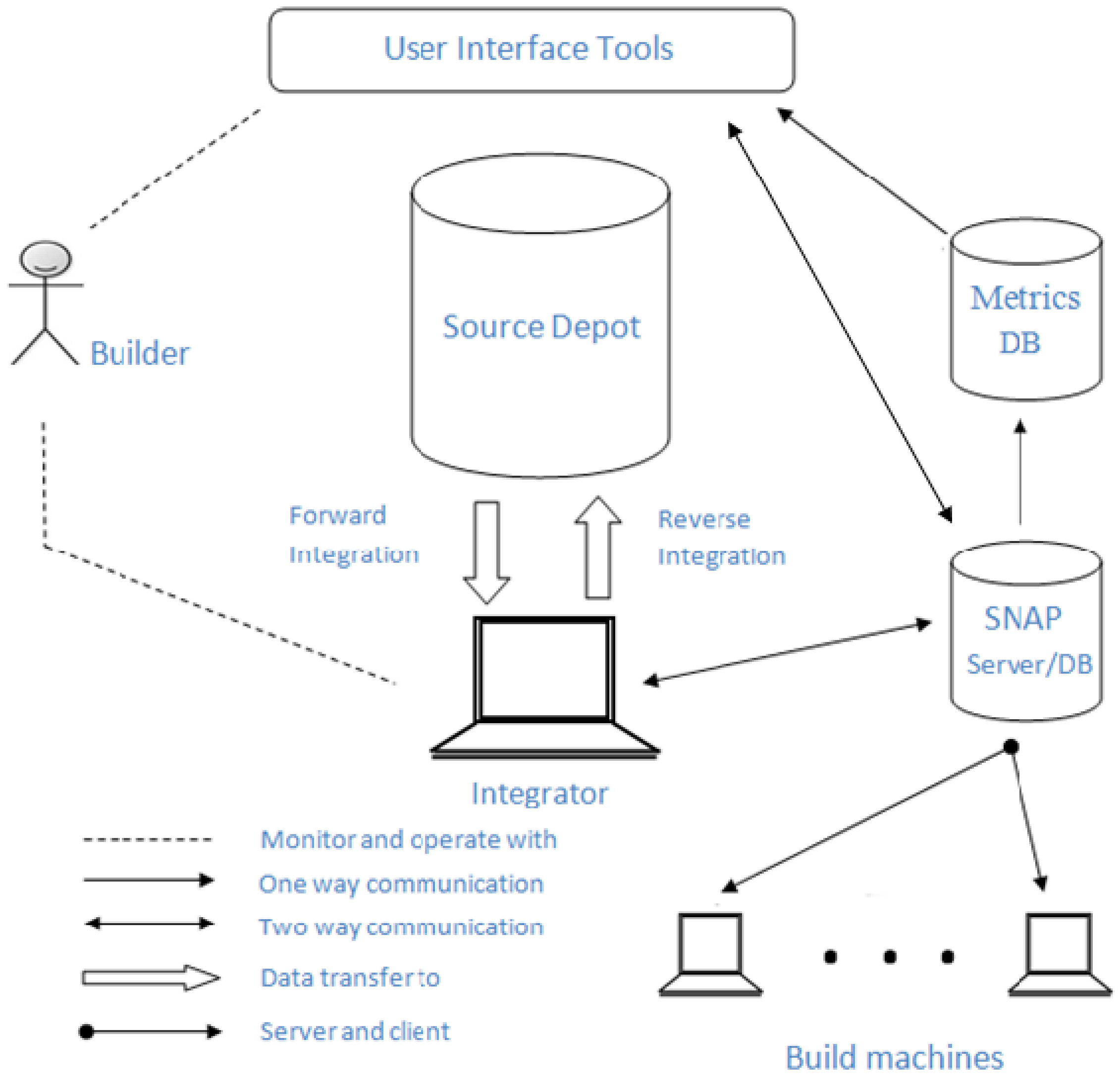


Figure 1.4-1

Source Depot: is the actual product name, which is our source version control system. It is command line based and provides file revision control, file sharing, branching and integration etc. It has the ability to effectively handle large projects with millions of files and large number of users.

Build Machines: they are the machines located at Build Lab, where the actual build tasks run at. SNAP daemon run on these machines and enable them to communicate with SNAP server.

SNAP: The Continuous Integration Server. The SNAP itself is a system, but here we can keep it simple as a server and database. The SNAP DB has all the build tasks and a set of machine daemons. And the SNAP server schedule tasks to build machines.

Metrics DB: a database keeps subset data of SNAP DB. The data are about the status of the build tasks, which can be viewed from build monitor tools.

To start a build, builder login into the Integrator to perform Forward Integration, copy all the sources going to build from Main Branch to Lab Branch and create a build job which includes thousands of projects or tasks. There have 100 build machines for each build, one build machine only run one single task at once, when a build machine finish one task it will request SNAP to get a new task. The task might be failed and could be a bug need developer to fix; if the task succeeds it will create some files and folders to update Lab Branch. All the build status information such as task's name, build machine's name, start running time etc. are go into Metrics database, and builder can monitor the build status through several User Interface Tools, they are all web based applications. When all the build tasks passed, the Lab Branch has the new version of files and performs Reverse Integration to update Source Depot.

1.3.2 Microsoft Office Build Process

MS Office has two types of build Daily Build and Weekly Build, they both schedule based and run twice a week. Daily Build usually takes 48 hours without releases. It is used for testing in build lab quickly identify problems.

Let's look at the build process of a Weekly Build. Weekly build, as the meaning of the name, is a whole build process will takes about 5 days with Build Verification Test and Releases. With the Shared Operations of Ireland Build Team and Redmond Build Team, allows for full coverage for at least 16 hours per day.

Build Verification Tests (BVTs): "BVTs are automated suites of tests designed to validate the integrity of each new build and the basic functionality of the build before it is released for more in-depth testing [book1]". It is a test process also a phase during build process. We have a dedicate team to perform BVTs, builders notice BVT team to start running BVT and get the test results from them, if the results is not good enough we have to fix problems and do another BVT, usually there have two or three BVTs for a Weekly Build.

During the build process each day has a milestone, that need to be reached ensure the build completes on time.

Day 1

- Build starts at 8am GMT
- Task is loaded and projects start building
- Build is monitored, breaks are logged and investigated
- Build break fixes are picked up as thy arrive

Day 2

- First BVT drop is triggered
- BVT results available by the start of Redmond's day

Day 3

- BVT fixes from the first drop are picked up, re-builds start
- Second BVT drop is triggered
- Logging of breaks and picking up of fixes

Day 4

- Results of the second BVT determine if wrap up can begin or need another BVT drop
- Once starts wrap up the release projects start to build

Day 5

- Final release projects are built
- Releases begin to work out

Figure 1.3.1-1 describes the process of a Weekly Build

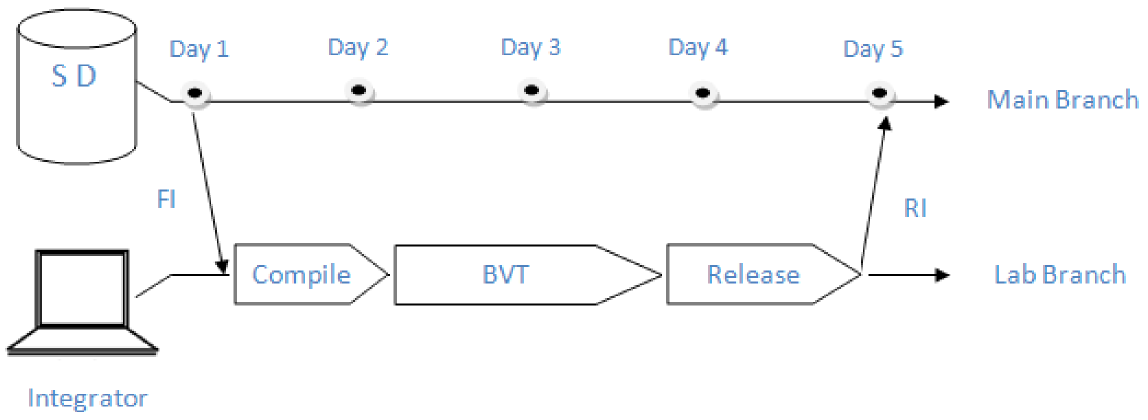


Figure 1.3.2-1 Weekly Build Process

There have three phases, during compile phase convert source code files into binary files; during BVT, BVT team test the executable build; during release phase, put all files into a package which can burn into CD and install.

1.3.3 Bug Tracking

Product Studio (PS), bug tracking system widely used internally at Microsoft used Microsoft SQL Server database for storing bug information and with front-end user interface. PS is the bugs' depository store all relevant information from open new bug until the bug be closed.

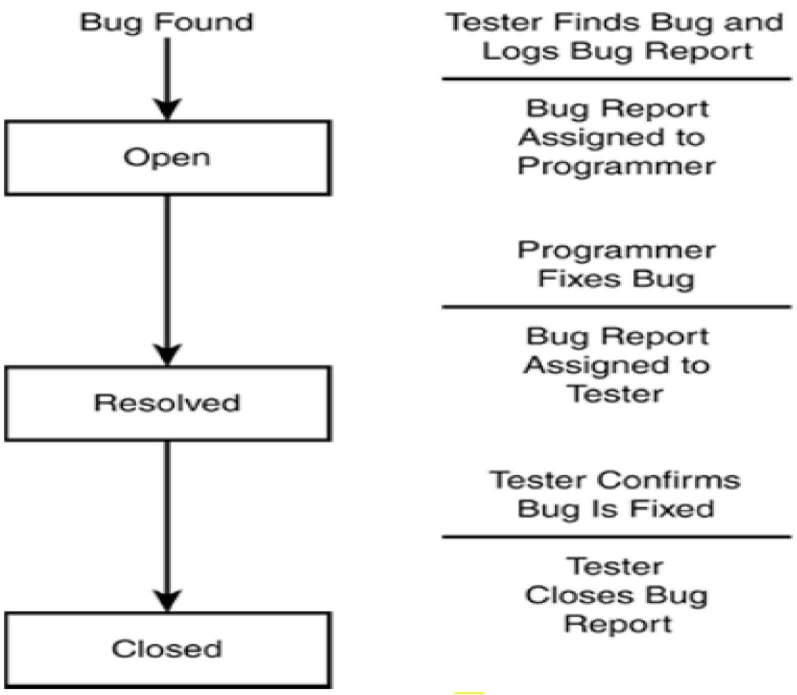
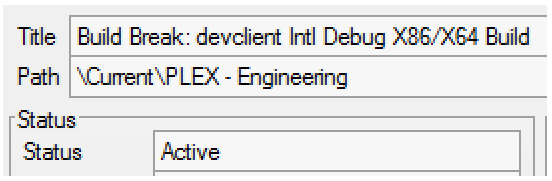

Bug Found
Open
Resolved
Closed
Tester Finds Bug and Logs Bug Report
Bug Report Assigned to Programmer
Programmer Fixes Bug
Bug Report Assigned to Tester
Tester Confirms Bug Is Fixed
Tester Closes Bug Report

Figure 1.3.3-1 Bug's Lifecycle [Book2]

Figure 1.3.3-1 shows fundamental bug's lifecycle. A bug has three statuses either Open, Resolved or Closed. "When a bug is first found by a software tester, a report is logged and assigned to a programmer to be fixed, the bug is open. Once the programmer fixes the code, he assigns the report back to the tester, the bug is resolved. The tester verification the fixes to confirm that the bug is fixed, then close the report [Book2]". If it is not fixed, tester will reopen the same bug and assign to developer.

Build break, the bug will happen when compiler or linker outputs an error caused by the source code it was run against. A basic rule for build break is that "who broke it, who fixes it". It is developer's responsibility to fix the build break not build team, "for the build team, building without errors and having the ability to track down the person who broke the build is the most important thing [book1]".

It's open by builder, and created a new bug in PS, the status is "Active", show as Figure 1.6-2.



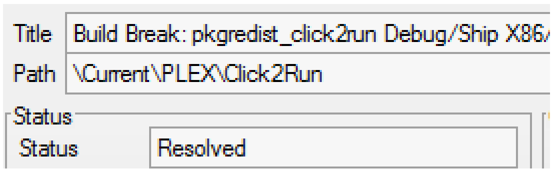
Title	Build Break: devclient Intl Debug X86/X64 Build
Path	\Current\PLEX - Engineering

Status

Status	Active

Figure 1.3.3-2 "Active" status

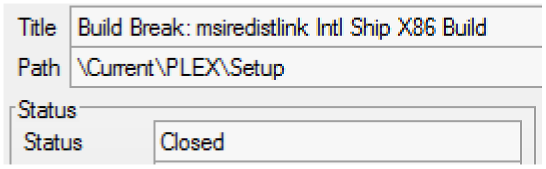
When the bug fixed by developer the status change to “Resolved” show as Figure 1.3.3-3.



Title	Build Break: pkgredist_click2run Debug/Ship X86/
Path	\Current\PLEX\Click2Run
Status	
Status	Resolved

Figure 1.3.3-3 “Resolved” status

Builder applies the fix for the bug and rebuilds the failed project, if the project builds successfully then the bug is “Closed”. Show as Figure 1.3.3-4.



Title	Build Break: msiredistlink Intl Ship X86 Build
Path	\Current\PLEX\Setup
Status	
Status	Closed

Figure 1.3.3-4 “Closed” status

1.3.4 Office build system feedback mechanism

There have several tools provide different information for different users. Builders directly get feedback from build system, and update feedback information by some tools for other users not in build team; also these tools directly get feedback from build system.

BuildUI and SnapUI

Web tools for builder to monitor and control build process. With BuildUI we can track the current status of the projects as the wait, get built or fail. With SNAPUI we can monitor the use of build machines for each active build and task queue status. There do not have the ‘builder’ role in some company, that’s because the developer or manager is the ‘builder’ himself. In Microsoft we have this specific role for continuous monitor the build process and directly get feedback from the build system.

Build Break Management Tool

Especially for report build breaks to developers. It is the builder’s responsibility to open bugs and sent build break e-mail by using this tool. Build break information includes check-ins information and system log folder path. The e-mail should send to the developers who make check-ins until last build and stake-holders for the broken project.

ReleaseUI

ReleaseUI is similar to BuildUI ReleaseUI tracks the releases and allows the builder to see which releases have completed and are available for testers or which ones did not get created correctly. ReleaseUI is a private tool like SnapUI and BuildUI so it is only accessible by the build team.

OfficePipe

OfficePipe is the customer facing build monitoring tool. OfficePipe is where everyone else in the build organization gets information about the current running builds. OfficePipe display information such as:

- What stage the build is currently at
- The results of the BVT drops of this build
- Comments that the build lab would like to share with the rest of the org
- All of the bugs that have been logged for this build
- Which bugs have been resolved and which are still active
- Statistics about which project shave been failing most in recent build etc.

Non build team members do not have access to SnapUI or BuildUI so as a result require OfficePipe in order to get all the information that they require about a specific build.

1.4 Analysis Other CI Tools

There are several visible continuous integration tools, have their own way to give build status feedback, mostly they are web based application. Some tools only simply provide build status indicate by colors such as GreenScreen which is designed as a dynamic Big Visible Chart [GS].

Some tools are designed to use in an enterprise environment such as Apache's Continuum [Continuum]. Cruise Control [CC], they are some of the best tools provide visual dashboards and enforce the process of continuous integration. They all offer a variety of choices for the feedback method, such as e-mail, MSN, SMS, IRC (Internet Relay Chat) [IRC] and Jabber [Jabber] belong to Cisco, provide enterprise real-time communicating.

They all use XML for configuration of feedback method, the information and the receiver etc. The Figure 1.4-1 is the break notification configuration file of Continuum for using e-mail.

```
<ciManagement>
  <system>continuum</system>
  <url>http://127.0.0.1:8080/continuum</url>
  <notifiers>
    <notifier>
      <type>mail</type>
      <sendOnError>true</sendOnError>
      <sendOnFailure>true</sendOnFailure>
      <sendOnSuccess>false</sendOnSuccess>
      <sendOnWarning>false</sendOnWarning>
      <configuration>
        <address>continuum@127.0.0.1</address>
      </configuration>
    </notifier>
  </notifiers>
</ciManagement>
```

Figure 1.4-1 Continuum notification configuration

And Figure 1.4-2 is the Cruise Control notification configuration for using SMS.

```
<publishers>
  <email mailhost="smtp.mydomain.com"
    returnaddress=buildstatus@mydomain.com
    defaultsuffix=@mydomain.com
    returnname="Project Build Status"
    spamwhilebroken="false"
    buildresultsurl="SMS">
    <failure address="7035551212@mobilephone-emailaddress.com"
       reportWhenFixed="true"/>
  </email>
</publishers>
```

Figure 1.4-2 Cruise Control notification configuration

Advantages

- XML is seen as a universal, open, readable representation for data exchange. Using XML creates extendibility and flexibility.
- Provide multiple feedback methods provide better availability.

Disadvantages

- Use internet communication protocol may face security problem
- Only simple feedback information

1.5 Summary

By introducing CI system and Office build system, we can see similarities and differences. Each software build system will be not all the same, the feedback mechanism of Office build system has a great difference with others, it provides more information for more participants.

Chapter 2: Identify Problems

2.1 Introduction

Analysis the communication process between builder and developer, according the CI feedback mechanism principle "timely sent the right information to the right person" identify problems exist in current build break notification process.

2.2 Feedback Mechanism

The following figure illustrates the feedback mechanism for a build break.

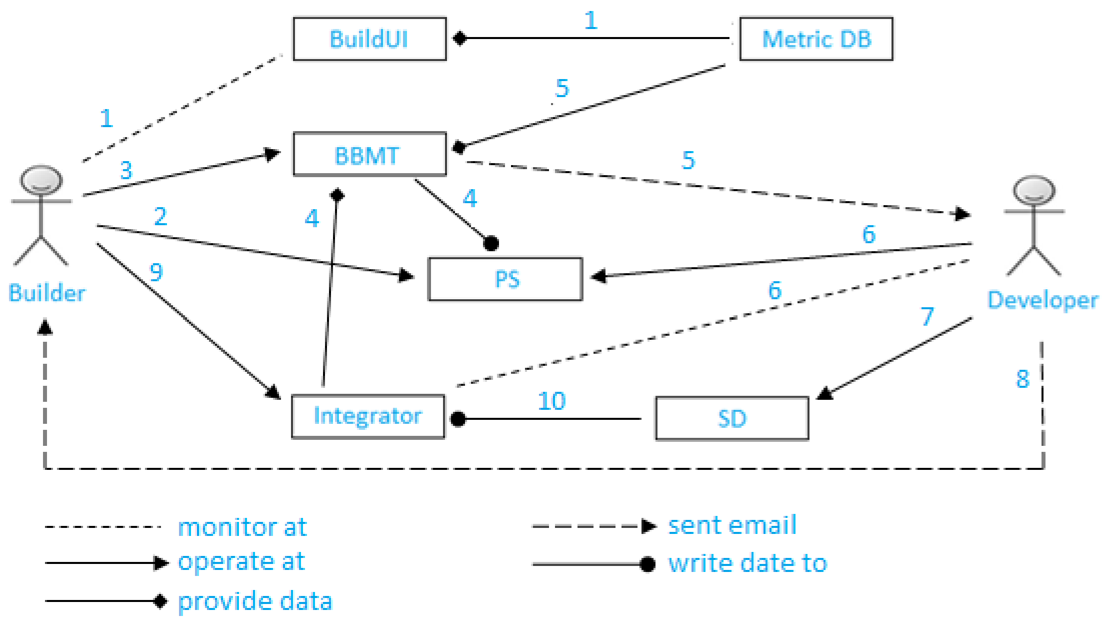


Figure 2.1-1 Feedback mechanism

The number on the Figure represents the steps, steps 1 – 5 is the process for generate feedback to developer; steps 6 – 10 is the process for fix bug and feedback to builder. Let's follow the order:

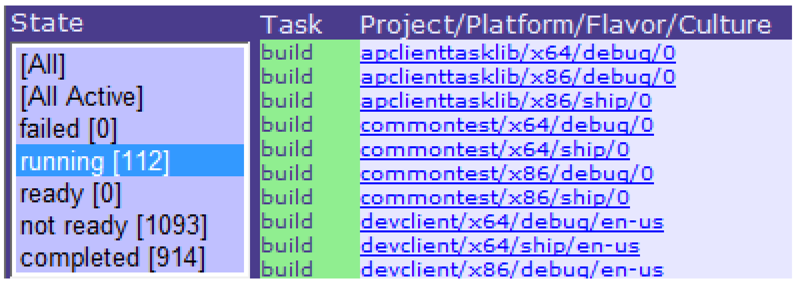
- Step 1 – When a task failed BuildUI get data from Metric DB and monitored by Builder

- Step 2 – When Builder not sure whether the failed task opened or not, he should check PS (product studio) where all bugs stored to make sure the status of the break, especially when there already have many failed tasks before Builder start monitor, because some bugs maybe already opened by other Builder.
- Step 3 – Builder open BBMT and fill in all required information such as build number, failed project name etc. manually.
- Step 4 – BBMT automatically get log folder path from Integrator and open bug in PS
- Step 5 – BBMT automatically get check-in data from Metric DB and generate Build Break Email sent to developers who make the check-ins and the owner of the project. Email includes links of the log folder and check-ins information.
- Step 6 – When developer received the Build Break Email he check more details by investigate the bug in PS and logs in Integrator.
- Step 7 – Developer check-in fixes to Source Depot
- Step 8 – Developer reply Build Break Email to notify Builder that the bug has been fixed.
- Step 9 – Builder login to Integrator to apply fix to the bug
- Step 10 – integrate changes from SD to Integrator

2.2.1 Feedback for developer

Step 1: Monitor BuildUI

Builder monitor build task status use BuildUI one of the web based user interface tools, according Build ID it represent general information such as build type, builder alias and Integrator name.



State

- [All]
- [All Active]
- failed [0]
- running [112]
- ready [0]
- not ready [1093]
- completed [914]

Task	Project/Platform/Flavor/Culture
build	apclienttasklib/x64/debug/0
build	apclienttasklib/x86/debug/0
build	apclienttasklib/x86/ship/0
build	commontest/x64/debug/0
build	commontest/x64/ship/0
build	commontest/x86/debug/0
build	commontest/x86/ship/0
build	devclient/x64/debug/en-us
build	devclient/x64/ship/en-us
build	devclient/x86/debug/en-us

Figure 2.2.1-1 BuildUI Task Statues and Running Tasks

As show in the following figure during the build process each task will failing into one of the statuses: failed, running, ready, not ready or completed.

Here is a UML state chart to illustrate the transition of the statuses

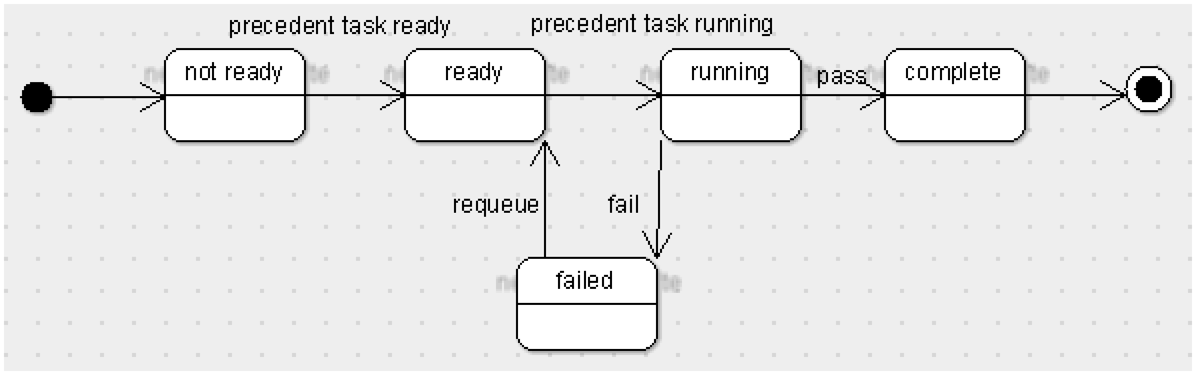
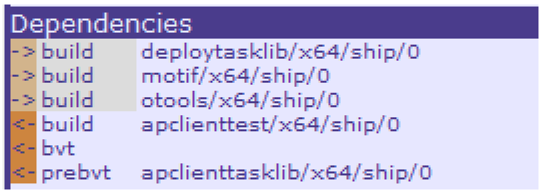


Figure 2.2.1-2 Task State Chart

- Not ready – if task has any unfulfilled dependencies which means its precedent tasks either "not ready" or "ready". It stays in this state.
- Ready – if task has fulfilled dependencies which means its precedent tasks either "running" or "complete".
- Running – the task for a project is executing
- Failed – the task failed to run, after re-queue it will become to "ready" waiting for next "run"
- Completed – the task has run successful

For each task we can check the dependencies, '->' represent the dependency direction, read as 'depend on', Figure 2.2.1-3 shows dependencies of a running task, all the running task's precedent are 'completed' and all descendant are 'ready'.



Dependencies		
->	build	deploytasklib/x64/ship/0
->	build	motif/x64/ship/0
->	build	otools/x64/ship/0
<-	build	apclienttest/x64/ship/0
<-	bvt	
<-	prebvt	apclienttasklib/x64/ship/0

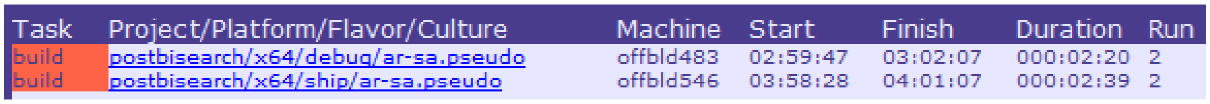
Figure 2.2.1-3 Task Dependencies

Figure 2.2.1-1 shows some running task, for each task there has project name, platform, flavour and culture

- Project - the name of components of Office like word, excel etc. there have hundreds project in Office

- Platform - x86 and x64 for 32bit and 64bit platform
- Flavour - debug and ship, two different version of executable. Debug version includes debugging information. Ship version for distributed to other people which is smaller and run faster
- Culture - for different language such en-us, if the value is ‘0’ means the project is not language specified

The following figure shot show there have two failed build tasks which highlighted in red.



Task	Project/Platform/Flavor/Culture	Machine	Start	Finish	Duration	Run
build	postbisearch/x64/debug/ar-sa.pseudo	offbld483	02:59:47	03:02:07	000:02:20	2
build	postbisearch/x64/ship/ar-sa.pseudo	offbld546	03:58:28	04:01:07	000:02:39	2

Figure 2.2.1-4 Failed Task

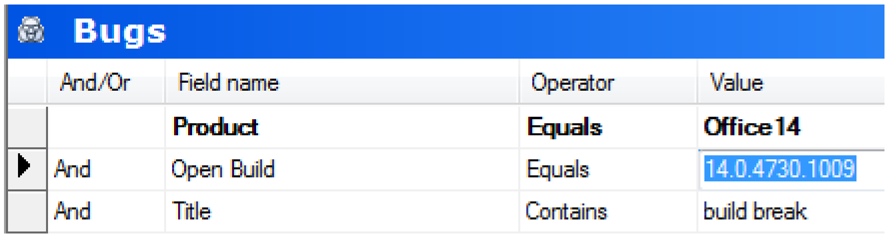
- Machines - the build machine which was running the task
- Run - the times this task has been executed, if the first time failed the task will re-queue automatically and run again to reduce other facts may lead to failure, such as build machine and network problem.

If there have task failed twice, it was considered to be a new bug. When Builder not sure the state of the break should check before open, otherwise reopen the same bug will confuse developers. If sure about the break status can skip Step 2.

Step 2: Check Break State

There have two way to check a build break’s state, either check email or PS.

- Check from email: search within Build Break Email by failed project name. This way is quicker than check from PS, and can follow the track of the email get more information.
- Check from PS: show as following screenshot, as PS is especially bug tracking tool, it provide more powerful functions, you can write your own query to search bugs with complex constraints.



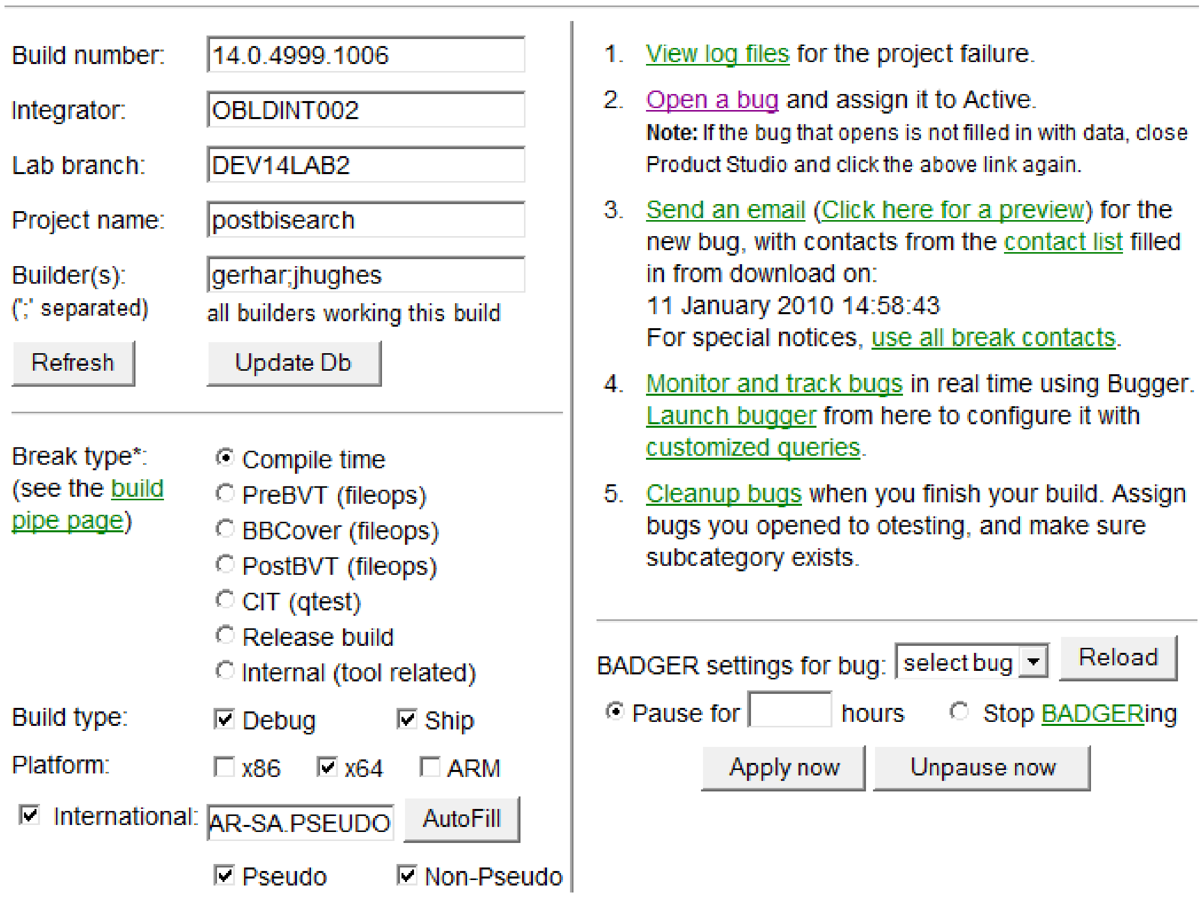
Bugs

And/Or	Field name	Operator	Value
	Product	**Equals**	**Office14**
And	Open Build	Equals	14.0.4730.1009
And	Title	Contains	build break

Figure 2.2.1-5 Query Bugs in Product Studio

Step 3: Input break info

Open Build Break Management Tool, and manually fill in required information on the left hand side, all these information we can get from BuildUI, there have several different break type, different break type according the different task, in this case the failed task is build task so it is compile time build break, and the reason why to distinguish break types is that different type of tasks create their own folder and logs, this help to locate the log files. there have two build task failed but for the same project so only need to log one bug, just specify the build type, platform and international which is the culture.



Build number:	14.0.4999.1006
Integrator:	OBLDINT002
Lab branch:	DEV14LAB2
Project name:	postbisearch
Builder(s): (';' separated)	gerhar;jhughes — all builders working this build

Break type*: (see the build pipe page): [✓ Compile time] PreBVT (fileops) BBCover (fileops) PostBVT (fileops) CIT (qtest) Release build Internal (tool related)

Build type: [✓ Debug] [✓ Ship]

Platform: x86 [✓ x64] ARM

[✓ International:] AR-SA.PSEUDO

[✓ Pseudo] [✓ Non-Pseudo]

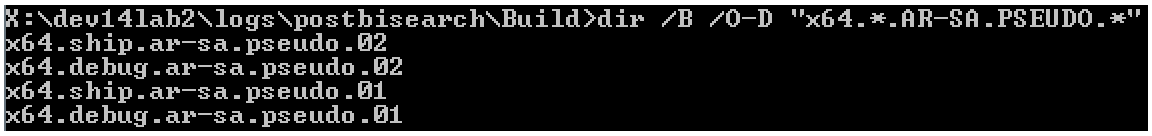
1. View log files for the project failure.
2. Open a bug and assign it to Active.
   Note: If the bug that opens is not filled in with data, close Product Studio and click the above link again.
3. Send an email (Click here for a preview) for the new bug, with contacts from the contact list filled in from download on:
   11 January 2010 14:58:43
   For special notices, use all break contacts.
4. Monitor and track bugs in real time using Bugger. Launch bugger from here to configure it with customized queries.
5. Cleanup bugs when you finish your build. Assign bugs you opened to otesting, and make sure subcategory exists.

BADGER settings for bug: select bug

[✓ Pause for] hours   Stop BADGERing

Figure 2.2.1-6 BBMT

When click the ‘view log files’ link on the right side of BBMT, it will show the logs path in a command window, the purpose for doing this is double check whether the information fill in BBMT before is correct, if any input incorrect no logs will presented. Following is an example of the two failed task which show in Screenshot 2.2-4, each task run twice so there have four logs totally.

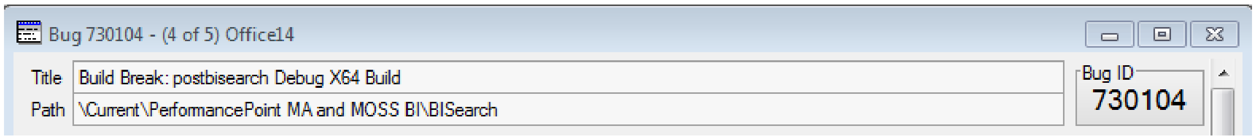


```
X:\dev14lab2\logs\postbisearch\Build>dir /B /O-D "x64.*.AR-SA.PSEUDO.*"
x64.ship.ar-sa.pseudo.02
x64.debug.ar-sa.pseudo.02
x64.ship.ar-sa.pseudo.01
x64.debug.ar-sa.pseudo.01
```

Figure 2.2.1-7 Log Folders

Step 4: Open a Bug

When click ‘open a bug’ a new bug will create with a bug id at Product Studio and there shows the bug title and bug ID in the following figure.



Bug 730104 - (4 of 5) Office14

Title: Build Break: postbisearch Debug X64 Build

Path: \Current\PerformancePoint MA and MOSS BI\BISearch

Bug ID: 730104

Figure 2.2.1-8 Bug Title and Bug ID in PS

Step 5: Send Build Break Email

Click ‘send an email’ and provide the bug id which you get from Product Studio, it will send email automatically to correspond developer.

Build break email is the major communication method to notify developers that there have breaks need to be fixed, and the email provides logs and check-in lists to help developer to investigate, and developer can reply the email to discuss solutions.

The best-case scenario for build break email is only sent the check-ins which caused the break and only sent to the correspond developers, so the developers who broke the build can fix it quickly; otherwise sent the email to all related people without indicate which check-in cause the break will cause too much spam and wasting a lot of time to find out the right developer.

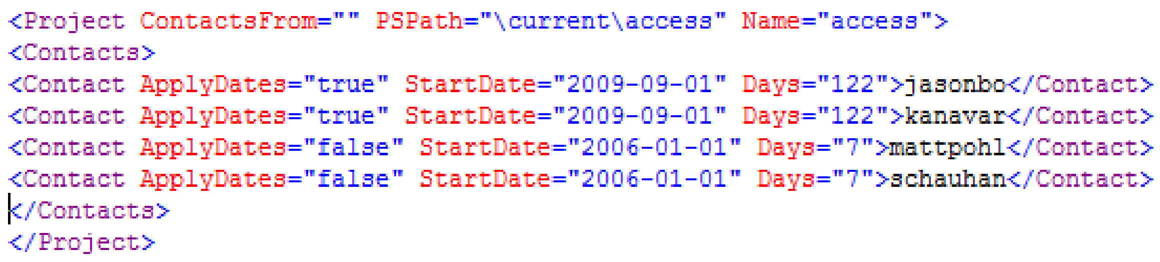
BBMT can automatically sent build break email. The recipients include two groups of people:

- To – developers who directly check in the project since the last build
- Cc – developers who own the project

If there is no check in developers then the ‘To’ group is the builder who log the break.

BBMT get check in users by query Metrics database provide with build number and project name to get all check in developers who check in the specific project between the current build and the latest build. Because after each build checkpoint all check-ins were build and reverse integrated, there is no breaks exists in the build, if a broken project can't get fix then it will be included in the build.

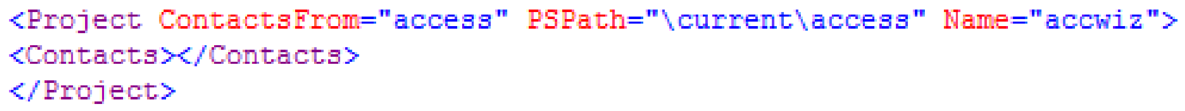
BBMT get project contacts by look into an XML file, this file include all the projects and its owner's, following is an example for project 'access'.



```
<Project ContactsFrom="" PSPath="\current\access" Name="access">
<Contacts>
<Contact ApplyDates="true" StartDate="2009-09-01" Days="122">jasonbo</Contact>
<Contact ApplyDates="true" StartDate="2009-09-01" Days="122">kanavar</Contact>
<Contact ApplyDates="false" StartDate="2006-01-01" Days="7">mattpohl</Contact>
<Contact ApplyDates="false" StartDate="2006-01-01" Days="7">schauhan</Contact>
</Contacts>
</Project>
```

It include four 'Contact' elements, but BBMT will just sent email to the first two contacts, because the first two people is the current owner from 2009-09-01, the last two people are the previous owner.

The attributes 'ContactsFrom' means get contacts from which project. In the following example that project 'accwiz' redirected to the contacts of 'access', and its 'Contacts' element is empty.



```
<Project ContactsFrom="access" PSPath="\current\access" Name="accwiz">
<Contacts></Contacts>
</Project>
```

The value of 'PSPath' indicate which team introduce the change, project 'access' and 'accwiz' belong to the same team, so they have the same value for 'PSpath'.

BBMT automatically sent Build Break Email to the contacts. The email includes logs path and check-ins information.

2.3 Problems

After introduce the details about sent build break email, some problems will be identified in each steps.

2.3.1 In Step 1 and 2

Problem 1: Monitor multi-build

It is inconvenience to monitor multi-build at the same time. When Builder monitor two builds, he need to open two BuildUI pages and switching between them and when both build have breaks he need to open two BBMT as well because one BBMT for one build can reduce manual input especially when the build just start there will have many breaks to open. Otherwise, just one BBMT for all builds builder need continue modify the inputs.

In the future the BuidUI and BBMT should be combined, probably add BBMT functions in BuildUI. Otherwise, require redesign BBMT user interface can indicates the failed break such as BuildUI did.

Problem 2: Check breaks state

There is no difference of a build break whether it is has been open or not, they are all in the 'failed' state mixed together in the ranks in BuildUI, builder need spend time to confirm, and sometimes made mistake that reopen bug and caused trouble to developers.

The idea to solve this problem is simple, just quey PS with the build ID, project name and which the bug status is 'Active'. There have SDK for Product Studio, but only available in .NET platform, which BBMT currently did not use.

2.3.2 In Step 3

Problem 3: Manually input

In order to input the information like Integrator and project name into BBMT, builder need type them manually or copy from BuildUI.

To reduce the manually input should let builder choose build ID from a drop down list which lists the running build and that's it, all the relative information should get from database automatically. When open a bug, use the same way, chooses the failed project name from a list.

2.3.3 In Step 4 and 5

Problem 4: Indirectly check logs

The build break email only provide the link of logs folders, the developer still need to get into the folder and open the error log files to investigate the problem. These log files are located in Integrator to open it need more time than open it in local machine, and another fact is that sometime developer not on site but he view the email on any computer or Mobile phone. When he received a build break email, he cannot access intranet to view log files.

Problem 5: Incomplete contacts

Another problem is that sometime the email contacts will not include the developer who introduced the problem when there is no check-in for this build since last build or none of the check-ins cause problem. So, in order to find out the problem developers need to investigate log files find out the problem and forward email to the right person.

2.3.4 The build lab issue

Problem 6: Phony break

Sometimes the build break is caused by build lab issue such as lose network connection or disk out of space. This kind of break is not real bug, should not be open in PS and assign to developer. But all build breaks look like the same, only if you check the error log file you can know the reason, while this is hard for builder to do, not because of it is difficult to check an error file, it is very boring to check for every breaks and waste a lot of time.

2.5 Summary

By analysis the build break notification method used for Office build system, the outcome is identified six problems exists in each steps. And discuss solutions for some problems, because these problems are not provided with detailed solution in the following chapter.

Chapter 3 Finding Solution

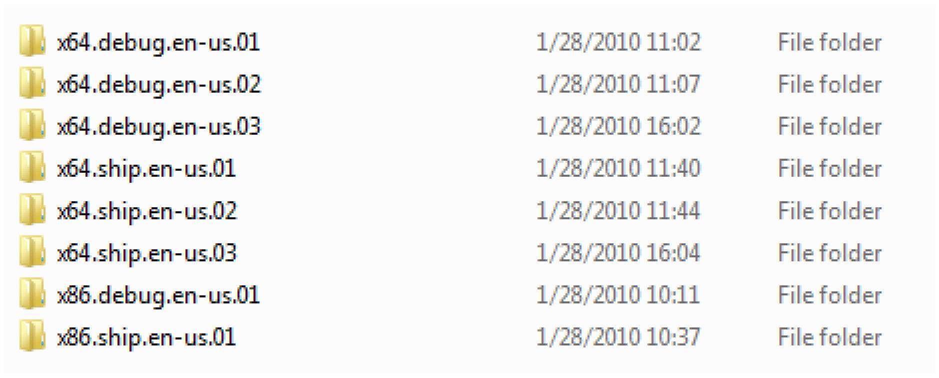
3.1 Introduction

In this chapter I will provide detail solution for three problems which are identified in Chapter 2. Consider the time limits the rest of problems will not be solved in this thesis, I will consider them as my future work which I will talk more in Chapter 6.

3.2 Solution of Indirectly Check Logs

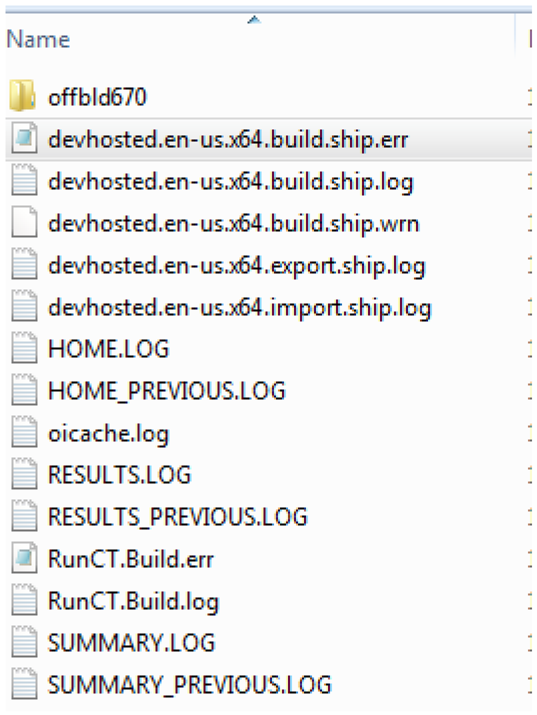
In order to help developer investigate the without the limitation of intranet accessibility and reduce the frequency to open logs from remote Integrator, the first solution straight in my mind is copy the error log sent as attachments with e-mail, because considering the difficulty to get the actual error message from an error log.

This is root log folder for 'devhosted' build task



x64.debug.en-us.01	1/28/2010 11:02	File folder
x64.debug.en-us.02	1/28/2010 11:07	File folder
x64.debug.en-us.03	1/28/2010 16:02	File folder
x64.ship.en-us.01	1/28/2010 11:40	File folder
x64.ship.en-us.02	1/28/2010 11:44	File folder
x64.ship.en-us.03	1/28/2010 16:04	File folder
x86.debug.en-us.01	1/28/2010 10:11	File folder
x86.ship.en-us.01	1/28/2010 10:37	File folder

And when build break happened we usually get into the latest folder to check error logs, to the latest folder could by the create time or just by the name itself as the name structure is platform.flavour.culture.number, with the biggest number is always the latest one.



```
Name
offbld670
devhosted.en-us.x64.build.ship.err
devhosted.en-us.x64.build.ship.log
devhosted.en-us.x64.build.ship.wrn
devhosted.en-us.x64.export.ship.log
devhosted.en-us.x64.import.ship.log
HOME.LOG
HOME_PREVIOUS.LOG
oicache.log
RESULTS.LOG
RESULTS_PREVIOUS.LOG
RunCT.Build.err
RunCT.Build.log
SUMMARY.LOG
SUMMARY_PREVIOUS.LOG
```

There have several different types of error logs, in this case there are three '.err' files

.build.err

This file is created automatically by a SNAP build scripts. When a project is building it creates and adds to a large log file (RunCT.log), if it breaks the errors are added to the large log file. The *build*.err file is created automatically. It contains just the actual errors from the large files and none of the other info. This is the file we look at when investigating a break.

.export.err

Every project has an export file, this contains a list of files that are created by building this particular project. It has to do with the dependency chain between projects. If a project "A", produces fileX, fileY and fileZ, then these 3 files need to be listed in Project A's export file. If they are not all listed there will be an export break and the *export*.err file is created.

RunCT.*.err

The RunCT.err file is very short and kind of useless, it is created when a project breaks but doesn't contain and information about the break.

The *.build*.err is usually developer looked into, I ask a developer in our team which kind of error is useful in this file, the answer is quite simple that" the error messages before the first empty line". Because of the structure of the error file, to get the error message is much easier than I expected.

3.3 Solution of phony break

The break caused by Lab Issue is not often, as well builder do not check the log often because not worth, so it has great potential to report this kind of break to developer that's annoying.

Based on the solution of previous this problem is also easy to fix, the solution is the same as the first one, only different here is search for certain sentences, such as "Source Depot lost connection" which is network issue, and "cannot copy file to integrator" which is because integrator disk out of space.

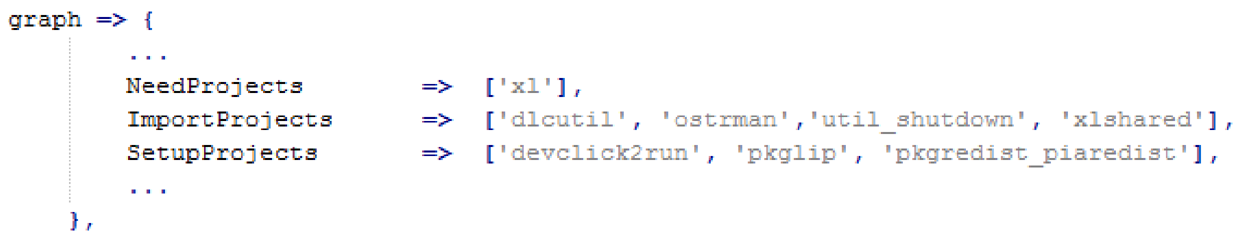
3.4 Solution of incomplete contacts

Some of the details about get contacts has mentioned earlier in section 2.2.5 Step 5. The contacts include the owners and check-in developers, but sometimes check-ins for one project could cause another project failed, because of source code dependencies, so the contacts should include the check-in developer for dependent projects.

Get Project Dependencies

All projects dependency information were kept in a file 'projects.pm', this is Perl Module file. A Perl Module is a self-contained piece of Perl code can be used by a Perl program or by other Perl Module. It is equivalent to the class concept in Object-oriented world.

'projects.pm' is project information database all information is kept in an associative array. The associative array is a list of key-value pairs. Following is an example for project 'graph', project name is a key and its value include three types of dependencies they are 'NeedProjects', 'ImprotProjects' and 'SetupProjects', I add the '…' which is not in the file to represent there have many other values. Each type of dependencies also is a key and its value were kept in an ordinary list array '[ ]' the values are other projects' name. All the dependencies are organized at project level.



```
graph => {
        ...
        NeedProjects        =>  ['xl'],
        ImportProjects      =>  ['dlcutil', 'ostrman','util_shutdown', 'xlshared'],
        SetupProjects       =>  ['devclick2run', 'pkglip', 'pkgredist_piaredist'],
        ...
    },
```

- SetupProjects – this is the mapping of setup projects and their platforms which consume files from this project and platform. Means to setup project ‘devclick2run’ require files from project ‘graph’. Means ‘devclick2run’ depend on ‘graph’
- ImportProjects – the list of projects that this projects imports from. To build ‘graph’ will import files from ‘dlcutil’, means ‘graph’ depend on ‘dlcutil’.
- NeedProjects – keys of other projects whose sources are required to build this project. This kind of dependency will not affect the build sequence, even if project X and projects Y ‘need’ each other. To build project ‘graph’ SNAP server will copy ‘xl’ source code and build them together, so the source code of ‘xl’ could cause ‘graph’ failure.

Both ‘SetupProjects’ and ‘ImportProjects’ are dependencies on binary files, which will affect build sequence. ‘NeedProjects’ is dependencies on source code, the changes made for the source code of ‘xl’ also cause ‘graph’ build break, even there is no changes made directly for ‘graph’, so when the ‘graph’ break BBMT need get its ‘NeedProjects’ which is ‘xl’

BBMT is a HTML application, to create an HTML application just write an HTML page and save it as ’.hta’. HTAs not only support everything a web page does, also have functionality control over user interface design and access to the client system, run as trusted applications, run like any executable. BBMT is written by Jscript, which is Microsoft version of JavaScript which is customized for Internet Explore.

However, BBMT cannot use ‘projects.pm’ directly. There have many batch file in the Source Depot to perform a variety of tasks and BBMT usually call these procedures for some functions. So, a batch file in Perl can fetch ‘NeedProjects’ from ‘projects.pm’.

‘projects.pm’ file is very large and stored in different locations with BBMT. Access ‘projects.pm’ for every break is not efficient. And ‘projects.pm’ does change frequently by developers, we cannot copy it to the same location of BBMT because we do not know when it will change and keep it updated. So, only get necessary data from ‘projects.pm’ and saved to a file in the same location with BBMT.

Get dependent projects check-in developer

For each ‘NeedProjects’ for the failed project, do the same as the failed project get all check-in developers by query Metric database.

User define dependencies

There have some already know build breaks ‘pattern’, such as project X break probably because of project Y. In addition to the ‘NeedProjects’ other dependencies and some facts also could cause breaks. So, we can define these dependencies in a file such as XML or text file. BBMT can use this file when build break happen, do the as for the file which keep the ‘NeedProjects’ data.

3.5 Summary

Three problems had found detailed solutions. Even the first two solutions are similar but solve the tow different problems, one problem for builder and another problem for developer. After finding solutions the implementation is straightforward.

Chapter 4: Implementation

4.1 Introduction

In this chapter I will implement the three solutions discussed in Chapter 3 and solved the problems. The implementation is based on existing source code of BBMT and the major languages are Perl and Jscript.

4.2 Add Error Message

Add error message in the build break email to solve the developer indirect check log problem. According the solution do following steps.

Get needed error logs

The command "dir * > a.txt /B /O-D" could list all the files in the current folder and save into a.txt file. The "/B /O_D" can only keep the file name and list by reverse order. So, the line 4 can save the project log folder in the temp.txt

```
// Get the log file contents

1.    oTextStreamOut.WriteLine("@echo
_____________________________________________________________");

2.    oTextStreamOut.WriteLine("@echo Log file directories for the bug,
sorted from newest to oldest:");

3.    oTextStreamOut.WriteLine("dir /B /O-D " + getLogFileMask() + "");

4.    oTextStreamOut.WriteLine("dir /B /O-D " + getLogFileMask() +
">c:\\office\\dev14\\temp.txt");

5.    oTextStreamOut.WriteLine("@echo test:" +  getLogFileMask());

6.    oTextStreamOut.WriteLine("@echo test:" + getFullLogFolder());

      //------------put log files name in tempFiles.txt

7.    var tempFileStream = g_oFS.OpenTextFile("c:\\office\\dev14\\temp.txt");

8.    var slog = "";
```

```
9.    slog = tempFileStream.ReadLine();
10.    var logFolder = getFullLogFolder()+"\\"+slog;
11.    oTextStreamOut.WriteLine("dir /B /O-D " + logFolder+
">c:\\office\\dev14\\tempFiles.txt");
12.   tempFileStream.Close();
13.   getErrorLogPath(logFolder);
```

The temp.txt is like:

x64.ship.en-us.02

x64.ship.en-us.01

x64.debug.en-us.02

x64.debug.en-us.01

And at line 13 pass the log folder path which is x64.ship.en-us.02 to the function getErrorLogPath(), and in this function will search the .err file and for each .err file search the error message and save all the error message in g_sErrorMessage

```
function getErrorLogPath(logPath)
{
    var errorMessage="now this is empty";
      var tempLogFS = g_oFS.OpenTextFile("c:\\office\\dev14\\tempFiles.txt");
      var fileNames = new Array();
      var i=0;
      while (!tempLogFS.AtEndOfStream)
      {
            fileNames[i] = tempLogFS.ReadLine();
            i++;
      }
    //to match a file name to get more file could have more RE
```

```
      //var reLog = new RegExp("^.*\.err$","i");

      //Get all err files

      var reLog = new RegExp("^.*err$","i");

      var reRunCT = new RegExp("^RunCT.*","i");

      var errorLogName = "";

      //for each element find err file

      //g_sErrorMessage = "-----------Error Message-------------<br>";

    for (var j=0; j<fileNames.length; j++)

    {

      // search err

      errorLogName = fileNames[j].match(reLog);

            //for err file

            if (errorLogName)

        {  // file name + error message

             g_sErrorMessage +="&gt&gt&gt&gt&gt&gtError From File: " +
errorLogName + "&lt&lt&lt&lt&lt&lt<br>"

                  +
getErrorMessage(logPath+"\\"+errorLogName)+"<br>"+"&gt&gt&gt&gt&gt&gtEnd of
File&lt&lt&lt&lt&lt&lt<br><br>";

               if(haveLabIssue(logPath+"\\"+errorLogName))

                   alert("This is Lab Issue, do not open bug");

            }

    }
```

Get error message

```
function getErrorMessage(filePath)

{
        //read file into string
        var sError = "";
        var tempLogFS = g_oFS.OpenTextFile(filePath);
        var alines = new Array();
        var i=0;
        //read file
        while (!tempLogFS.AtEndOfStream)
        {
    var line  = tempLogFS.ReadLine();
         //get all lines before the first empty line
         if( line != "\\n" )
                {
                        sError += line + "<br>";
                }
         else
                {
                        break;
                }
        }
        return sError;
}
```

Put in e-mail body

Modify the e-mail template to add a paragraph of ‘ERRORMESSAGE” which indicate the location going to be replaced by actual error message.

```
<p>Log file directories, sorted from newest to oldest:</p>
```

```
<p>

LOGFILES

</p>

<p>

ERRORMESSAGE

</p>

<p>Check-ins to this project, Import Projects and Need Projects :</p>

<p>

CHECKINS

</p>
```

Before sent the e-mail replace the e-mail template with the error message with saved in a global string varialble.

```
sLine = sLine.replace(/ERRORMESSAGE/,g_sErrorMessage);
```

4.3 Finding Lab Issue

I already can get the error file path, then the following function just take the file path as parameter and search the specific string in the file, if the error file contain matched string return ‘true’ .

```
function haveLabIssue(filePath)

{

     var reSD = new RegExp("Source Depot client error","g");

       var tempLogFS = g_oFS.OpenTextFile(filePath);

       var file ="";

       var have = false;

       while (!tempLogFS.AtEndOfStream)

            {

                file = tempLogFS.ReadLine();
```

```
                    if (file.match(reSD))
                          have = true;
              }
       return have;
}
```

And this function is called in getErrorLogPath() function, each time get an error log file will call this function to check if it cause the by the lab issue, and if it is the case will open an alert window to warning builder do not open bug.

```
if (errorLogName)
        {  // file name + error message
             g_sErrorMessage +="&gt&gt&gt&gt&gt&gtError From File: " +
errorLogName + "&lt&lt&lt&lt&lt&lt<br>"
                  +
getErrorMessage(logPath+"\\"+errorLogName)+"<br>"+"&gt&gt&gt&gt&gt&gtEnd of
File&lt&lt&lt&lt&lt&lt<br><br>";
               if(haveLabIssue(logPath+"\\"+errorLogName))
                   alert("This is Lab Issue, do not open bug");
```

4.4 Add more receivers

Get 'NeedProjects'

Following is a piece of code from 'oNeedProjects.bat' use 'projects.pm', and extract each project and its 'NeedProjects' write into 'NeedProjects.txt' a text file.

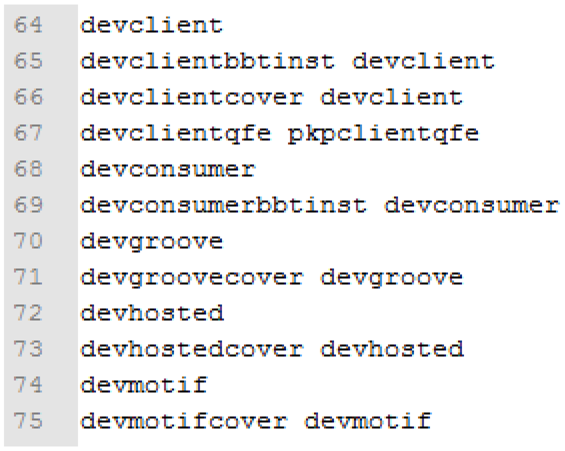
```
1 use Office::Projects;
2 open (NEEDFILE, ">NeedProjects.txt");
3 foreach my $project (sort keys %ProjectData)
```

```
{
 print NEEDFILE $project." ";
    my $need = $ProjectData{$project}{NeedProjects};
    my $i=0;
    while ($need->[$i])
    {
        print NEEDFILE $need->[$i]." ";
        ++$i;
     }
     print NEEDFILE "\n";
close (NEEDFILE);
```

The ‘oNeedProjects.bat’ is a batch file in Perl scripts, the reason to use batch file is because it is easy to run in command line and can be used for other tools or batch file exist in the system.

The ‘NeedProjects.txt’

This file include every project and its ‘NeedProjets’, each line start with the project name followed by its ‘NeedProjects’, all the project name separate by a space.



```
devclient
devclientbbtinst devclient
devclientcover devclient
devclientqfe pkpclientqfe
devconsumer
devconsumerbbtinst devconsumer
devgroove
devgroovecover devgroove
devhosted
devhostedcover devhosted
devmotif
devmotifcover devmotif
```

BBMT function GetNeedProjects ()

This function open the file ‘NeedProjects.txt’ and read line by line, put each line into an array, and find the failed project from the first elements of the array, when find the failed project return its ‘NeedProjects’ array maybe its empty.

```
Function GetNeedProjects ()
{
         var aProjectList = new Array();


         var fso = new ActiveXObject("Scripting.FileSystemObject");
         var needFileStream =
fso.OpenTextFile("c:\\office\\dev14\\bat\\NeedProjects.txt",1,false);
         var eachLine = "";
         var temp = new Array();
         var next = true;


         while ((!needFileStream.AtEndOfStream)&&next)
{
eachLine = needFileStream.ReadLine();
temp = eachLine.split(' ');
if ((g_sProjectName == temp[0])&&temp[1])
{
                    for (var i=1; i<temp.length-1; i++)
{
                    aProjectList[i-1]=temp[i];
                   }
                 next = false;
}
         }
         needFileStream.Close();

         return aProjectList;
```

BBMT function onSendEmail()

GetNeedProjects() is called here, and assign the value to array ‘aNeedProjects’, for each element in the array get the check-in developers which assign to string ‘sCheckInUsers’ and get all check-in changes which assign to string ‘sCheckinChanges’.

```
aNeedProjects = GetNeedProjects();
var sCheckInUsers = GetCheckinUsersForProjects(g_sProjectName, oAdoConn);
while(aNeedProjects[Count])
{
sCheckInUsers += GetCheckinUsersForProjects(aNeedProjects[Count++],
oAdoConn);
}
Count = 0;
```

```
var sCheckinChanges = GetCheckinUsersByChanges(g_sProjectName, oAdoConn);

while(aNeedProjects[Count])

{

sCheckinChanges += GetCheckinUsersByChanges(aNeedProjects[Count++],
oAdoConn);

}

oAdoConn.Close();

oAdoConn = null;
```

The check-in developers of failed project and its ‘NeedProjects’ were on the ‘To’ line of email contacts and ignore check-ins by integration.

```
if (sCheckInUsers)

{

// People who check in go on the To: line. Ignore integration checkins

sToList = sCheckInUsers.replace(/=y-arnold;(.*)/gi, "$1");

// People in ProjectInfo go on the CC: line

sCCList = sProjectContacts;

}
```

Open Build Break Email Template which is a text file pre-defined the email format and content. And replace with breaks information such as ‘PROJECT’, ‘TOLINE’ and ‘CHECKINS’.

```
// read in the break mail template and swap in the bug number, etc.

var oTextStream = g_oFS.OpenTextFile(g_sBreakMailTemplate);

if (oTextStream)

{

    var sLine = "";
    while (!oTextStream.AtEndOfStream)
    {
        sLine = oTextStream.ReadLine();
        sLine = sLine.replace(/PROJECT/, getProjectFlavorName());
        sLine = sLine.replace(/BUGNUM/g, g_sBugNumber);
        sLine = sLine.replace(/OFFICEVERSION/g, g_sOfficeVersion);
        sLine = sLine.replace(/BUILDNUM/, g_sFullBuildNumber);
        sLine = sLine.replace(/LOGFILES/, sLogErrors);
        sLine = sLine.replace(/CHECKINS/, sCheckinChanges);
        sLine = sLine.replace(/TOLINE/, sToList);
        sLine = sLine.replace(/CCLINE/, sCCList);
        sLine = sLine.replace(/BUILDER/g, sBuilder);
        sEmailContents += sLine;
    }
}
oTextStream.Close();
```

4.5 Summary

The actual coding to implement the solution is not the hard part, because the direction is already clear, and only need focus on the very specific problem which is related with programming languages.

Chapter 5 Testing and Evaluation

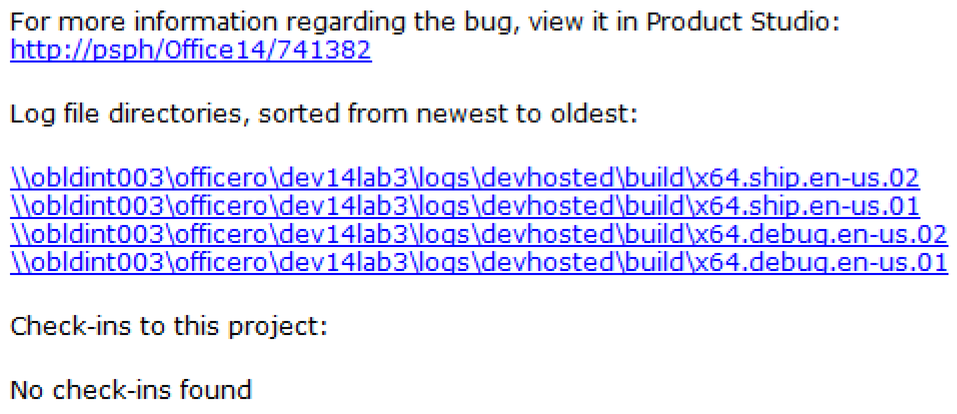
5.1 Introduction

I did some unit tests one the functions during coding, and here mainly is functional or system testing, which I test the new develop functions in the real working environment.

5.2 Test Adding Error Message

The original e-mail

In the original email there is only four links to the log folder.



For more information regarding the bug, view it in Product Studio:
http://psph/Office14/741382

Log file directories, sorted from newest to oldest:

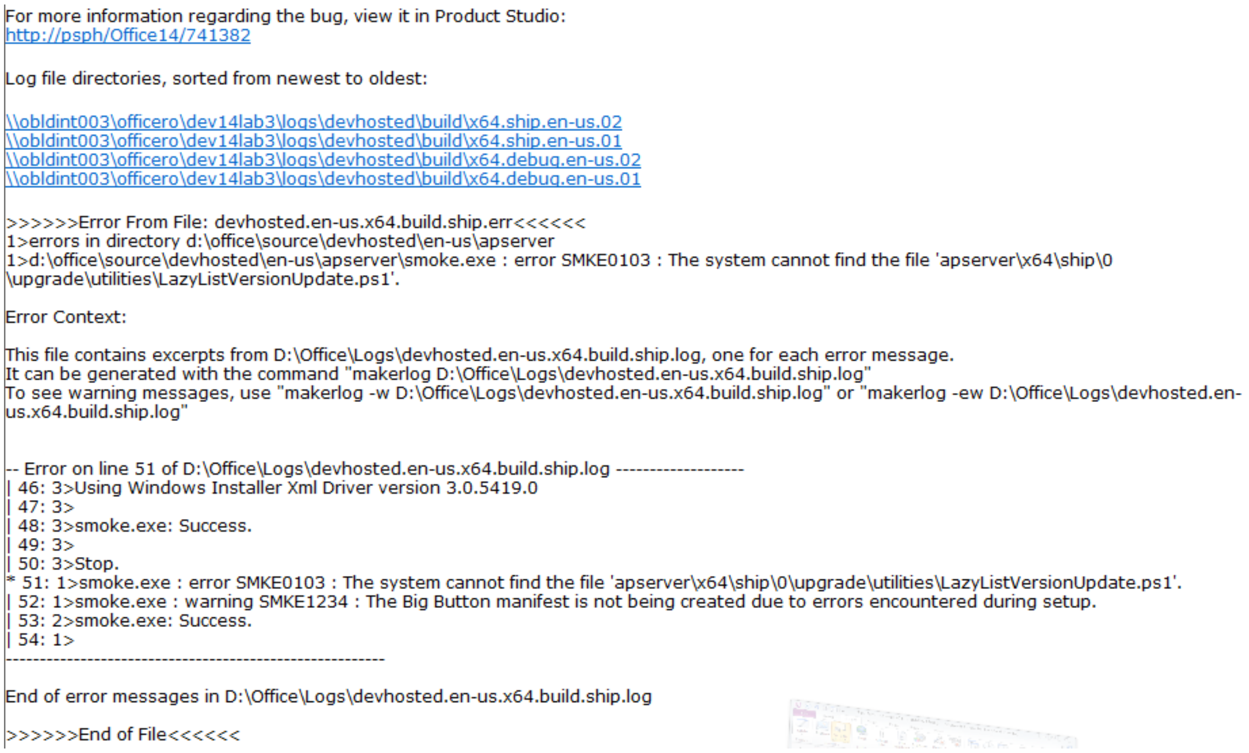
\\obldint003\officero\dev14lab3\logs\devhosted\build\x64.ship.en-us.02
\\obldint003\officero\dev14lab3\logs\devhosted\build\x64.ship.en-us.01
\\obldint003\officero\dev14lab3\logs\devhosted\build\x64.debug.en-us.02
\\obldint003\officero\dev14lab3\logs\devhosted\build\x64.debug.en-us.01

Check-ins to this project:

No check-ins found

The improved e-mail

In the improved email the error message from each error log is added.



```
For more information regarding the bug, view it in Product Studio:
http://psph/Office14/741382

Log file directories, sorted from newest to oldest:

\\obldint003\officero\dev14lab3\logs\devhosted\build\x64.ship.en-us.02
\\obldint003\officero\dev14lab3\logs\devhosted\build\x64.ship.en-us.01
\\obldint003\officero\dev14lab3\logs\devhosted\build\x64.debug.en-us.02
\\obldint003\officero\dev14lab3\logs\devhosted\build\x64.debug.en-us.01

>>>>>>Error From File: devhosted.en-us.x64.build.ship.err<<<<<<
1>errors in directory d:\office\source\devhosted\en-us\apserver
1>d:\office\source\devhosted\en-us\apserver\smoke.exe : error SMKE0103 : The system cannot find the file 'apserver\x64\ship\0
\upgrade\utilities\LazyListVersionUpdate.ps1'.

Error Context:

This file contains excerpts from D:\Office\Logs\devhosted.en-us.x64.build.ship.log, one for each error message.
It can be generated with the command "makerlog D:\Office\Logs\devhosted.en-us.x64.build.ship.log"
To see warning messages, use "makerlog -w D:\Office\Logs\devhosted.en-us.x64.build.ship.log" or "makerlog -ew D:\Office\Logs\devhosted.en-
us.x64.build.ship.log"


-- Error on line 51 of D:\Office\Logs\devhosted.en-us.x64.build.ship.log -------------------
| 46: 3>Using Windows Installer Xml Driver version 3.0.5419.0
| 47: 3>
| 48: 3>smoke.exe: Success.
| 49: 3>
| 50: 3>Stop.
* 51: 1>smoke.exe : error SMKE0103 : The system cannot find the file 'apserver\x64\ship\0\upgrade\utilities\LazyListVersionUpdate.ps1'.
| 52: 1>smoke.exe : warning SMKE1234 : The Big Button manifest is not being created due to errors encountered during setup.
| 53: 2>smoke.exe: Success.
| 54: 1>
--------------------------------------------------------

End of error messages in D:\Office\Logs\devhosted.en-us.x64.build.ship.log

>>>>>>End of File<<<<<<
```

5.3 Test Finding Lab Issue

This test I did not test in the real working environment, because to test this function require the actual lab issue happened, and I cannot use the error log files in the history, because they already removed from integrator, but as it use the same function to get error files and the function has been tested in previous example, here I just perform a unit test on function haveLabIssue(filePath) by provide two text file one contain the string pattern and the other not.

So I create a test environment, another HTA file and contain the haveLabIssue(filePath) and another function to call it.

```
<script>

function getFile()

{

   var file = name.value;

   if (haveLabIssue(file))
```

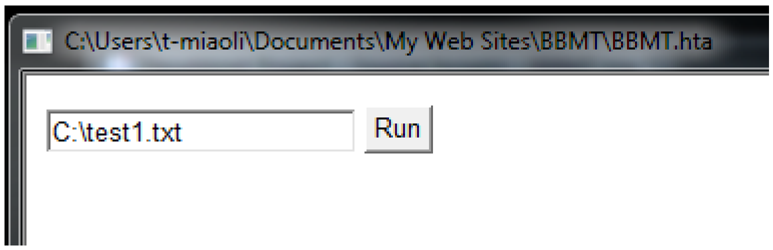
```
        alert ("this is caused by Lab Issue, Do not open bug");
   else  alert ("should open bug");


}
function haveLabIssue(filePath)
{
     var reSD = new RegExp("Source Depot client error","g");
                var tempLogFS = g_oFS.OpenTextFile(filePath);
                var file ="";
                var have = false;
                while (!tempLogFS.AtEndOfStream)
                            {
                                file = tempLogFS.ReadLine();
                                          if (file.match(reSD))
                                                      have = true;
                            }
                return have;
}


</script>


<input id="p" type="text" name="name" />
<input type="button" name="button1" value="Run" onClick="getFile()" />
```

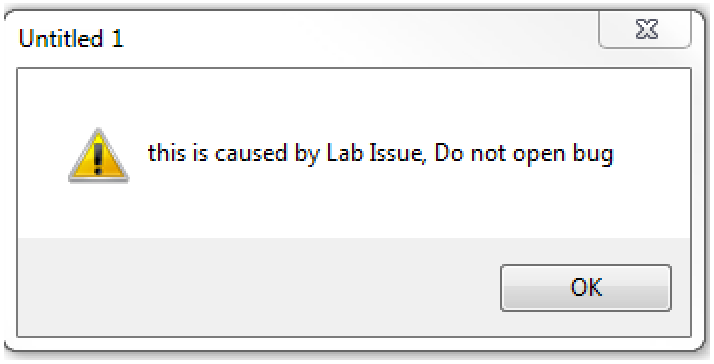
To run the program an provide a text file which contain the string pattern,



C:\Users\t-miaoli\Documents\My Web Sites\BBMT\BBMT.hta

C:\test1.txt

It shows the expected result



Untitled 1

this is caused by Lab Issue, Do not open bug

And provide another text file which do not contain the string pattern also get expected output.

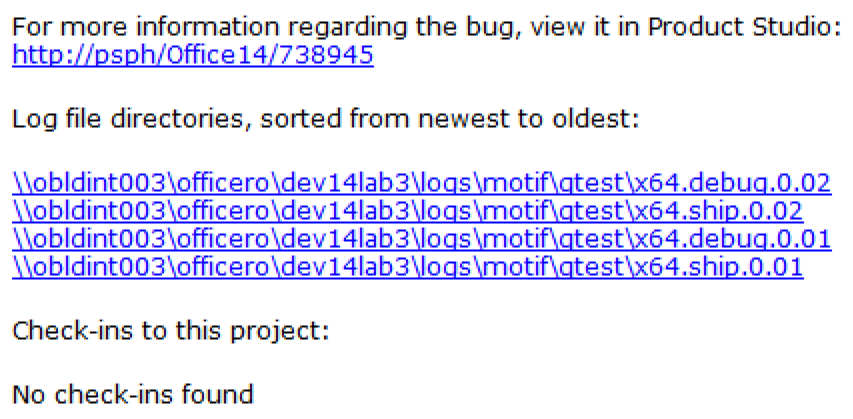
5.4 Test add more receivers

Test conditions:

There have ‘failed’ project in BuildUI and a bug has been open.

The ‘failed’ project has ‘NeedProjects’

Following Screenshot 5-1 is the original Build Break Email for bug #738945 for project ‘motif’.



For more information regarding the bug, view it in Product Studio:
http://psph/Office14/738945

Log file directories, sorted from newest to oldest:

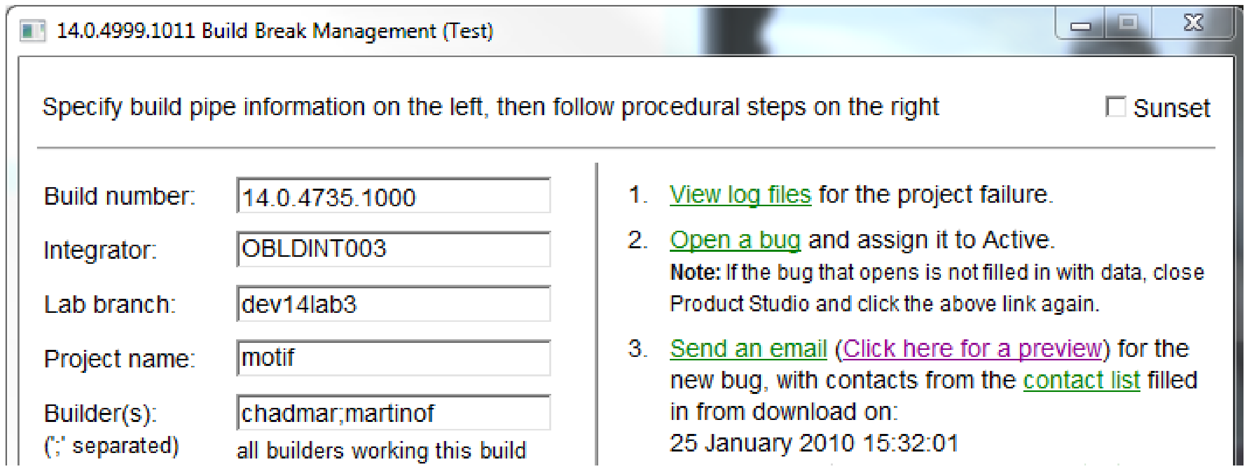
\\obldint003\officero\dev14lab3\logs\motif\qtest\x64.debug.0.02
\\obldint003\officero\dev14lab3\logs\motif\qtest\x64.ship.0.02
\\obldint003\officero\dev14lab3\logs\motif\qtest\x64.debug.0.01
\\obldint003\officero\dev14lab3\logs\motif\qtest\x64.ship.0.01

Check-ins to this project:

No check-ins found

Screenshot 5-1 original Build Break Email

Before the bug got fix by developer I re do the open process with BBMT, instead 'send an email' just for a preview.



14.0.4999.1011 Build Break Management (Test)

Specify build pipe information on the left, then follow procedural steps on the right

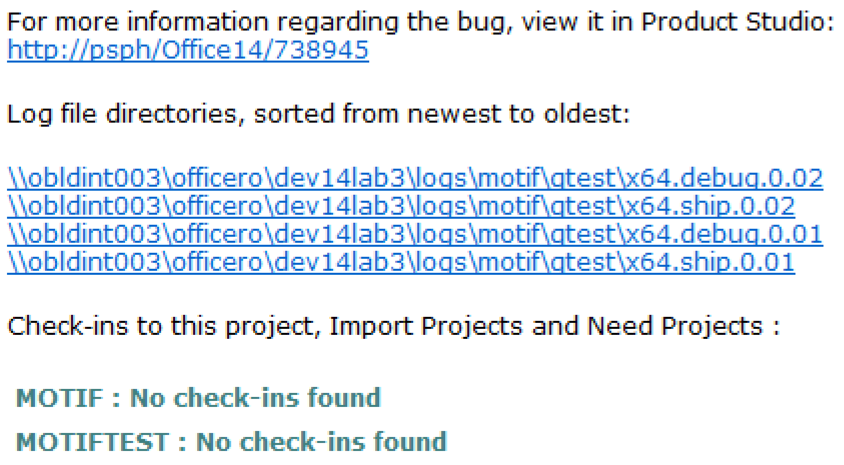
☐ Sunset

Build number:	14.0.4735.1000
Integrator:	OBLDINT003
Lab branch:	dev14lab3
Project name:	motif
Builder(s): (';' separated)	chadmar;martinof all builders working this build

1. View log files for the project failure.
2. Open a bug and assign it to Active.
   Note: If the bug that opens is not filled in with data, close Product Studio and click the above link again.
3. Send an email (Click here for a preview) for the new bug, with contacts from the contact list filled in from download on:
   25 January 2010 15:32:01

Screenshot 5-2 open bug for preview

The screenshot 5-2 shows the improved Build Break Email, for the same bug and project. Before only check project 'motif', now also includes another project 'motiftest' the 'NeedProjects' of 'motif'


For more information regarding the bug, view it in Product Studio:
http://psph/Office14/738945

Log file directories, sorted from newest to oldest:

\\obldint003\officero\dev14lab3\logs\motif\gtest\x64.debug.0.02
\\obldint003\officero\dev14lab3\logs\motif\gtest\x64.ship.0.02
\\obldint003\officero\dev14lab3\logs\motif\gtest\x64.debug.0.01
\\obldint003\officero\dev14lab3\logs\motif\gtest\x64.ship.0.01

Check-ins to this project, Import Projects and Need Projects :

MOTIF : No check-ins found
MOTIFTEST : No check-ins found

Screenshot 5-3 improved Build Break Email

5.5 Evaluation

The first improvement provides developers directly access to error message and reduce the network access limitation, save time for developer to take action to fix the problem. This can save approximate 2-3 minutes for each build break

The second improvement reduce builder workload, even though the lab issue is rarely happened, but when it happened the tool can detected it, that is very easy for builder to miss.

The third improvement includes all developers who make check-ins until the latest build, by add the check-in developer for 'NeedProjects'. There have about 5% projects have 'NeedProjects' dependency, in theory before the improvement the BBMT did not work for this 5% and developer need to find out the real cause and include the relative developer in the receivers.

5.6 Summary

The test result examples shown here is not the best example to show the difference, because I test the tool in the real environment, there do not have the suitable break can show the difference.

Chapter 6: Conclusion

The biggest problem for me to do the thesis is the visibility of the whole build system, as a builder I only familiar my work flow and tools, because the Microsoft Office Build System is extremely large and complex after long-term development, the way I learn the system is by some internal document and meeting with senior members.

The build break notification method used by Office build team generally meet the needs of the system, it can accurate and useful information and sent to the correspond developers, while because it is sent by builder manually so not very fast, in the future the tool should be automatic open bugs and sent feedback information to developers. And it will be better if the tool can provide more notification method to choose by developer. They can customize their favorite feedback methods, and information types and even the style how to represent. Feedback can be sent by e-mail and SMS, and developer can either way suitable.

Because the limit time, I did not implement all the features and I would like to keep working on it in the future.   By researching the area of feedback mechanism I also have some new ideas. I would suggest create e-mail template with XML rather than HTML. That would provide potential for new features, such as automatically open bug and search for solutions. As the e-mail defined by XML, the data could be saved in database or Active Directory which is searchable and manageable, that could be used to create an Expert System to help developers find similar solutions and automatically open bug under the complex situation.

Reference

[1] more information about Microsoft Ireland at

http://www.microsoft.com/ireland/about/

[2]http://en.wikipedia.org/wiki/Integrated_development_environment

[3]http://www.eclipse.org/

[4]http://msdn.microsoft.com/en-us/vstudio/default.aspx

[5]:The Art of Unit Testing: with examples in .Net, Pg.4

[6]Planning Extreme Programming , By: Kent Beck; Martin Fowler Integration Hell Pg.66 http://pqtechbus.safaribooksonline.com/0-201-71091-9

[7]Continuous Integration: Improving Software Quality and Reducing Risk, by Paul Duvall, Steve Matyas Pg.3

http://pqtechbus.safaribooksonline.com/9781933988276

[8]Continuous Integration: Improving Software Quality and Reducing Risk, by Paul Duvall, Steve Matyas Pg.10

http://pqtechbus.safaribooksonline.com/9781933988276

[9]Continuous Integration: Improving Software Quality and Reducing Risk, by Paul Duvall, Steve Matyas Pg.203

http://pqtechbus.safaribooksonline.com/9781933988276

[10]Quality Software Project Management, By: Robert T. Futrell; Donald F. Shafer; Linda I. Safer Pg.947

http://pqtechbus.safaribooksonline.com/0-13-091297-2

[Book1]: The Build Master, by Vincent Maraia

http://pqtechbus.safaribooksonline.com/0321332059

[Book2]: Software Testing, Second Edition, by Ron Patton http://pqtechbus.safaribooksonline.com/0672327988

[Office]http://office.microsoft.com/en-us/FX100647101033.aspx?pid=CL100569831033

[CI]http://martinfowler.com/articles/continuousIntegration.html Continuous Integration Martin Fowler

[GS]http://martinjandrews.github.com/greenscreen/

[Continuum]http://continuum.apache.org/

[CC]http://cruisecontrol.sourceforge.net/

[IRC]http://en.wikipedia.org/wiki/Internet_Relay_Chat

[Jabber]http://www.jabber.com/CE/JabberHome2

Appendix A: oNeedProjects.bat

```
@echo off

perl -x %~dpf0 %*

exit /b %errorlevel%



#!perl @rem = '-*- Perl -*-';

@rem = '

@if "%overbose%" == "" echo off

setlocal

call oenvtest.bat nodirs

set ARGS= %*

call perl%OPERLOPT% -w %~dpnx0 %ARGS:/?=-?%

exit /b %ERRORLEVEL%

';



#

# SCRIPTNAME

#

# Brief description of purpose of script.

#



BEGIN {        # common %otools%\bin\perl_template.bat script config -- do not edit

require "$ENV{OTOOLS}\\lib\\perl\\otools.pm"; import otools;

}
```

```
require 5;

use strict;

use Getopt::Long;

use Office::Projects;



#

# Display detailed usage information and quit.

#

sub usage {

print <<EOT;



Usage:

  ScriptName [options] ...



  Detailed description of meanings of options and arguments

EOT

exit 1;

}



#

# main

#



# first off, check if user needs help
```

```
GetOptions("help|h|?", \&usage);      # augment with arguments for other options



open (NEEDFILE, ">NeedProjects.txt");

foreach my $project (sort keys %ProjectData)

{

print NEEDFILE $project." ";

my $need = $ProjectData{$project}{NeedProjects};

my $i=0;

while ($need->[$i])

{

print NEEDFILE $need->[$i]." ";

++$i;

}

print NEEDFILE "\n";

}

close (NEEDFILE);

exit 0;
```

Appendix B: Email template (original)

Please reply to this email immediately so that we know you are managing the build break. The bug needs to be assigned to a developer in the next hour and the bug resolved within 1 to 2 hours (for compile breaks) or 2 to 4 hours (for CIT breaks). Note that backing out a changelist or turning off a test is often the fastest way to fix a break. For more information on build break responsibilities, see: http://office/sites/build/Shared Documents/Office Pipeline Build Breaks and Contact Policies.mht

For more information regarding the bug, view it in Product Studio:

http://psph/OfficeOFFICEVERSION/BUGNUM

Log file directories, sorted from newest to oldest:

LOGFILES

Check-ins to this project:

CHECKINS

While you may not have caused the build break, you are receiving this e-mail because you checked into the project since the last build (if you are on the To: line), or you are on the list of build break contacts (if you are on the CC: line). Please assign this bug to the person responsible for the build break as soon as possible. You can update your project contacts here: http://office/sites/build/Shared Documents/ProjectInfo14.xml

You are currently in the first stage of the build break notification process. If no action is taken, the automated build break escalation system will proceed with the next stage as outlined below:

Time	Mail Sent
Now	Mail sent out to build break contacts for all build breaking bugs
In one hour	Mail is sent out for unassigned bugs Mail is sent to dev owners of compile breaks
In two hours	Mail is sent out for unassigned bugs Mail is sent to dev owners of compile breaks Notification mail is sent to dev owners of CIT breaks
In four hours	Mail is sent out for unassigned bugs Mail is sent to dev owners of compile breaks Mail is sent to dev owners of CIT breaks

If you need more time than the schedule listed above to resolve this issue, please reply-all to this mail and CC your manager, providing an ETA for the fix.