Continuous Integration

Why and How to build a Continuous Integration Environment for the .NET platform

Document Version 1.0

Continuous Integration Guide and Reference Manual

What this document is for: This guide is a learning tool and also a detailed manual to reference for executives, managers and developers on 'Why and How to build a Continuous Integration Environment for the .NET platform'. It covers everything from the basics of Continuous Integration to significant business advantages, as well as overview diagrams to detailed setup. Documentation included covers requirements, processes, procedures, references and example code involved in setting up an environment.

Written By:
James Kaplewicz
Yousuf Baqir
Joseph Normandeau
Alex Begun
Eladio Martin
Andy Blum
Talha Anwer

How to use this guide:

Executives: Suggested - Sections 1, 2
Development Managers: Suggested - Sections 1, 2, 3, 4, 6, 8
Developers: Suggested - Sections 1, 2, 3, 4, 5, 6, 7, 8, 9

Each development environment has its own specific requirements. It is recommended to determine these up front by mapping the goals of the software and development architecture needed at a high level. Completion will likely be different from the initial expectations and it is important to track and understand differences to continually improve on the process of building Continuous Integration environments.

After building a set of requirements, this guide should be used as a framework to start building a Continuous Integration environment tailored to a specific project(s).

Please direct any questions or comments to whitepapers@espusa.com.

Table of Contents

1.0 Executive Summary

An important part of any software development process is getting reliable builds of the software. Despite its importance, we are often surprised when this isn't done.
– Martin Fowler

1.1 What is Continuous Integration (CI)?

Continuous Integration (CI) is the strategy and practice of making sure that changes to a software project's code base are successfully built, tested, reported on, and rapidly made available to all parties after they are introduced.

This section is simply a brief and high level explanation of Continuous Integration. Please refer to Section 8.0 References and Suggested Readings for more information.

There are several basic requisites in setting up CI environment:

- Source code should be maintained in a central location, preferably a source code control product like Visual SourceSafe or CVS.
- Each project has build scripts to create a build.
- All code bases include an auto update framework.

In a Continuous Integration Environment source code is maintained in a central location where an application monitors the repository and springs into action when it notices changes (commits) to the code.

The objective involves using a full version of a given projects code base whenever any part of it changes, automatically run a build file (or manually), run automated tests, report on the results for quick problem resolution, and quickly make all changes available to teams involved.

1.2 Advantages of Continuous Integration

- Dramatically increase ROI through full cycle efficiencies.
- Guarantees successfully compiled software.
- Visible progress reporting and problem tracking.
- Low TCO (Total Cost of Ownership).
- Relatively easy to build or integrate into an existing development environment.
- High impact environment upgrade with low maintenance.
- Improve development standards, consistencies and accountability.
- Increase amount of quality code.
- Rapidly identify bugs, who created them, and where it is.
- Quickly push high quality change updates to testing.
- Reduce development integration effort.

1.3 Disadvantages of Continuous Integration

- Migration of large volumes of internal development projects into a CI environment that are spread across various development environments require tight planning and coordination to successfully migrate.
- Requires understanding of CI and discretion when setting up projects.

2.0 Continuous Integration Best Practices

2.1 Understanding

That Continuous Integration itself is a software development Best Practice – central source file location, fully automated build, test and partly-automated distribution process that allows teams to build and test their software many times a day.

2.2 Buy In

This process starts with buy in for CI with Executives, which leads to Development Manager's understanding; the end result being CI execution from Developers.

2.3 Continued Learning

Development Managers and Developers commit to learning and following the CI practices that best fit their environment.

2.4 Back Up

Environment and Source File back-up and recovery procedures are properly followed.

2.5 Quick Deployment

One of the main goals with CI is the speedy development from concept to user; ensure that frequent builds are consistently helping this process from start to finish.

2.6 No Broken Builds

The developer's top priority is fixing breaks as new builds are created. Since CI involves a high amount of accountability, broken builds potentially affect every team member as well as the entire process. Team members must be able to rapidly trouble shoot and roll back changes to a successful build if needed.

2.7 Tight Build Promotion Process

Build promotion is how changes are taken from concept to development to the end user. The more consistent this process is, the more quickly ideas become a reality.

3.0 Definitions

This Section contains definitions of Continuous Integration components. The bottom of certain sections contains a line to additional information on the particular CI component. ESP believes it is important to be familiar with some of the terminology before spending a good amount of time reading about it.

3.1 Continuous Integration (CI)

Continuous Integration (CI) is the strategy of making sure that changes to the project's code base are built, tested, reported on, and rapidly made available to all parties after they are introduced. This framework frees up developers from having to constantly create new builds for the Test and QA groups. Each subsequent build is established and promoted, then downloaded automatically for testing.

Ref: http://www.theserverside.net/articles/showarticle.tss?id=ContinuousIntegration

3.2 Visual Source Safe (VSS)

Microsoft's Visual Source Safe (VSS) helps to manage projects regardless of the file type - (e.g. text files, graphics files, binary files, sound files, or video files) by saving them to a database. When a file is added to VSS, the file is backed up on the database, made available to other people, and changes that have been made to the file are saved so old versions can be recovered an at any time. Members of the team can see the latest version of any file, make changes, and save a new version in the database.

3.3 Source Off Site (SOS)

Source Off Site (SOS) provides a collaboration environment for users of Microsoft Visual SourceSafe 6.0 (VSS). SOS Collab combines source code control and bug tracking tool together with collaboration tools such as chat, discussion groups, and notifications.

The SOS Collab Servers directory does not contain any project data. All information is stored in the SQL database, located in the SQL installation folder.

SOS servers need to have VSS database on same machine or same LAN as the SOS Collab Server and VSS 6.0 APIs (installed with VSS 6.0 Client) on the same machine.

3.4 Successful Build

An aggressive goal for a successful build would include:

- All the latest sources are checked out of the configuration management system.

- Every file is compiled from scratch.
- The resulting object files are linked and deployed for execution.
- The system is started and a suite of tests is run against the system.
- If all of the above execute without error or human intervention and every test passes, that is a successful build.

Ref: http://www.martinfowler.com/articles/continuousIntegration.html

3.5 Build Promotion

Promoting a build means copying it to the target server. The build must not be rebuilt but rather moved. Therefore there is the need to identify and archive builds in a build promotion process (Section 6.0 Build Promotion Process) as builds are created.

Ref: http://www.15seconds.com/issue/040810.htm

3.6 Web Development Models (under VSS)

This defines Microsoft's suggested approach Web Application development in a team environment.

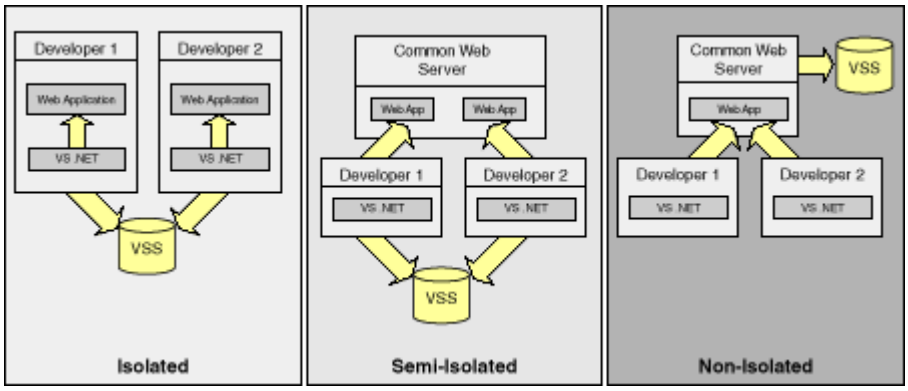

Figure 1 – Web Development Models - Isolated

It is strongly advised to adopt the "Isolated" development model shown in Figure 1 for team developments because it offers a number of significant advantages, for instance, each individual can continue working on a Web Application without the worry of another developer making a change conflicting with their own.

With the Isolated model, development (edit, debug and run) is in complete isolation on the developer's workstation using a local Web server (http://localhost). Access to the master source files is controlled via a Microsoft Visual SourceSafe (VSS) database located on a network file share.

Ref: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/tdlg_ch2.asp

3.7 NAnt

NAnt is an open source .NET build tool used to compile .NET applications.

3.8 NAntContrib

NAntContrib is the Project application for tasks and tools for NAnt. It provides extra tasks and tools that are not present in NAnt.

3.9 Draco.NET

Draco.NET is a Windows Service Application designed to facilitate Continuous Integration. Draco.NET monitors the source code repository, automatically rebuilds a project when changes are detected and then emails the build result along with a list of changes since the last build.

Draco.NET version 1.5 supports builds using NAnt build tool or via Visual Studio .NET 2002/2003 solution files.

3.10 NDoc

NDoc generates class library documentation from .NET assemblies and the XML doc files generated by the C# compiler (or with an add-on tool for VB.NET). It is used by NAnt build tool to automatically generate library documentation.

3.11 NUnit

A Unit-testing framework for all .NET languages. NUnit is built within NAnt and exposed as a NAnt task.

3.12 NUnitForms

NUnitForms is a tool extended from NUnit for automatically testing Windows Forms applications written on .NET.

3.13 FxCop

FxCop is a code analysis tool that checks .NET managed code assemblies for conformance to the Microsoft .NET Framework Design Guidelines. It uses reflection, MSIL parsing, and calls graph analysis to inspect assemblies for more than 200 defects in the following areas: Library design, Localization, Naming conventions, Performance, and Security.

FxCop includes both GUI and command line versions of the tool, as well as a SDK to create custom rules.

3.14CruiseControl.NET

CruiseControl.NET is similar to Draco.NET that can monitor the source code repository and automatically rebuilds a project when changes are detected. It has several extra features when compared to Draco.NET.

Ref:
http://confluence.public.thoughtworks.org/display/CCNET/What+is+CruiseControl.NET

4.0 CI Environment Overview

This section discusses high level details of Continuous Integration and outlines terminology for server components of the CI model. Included are the basic components in the Microsoft CI model and modifications ESP made. **NOTE:** The attached environments are only "logical" and not necessarily "physical". Applications will run the same on a single server as they will run in a complex network architecture.

4.1 Microsoft Team Development Environment

Microsoft's website provides "guidance and recommendations to enable you to set up a team development environment and work successfully within it" found at this link:

http://msdn.microsoft.com/library/en-us/dnbda/html/tdlg_rm.asp

The following diagram outlines what the Microsoft article discusses as "**In Scope"** for their "Team Development Environment" setup.

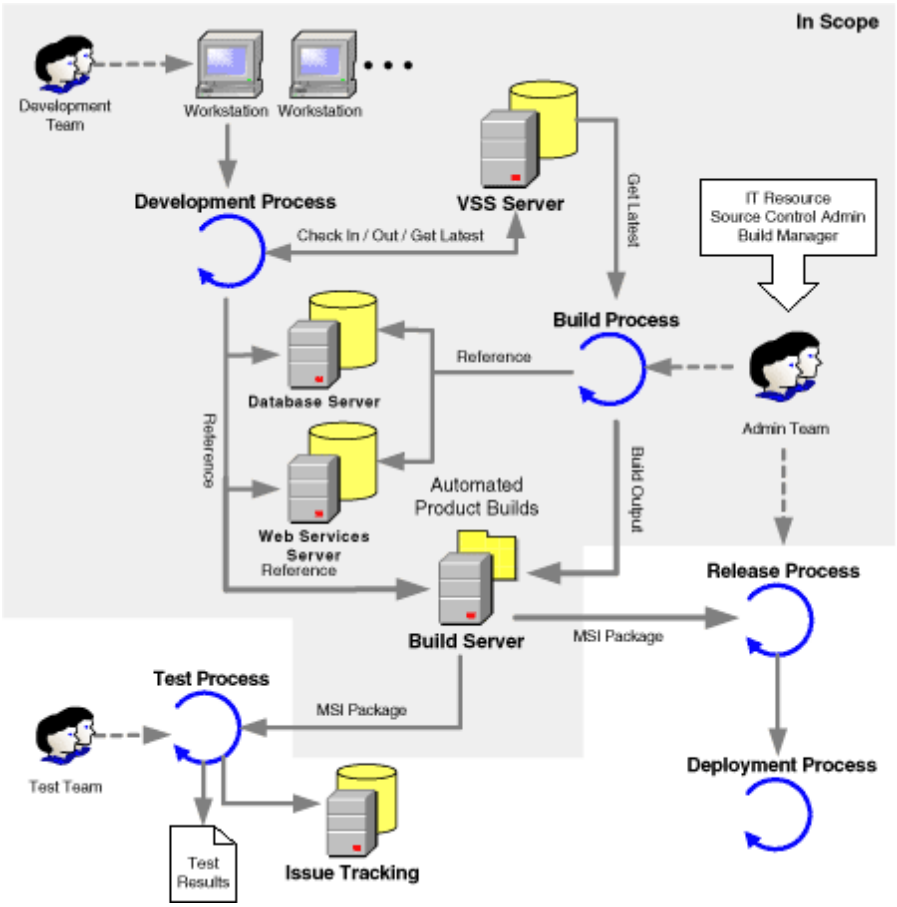

Figure 2 – Microsoft Team Development Environment

4.1.1 The VSS Server

This is a central server that hosts one or more Microsoft Visual SourceSafe (VSS) databases used to provide versioned controlled access to project source files. Developers interact with it on a daily basis to check project files in and out through the Microsoft Visual Studio.NET Integrated Development Environment (IDE). It is also accessed by the build script to obtain the latest source code required for the current system build.

4.1.2 The Build Server

The Build Server is an automated build script server used to compile and build entire projects. The build scripts are critical elements for all software development projects. These allow generation of successive versions of the system in an automated and consistent, repeatable fashion. The output assemblies generated by the build process are maintained in folders on this server.

4.1.3 Database Server

This server hosts instances of Microsoft SQL Server and provides a central location where developers can connect to databases whose schemas match the current system database design. In some scenarios, local SQL Server databases are needed on development workstations to perform isolated unit testing. For example, local servers allow the developer to manipulate a set of test data, remaining isolated and guarantee zero impact on other team members.

4.1.4 Web Services Server

The primary function of the Web Services Server in the team development environment is to host Extensible Markup Language (XML) Web Services that are currently under development. The central Web Server allows the services to be published for other developers or development teams such as testing, QA or the end user to reference these projects outside of a developer's local machine.

4.2 ESP's Continuous Integration Environment

At a high level the development architecture used at ESP is very similar to Microsoft with a few minor process improvements.

The image below shows the overview of ESP's Continuous Integration Environment.

- The grey area identifies what is to be covered "**In Scope"** in the rest of the documentation.
- The red text identifies differences between ESP's CI model and the Microsoft Team Development Environment model.

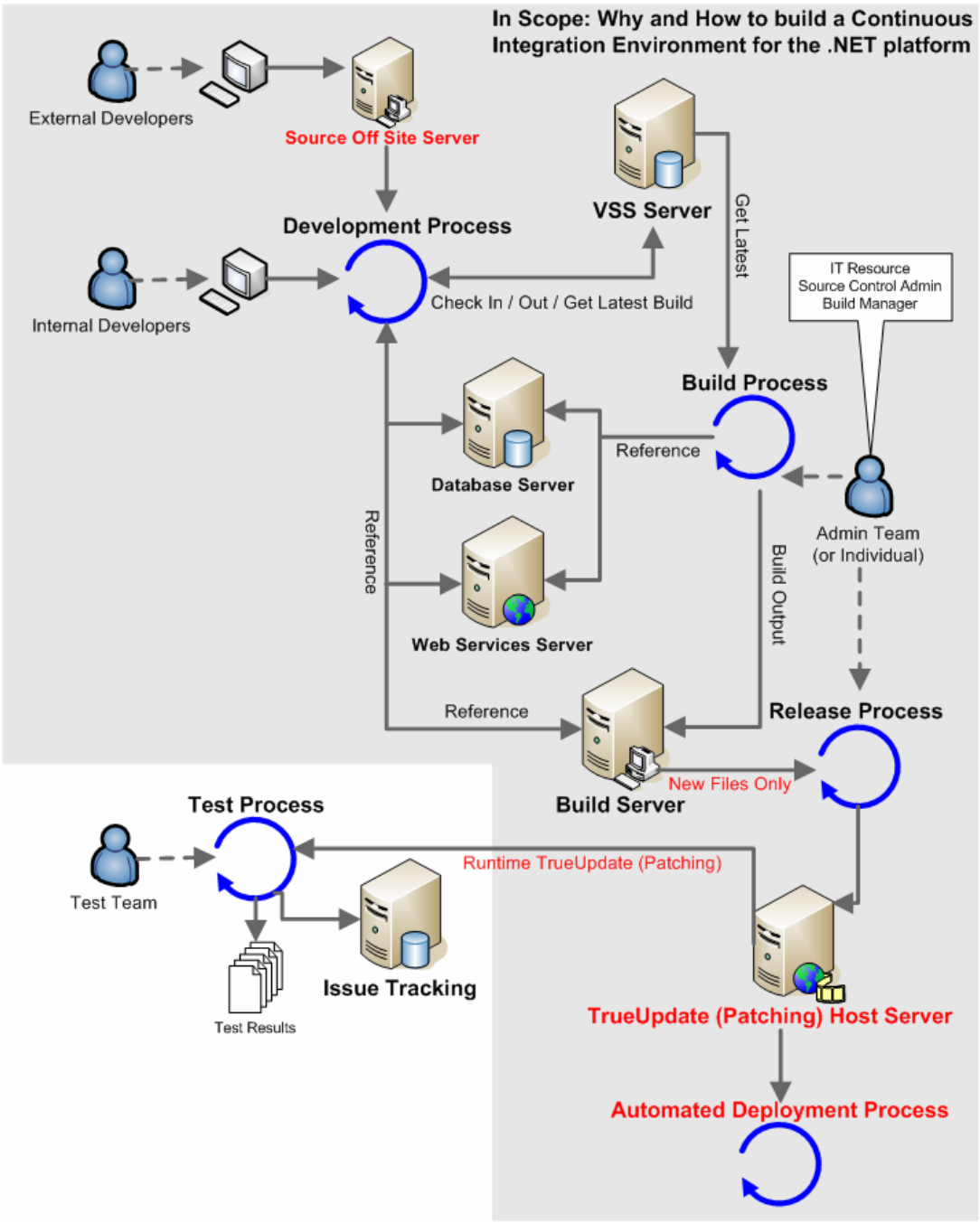

Figure 3 – ESP's Continuous Integration Environment

4.2.1 Source Off Site Server

This server interacts directly with the VSS server to access source files as part of the development process. Source Off Site allows those without access to the intranet a way to access source files and still maintain internal network infrastructure. This works well when two companies with differing infrastructures work together on a single project. In this case, all that is needed is web access. There are many other open source products or workarounds that can be used to perform this same function.

4.2.2 TrueUpdate (Patching) Host Server

The TrueUpdate Host Server is the Build Managers tool to distribute builds quickly and efficiently to individuals involved in a project. This server hosts a file that matches the client build versus the server build. If the server has a more recent version, the client connects and downloads the most recent version for use locally.

This architecture supports builds for Test, QA, and Production (End User) to be pushed out automatically after testing is approved. All distribution at this point is handled by the Build Manager(s). For more information about the Release Process (Build Promotion) please refer to Section 6.0 Build Promotion Process.

5.0 Building the CI Environment

5.1 General Requirements

At least one server is needed to start building the CI environment.

Prior to configuring these tools this guide will step through creating sample .NET applications to help demonstrate how these tools work and interact with each other. The guide uses two sample .NET applications, one Windows Application and one web (ASP.NET) application. The reason for using these two types of applications is to demonstrate the difference between the way the tools are configured for web and non-Web Applications.

Once these applications are created and added to source control (VSS) the C.I tools will be configured and scripts created for some tools. Each script that is created is discussed in detail.

Section 6.0 will discuss high level build promotion to different parts of the team environment. Accomplishing this in the real world requires a tight build promotion process and potentially an auto update (patching) feature for the Windows Application using TrueUpdate.

5.1.1 Server Requirements for the .NET Environment

5.1.1.1 .NET Framework 1.1 (optional with Visual Studio.NET 2003)

Visual Studio.NET is not a necessity for server machine, but it is installed since the guide uses Visual Studio.NET to create sample .NET applications.

These sample .NET applications can be created on some other machine with Visual Studio.NET installed and then copied on to the server. Installation of .NET Framework 1.1 is essential.

5.1.1.2 Microsoft Visual SourceSafe 6.0

Other source control environments could be used as well, such as Subversion or CVS. However, make sure the selected software is compatible with other CI Tools used in this guide. The following scripts used for each tool assume the use of VSS.

5.2 Installing and Understanding CI Tools

This section will detail the installation of the following CI tools on the server and discuss functionality. Please refer to Section 3.0 for more information on the tools and terminology.

5.2.1 CI Tools Requirements

The following tools are required:

- NAnt & NAntContrib
- Draco.NET
- Draco.NET Web Viewer
- NDoc
- FxCop
- NUnit
- TrueUpdate

The diagram below displays an installation architectural overview for general path info.

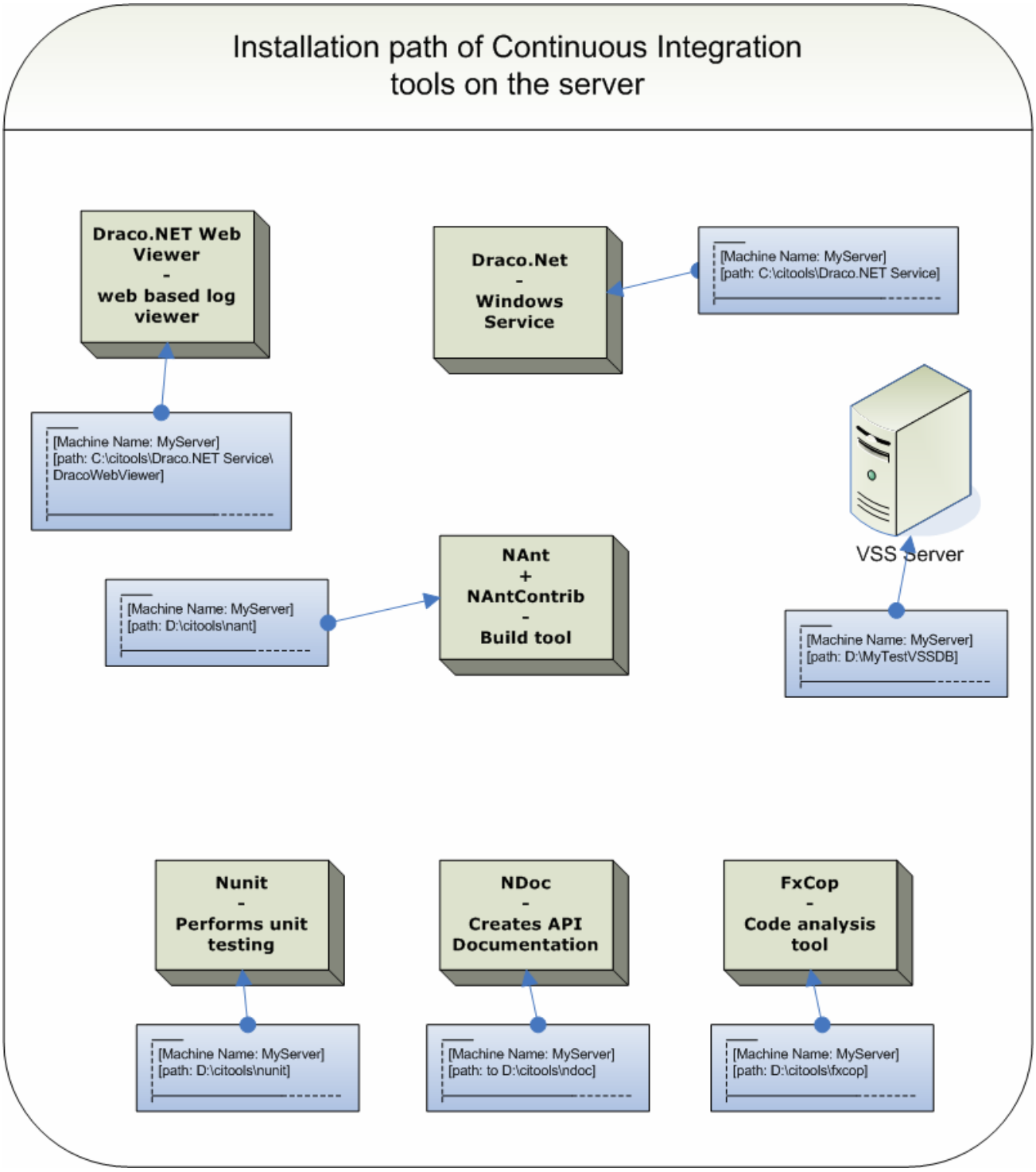

Figure 4 – Installation path of Continuous Integration tools on the server

5.2.2 NAnt & NAntContrib

NAnt is an open source .NET build tool used to compile .NET applications. The compilation is performed by running NAnt manually and passing the path to NAnt build scripts. The build script is an XML file that consists of different tasks that NAnt will perform, such as fetching latest source code from the source control, creating and/or deleting certain directories and/or files on the file system, building solutions, etc.

NAntContrib provides extra tasks and tools that are not present in NAnt.

5.2.2.1 Setting up NAnt along with NAntContrib

Follow these steps to set up NAnt along with NAntContrib

i Download latest version of **NAnt** from http://nant.sourceforge.net

At the time of writing this document the latest version is 0.85 - nightly build. Version 0.84 has a bug that does not allow compilation of Web Applications. Make sure to download Version 0.85.

ii Unzip NAnt to D:\citools\nant folder. Add the \nant\bin to systems Path environment variable.
iii Download latest version of NAntContrib from http://nantcontrib.sourceforge.net
iv Unzip NAntContrib to D:\citools\nantcontrib folder.
v Copy all the files from \nantcontrib\bin into \nant\bin

5.2.2.2 How does NAnt work?

NAnt is driven by an XML file that contains instructions about which projects to build and the *tasks* to be executed. The *task* (a tag in the NAnt build script) is where NAnt runs code and calls executables. *Tasks* can have multiple attributes or arguments. NAnt expects these XML build file to be named filename.build.

For more information about NAnt refer to http://nant.sourceforge.net/wiki

The basic function of NAnt is to build or compile .NET source code. The path to the .NET Solution file (.sln) is specified in one of the tasks of the NAnt script along with the path to the output directory where NAnt stores the result of compilation.

Along with compiling the source code NAnt also performs many other *tasks* as well, for instance to fetch source code from VSS and perform compilation on that source code. This *task* is in the build scripts along with several other *tasks*.

5.2.3 NDoc

NDoc generates class library documentation from .NET assemblies and the XML documentation files generated by the C# compiler (or with an add-on tool for VB.NET).

NDoc can be directly used by NAnt script to generate class library documentation through a *task* called <ndoc>.

To make class library documentation NDoc requires XML documentation files generated by Visual Studio.NET when the application is compiled. This is for C# only (the C# Project needs to be configured to generate XML documentation file in VS.NET). The reference of XML documentation file is present in the .csproj file, which is required by <solution> task to generate this XML on each compilation.

5.2.3.1 Setting up NDoc

i Download the latest version of NDoc from http://ndoc.sourceforge.net
ii Install NDoc to D:\citools\ndoc.

It is preferable to get familiar with NDoc before using NAnt's <ndoc> task to create the documentation. To get started with NDoc refer to http://ndoc.sourceforge.net/usersguide.html.

The <ndoc> task of NAnt creates documentation explained in Section '5.5 Creating NAnt Build Scripts for the Sample Applications'.

5.2.4 FxCop

FxCop is a code analysis tool that checks .NET managed code assemblies for conformance to the Microsoft .NET Framework Design Guidelines. It uses reflection, MSIL parsing, and call-graph analysis to inspect assemblies for more than 200 defects in the following areas: Library design, Localization, Naming conventions, Performance, and Security.

5.2.4.1 Setting up FxCop

i Download the latest version of FxCop from http://www.gotdotnet.com/team/fxcop
ii Install FxCop to D:\citools\fxcop.

To get started with FxCop refer to online documentation at http://www.gotdotnet.com/team/fxcop/gotdotnetstyle.aspx?url=DocFrameset.htm or FxCop.chm that comes along with the installation package.

Using FxCop through NAnt to perform code analysis is explained in section '5.5 Creating NAnt Build Scripts for the Sample Applications'.

5.2.5 NUnit

NUnit is a unit-testing framework for all .NET languages.

5.2.5.1 Setting up NUnit

i Download the latest version of NUnit from http://sourceforge.net/projects/nunit
ii Install NUnit to D:\citools\nunit.

To get started with NUnit refer to QuickStart.doc that comes along with the NUnit installation package.

Using NUnit through NAnt to perform unit testing is explained in Section '5.5 Creating NAnt Build scripts for the Sample Applications'.

5.2.5.2 Why is unit testing important?

When an engineer builds a car he first tests that every separate piece works together before putting all the parts together to build a car. If pieces do not work separately, then there is no chance of them working when they are together.

Unit testing is the process of testing each significant object and its method separately to simplify overall integration testing later on and to minimize the number of bugs. Any development process that implements rigorous unit testing standards leads to higher quality code and less bugs during integration.

5.2.6 Draco.NET

NAnt is used to build the .NET Solutions *manually*. Draco.NET is used to automate the build process.

Draco.NET is a Windows service application designed to facilitate continuous integration. Draco.NET monitors the source code repository and automatically rebuilds a project (by using compiling tools like NAnt or Visual Studio.NET) when changes are detected and then emails the build result along with a list of changes since the last build. Draco.NET uses a .NET XML configuration file to determine which VSS databases to poll and how often to poll them.

5.2.6.1 Setting up Draco.NET

i Download latest version of Draco.NET (Draco-Server-x.x.x.msi) from http://sourceforge.net/projects/draconet
ii Install Draco.NET to C:\citools\Draco.NET Service

Another option to automate the build process instead of Draco.NET is of using CruiseControl.NET – http://cruisecontrol.sourceforge.net

5.2.7 Draco.NET Web Viewer

Draco.NET Web Viewer is a web based log viewer for Draco.NET. The name of the build module can be specified (via Web.Config) along with the paths to the XML build logs that are created by Draco.NET. Once configured, Draco.NET Web Viewer will display one or more solutions and detailed result of each build.

5.2.7.1 Setting up Draco.NET Web Viewer

i Download latest version of Draco.NET Web Viewer from http://www.biasecurities.com/SoftwareBuilds/DracoWebViewer
ii Unzip the downloaded file to C:\citools\Draco.NET Service\DracoWebViewer. Convert DracoWebViewer folder to a web folder.

5.2.8 TrueUpdate

TrueUpdate is a Dynamic Software Update System for integrating automated updating capabilities into software products. TrueUpdate provides a framework for determining required updates and retrieving and applying the necessary patch or installation files via Internet, intranet or LAN.

TrueUpdate is not a freeware or open source like other tools. A 30-day trial version can be downloaded from the link given below.

5.2.8.1 Setting up TrueUpdate

i Download latest version of TrueUpdate from http://www.indigorose.com/tu/index.php
ii Run the downloaded executable and install it to C:\citools\TrueUpdate.

5.2.8.2 How does TrueUpdate work?

TrueUpdate consists of two separate components:

- A client-side executable that runs on the user's system and
- A server-side data file located somewhere else (typically on a web site or FTP server on the Internet).

The TrueUpdate client contains a list of locations where the server file can be found. At run time, the client executable downloads the server file from one of these locations, and then uses the information in the server file to guide the rest of the update process.

The server file contains version identifiers and actions. The version identifiers enable the client to distinguish one version of software from another. They consist of criteria such as the value of a specific registry key or INI file entry, the version from a file's resource information, or the CRC value of a specific file. For instance, if the installer writes the software's version number to the registry, a version identifier could be used to compare the value at that registry key with a specific version number. If the values matched, that version would be identified.

Once a version has been identified, the TrueUpdate client performs a series of actions associated with that version. For instance, if an update is available, the actions might instruct the client to download and then execute a patch file.

5.3 Creating sample .NET applications

5.3.1 Overview

Two .NET Solutions will be created under MySystem, one Windows Application, MyWinAppSolution with MyWinApp Project and one Web Application, MyWebAppSolution with MyWebApp Project. Both of these will be C# Projects since Visual Studio.NET facilitates the creation of XML documentation for C# (unlike for Visual Basic.NET); this will be used to create library documentation.

A similar folder structure is used to store the .NET Solutions as described under "Use a Consistent Folder Structure for Solutions and Projects" section at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/tdlg_ch3.asp

The same structure is also used for VSS.

For this case create a directory 'MyBuildSolutions' in the root D drive and add another directory 'MySystem' underneath this. Next create the two applications mentioned underneath the MySystem directory.

The folder structure should look as such after the .NET Solutions have been created:

```
D:
    --- MyBuildSolutions|
        --- MySystem|
                --- MyWinAppSolution|
                        --- MyWinApp|
                --- MyWebAppSolution|
                        --- MyWebApp|
```

Make sure to add XML comments through out the code when writing the applications.

5.3.2 Creating sample Win App using Visual Studio.NET

MyWinApp Application will be created to do some simple math. The figure below shows what the form will look like.

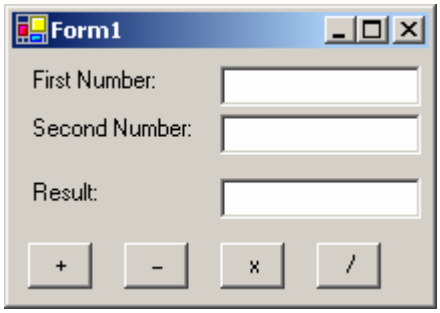

Figure 5 – Sample Win App

5.3.2.1 Create MyWinApp under MyWinAppSolution:

1. Open Visual Studio .NET, and on the File menu, point to New, and then click Project.
2. Select the Visual C# project type and Windows Application as template.
3. Enter the project name, MyWinApp, in the Name field.
4. Set the Location field to D:\MyBuildSolutions\MySystem.
5. Click the More button.
6. Select the Create directory for solution check box. This causes the project file to be created in a project sub folder beneath the solution folder.
7. Enter MyWinAppSolution into the New Solution Name field.
8. Click OK to complete the project and solution creation process.
9. Add new form to this project and add controls on the form to make it look like the figure above.
10. Add code for each click button as shown below:

```
private void btnPlus_Click(object sender, System.EventArgs e)
{
        tbResult.Text =
        MySimpleMath.Add(Convert.ToDouble(tbFirstNumber.Text),Convert.ToDouble(tbS
        econdNumber.Text)).ToString();
}
private void btnMinus_Click(object sender, System.EventArgs e)
{
        tbResult.Text =
        MySimpleMath.Subtract(Convert.ToDouble(tbFirstNumber.Text),Convert.ToDouble
        (tbSecondNumber.Text)).ToString();
}
private void btnMutiply_Click(object sender, System.EventArgs e)
{
        tbResult.Text =
        MySimpleMath.Multiply(Convert.ToDouble(tbFirstNumber.Text),Convert.ToDoubl
        e(tbSecondNumber.Text)).ToString();
}
private void btnDivide_Click(object sender, System.EventArgs e)
{
        tbResult.Text =
        MySimpleMath.Divide(Convert.ToDouble(tbFirstNumber.Text),Convert.ToDouble(t
        bSecondNumber.Text)).ToString();
}
```

This project will not compile yet since it uses MySimpleMath class which is created next.

To perform calculations, MyWinApp will use static class methods of MySimpleMath.cs for MyMathProject project in the same solution.

5.3.2.2 Create MyMathProject under MyWinAppSolution:

1. Right click the Solution's name in solution explorer and select New Project from Add sub menu.
2. Select the Visual C# project type and Class Library as template.
3. Enter the project name, MyMathProject, in the Name field.
4. Set the Location field to E:\MyTestProjects\MySystem\MyWinAppSolution.
5. Click OK to complete the project creation process.
6. Add new class to this project and name it MySimpleMath.cs
7. Add following static methods to this class:

```
public static double Add(double num1, double num2)
{
        return num1+num2;
}
public static double Subtract(double num1, double num2)
{
        return num1-num2;
}
public static double Multiply(double num1, double num2)
{
        return num1*num2;
}
public static double Divide(double num1, double num2)
{
        return num1/num2;
}
```

The next step is to create a NUnit test project to test the methods of MySimpleMath class. The .dll created by this Class Library project will be used by the NAnt script to perform unit testing.

5.3.2.3 Create MyMathProjectTest under MyWinAppSolution:

1. Right click the Solution's name in solution explorer and select New Project from Add sub menu.
2. Select the Visual C# project type and Class Library as template.
3. Enter the project name, MyMathProjectTest, in the Name field.
4. Set the Location field to E:\MyTestProjects\MySystem\MyWinAppSolution.
5. Click OK to complete the project creation process.
6. Add new class to this project and name it MyTest.cs.
7. Add [TestFixture] attribute to this class.
8. Add reference of nunit.framework.dll to this project. This .dll is placed in \nunit\bin
9. Add following test methods to MyTest class:

```
[Test] public void Add()
{
        Assert.AreEqual(3.3, MyMathProject.MySimpleMath.Add(1.1, 2.2), "Addition
        failed");
}
[Test] public void Subtract()
{
        Assert.AreEqual(1.1, MyMathProject.MySimpleMath.Subtract(2.2, 1.1),
        "Subtraction failed");
}

[Test] public void Multiply()
{
        Assert.AreEqual(25, MyMathProject.MySimpleMath.Multiply(5, 5), "Subtraction
        failed");
}
[Test] public void Divide()
{
        Assert.AreEqual(5, MyMathProject.MySimpleMath.Divide(10, 2), "Division
        failed");
}
[Test] public void DivideFail()
{
        Assert.AreEqual(1.1, MyMathProject.MySimpleMath.Divide(10,0).ToString(),
        "Expected failure");
}
```

Notice that the [Test] attribute to each of the test methods. This is how NUnit knows which method is a test method.

5.3.3 Creating a sample Web App using Visual Studio.NET

The main goal of creating this Web Application is to demonstrate how the NAnt script differs from those used for non-Web Applications. Therefore the functionality of this Web Application could be anything.

5.3.3.1 Create MyWebApp under MyWebAppSolution:

1. Open Visual Studio .NET, and on the File menu, point to New, and then click Blank Solution.
2. Enter MyWebAppSolution as the solution name, and then set its location to D:\MyBuildSolutions\MySystem.
3. Click OK. Visual Studio .NET creates the MyWebAppSolution folder beneath the specified root folder location.
4. Use Windows Explorer to create a new subfolder called MyWebApp beneath the D:\MyBuildSolutions\MySystem\MyWebAppSolution folder.
5. Use either Windows Explorer or Internet Services Manager to establish the MyWebApp folder as an IIS virtual root.
   Note: Visual Studio .NET requires that the virtual root name match the project folder name.
6. Return to Visual Studio .NET, and on the File menu, point to Add Project, and then click New Project.
7. Add an ASP.NET Web Application project.

8. In the Location field, enter http://localhost/MyWebApp, and then click OK. The project and Web source files will be created in the specified virtual root.

5.3.4 Adding sample .NET applications to VSS

Refer to section **"How to Check in a New Solution to VSS"** from http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/tdlg_ch6.asp for adding the .NET applications to VSS.

New Solutions are added to VSS by being on server machine itself; although they can be added to VSS remotely if Solutions are created elsewhere.

An important thing to remember about source control is always to use Visual Studio .NET to create projects and solutions in VSS. Visual Studio .NET ensures that only the appropriate files are placed under source control, and that files are updated with the proper VSS-specific information whenever they are checked in, checked out, or copied. Manipulating a solution with the VSS Explorer is a recipe for disaster. (Other tools such as Source off Site, SOS, which links up with VSS can be used to manipulate projects also.)

When placing a solution or project under source control, Visual Studio .NET adds the following file types to VSS:

- Solution files (*.sln)
- Project files (*.csproj, *.vbproj, etc)
- Application configuration files (Web.config or App.config)
- Source files (*.cs, *.vb, *.aspx, *.asax, *.resx, *.vsdisco, *.css, and so on).

Visual Studio .NET creates many other types of files on developer workstations and on the build server. These include solution user options, project user options, webinfo files, and project build outputs. These files are not placed under source control.

The following files are not added to source control because they are developer specific:

- Solution user option files (*.suo).
- Project user option files (*.csproj.user or *.vbproj.user).
- WebInfo files (*.csproj.webinfo or *.vbproj.webinfo).
- Build outputs that include assembly dynamic-link libraries (DLLs), Interop assembly DLLs and executable files.

It is recommended that developers also check out sections **"Working on an Existing Solution for the First Time"** and "**Working on an Existing Solution for a Subsequent Time".**

5.4 Creating & Modifying Scripts for CI Tools

After the installation of CI tools and creating sample .NET applications the next step is to configure the CI tools to work. Next Steps include:

- Create NAnt build script to work on both the .NET Solutions created.
- Configure Draco.NET's build script to run the NAnt scripts.
- Configure Draco.NET Web Viewer to display the result of each Solutions build.

Figure 2 below summarizes how these tools will collaborate once they are set into action.

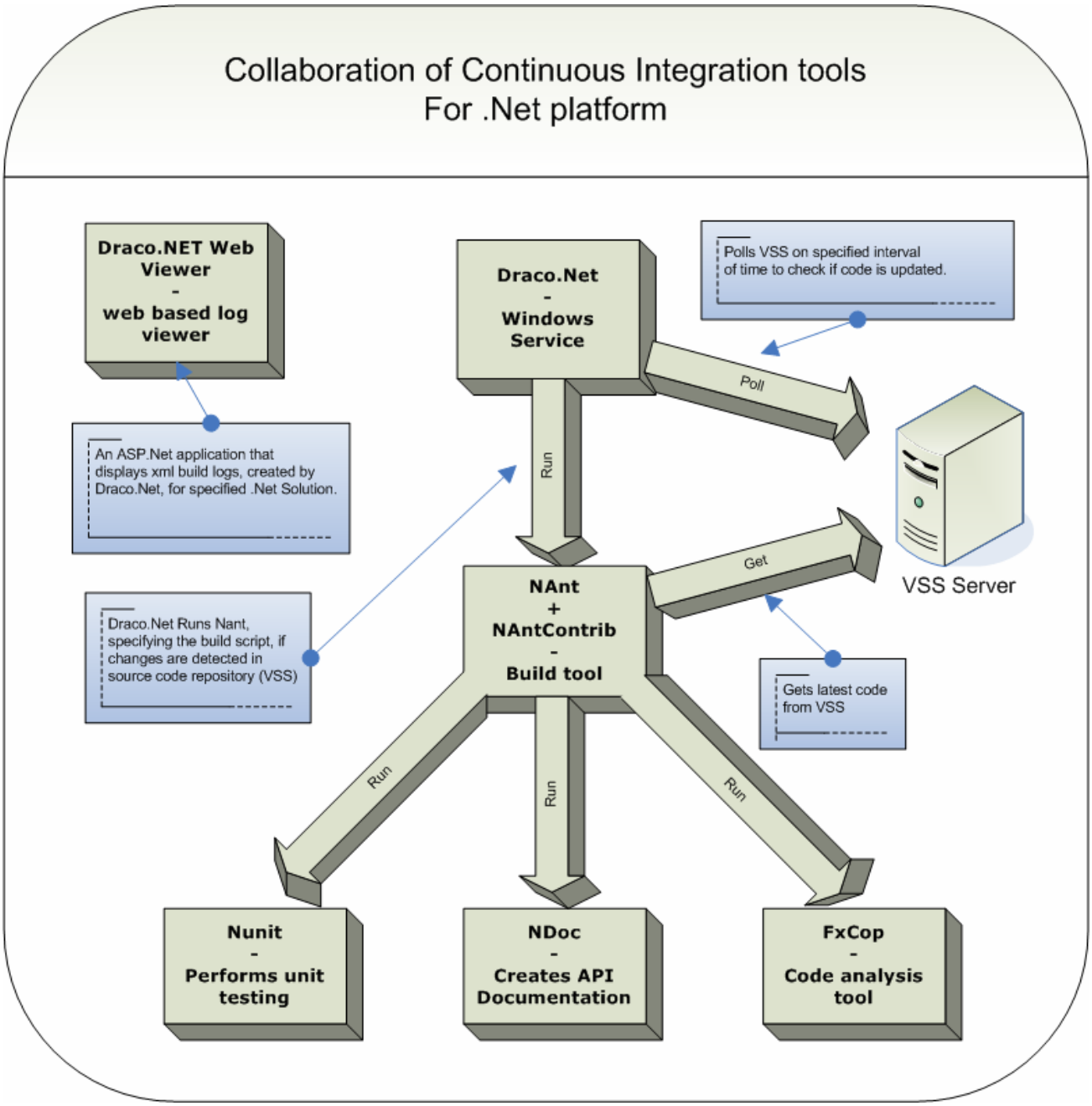

Figure 6 – Collaboration of Continuous Integration tools for .NET platform

Draco.NET polls VSS to check for source code update. If Draco.NET finds new source code then it tells NAnt which NAnt build script to run and sets NAnt into action. Nant then fetches the source code from the VSS and places it on to the specified path on the file system. It then compiles that source code and run NUnit, NDoc and FxCop as specified in the NAnt script.

5.5 Creating NAnt Build Scripts for the Sample Applications

Before creating the build script it is important to mention how to deal with the previous builds of the system.

5.5.1 Maintaining previous builds

To maintain the output from previous system builds a technique is used incorporating the version number and storing the folder name as its version number. This allows the Build Manager to promote specific versions of the application for Test, QA, and End Users to download via TrueUpdate without being affected by ongoing development changes. Without TrueUpdate this process would simply be a copy to location process.

When NAnt compiles the source code, the build output is stored in the 'Latest' folder. On successful build it will be copied to the folder of its version. Unsuccessful builds are discussed later on.

```
D: --- MyBuildSolutions|
        --- MySystem|
            --- MyWinAppSolution|
                --- MyWinApp|
                    --- bin|
                        --- Latest|
                        --- 1.0.0.1|
                        --- 1.0.0.2|
                        --- 1.0.0.3|
            --- MyWebAppSolution|
                --- MyWebApp|
```

The versioning system is not applied to the Web Application above. This is because Web Applications have many different files such as Web.Config and .aspx files apart from the executable. Since execution of a Web Application is dependent on the virtual paths of these files, they have to be kept in one place.

(All the files required to run the Web Application can be copied to folder with a version number, the same as for MyWinAppSolution. Directly executing the application will not be possible however, it must be copied manually to the correct path before execution).

Note: Each source file can be labeled in VSS with current version number to keep track of relationship between the build output and source files. But since Draco.NET service runs NAnt after it detects any

change in the source files in VSS, applying labeling functionality makes Draco.NET find changed source files on every poll which makes run NAnt on every poll! At this point ESP doesn't use this process.

Next is creating two NAnt build scripts, one each the Windows Application and another for the Web Application. Call them MyWinAppSolution.build and MyWebAppSolution.build respectively.

Although these scripts can be placed anywhere, try keeping them in one place under D:\MyBuildScripts\.

To get started with creating the build script check out the sample scripts that come with the NAnt application.

To get details about the tag and their attributes of NAnt build script refer to:
http://nant.sourceforge.net/help and/or http://nantcontrib.sourceforge.net/help

5.5.2 Creating MyWinAppSolution.build

Create the *project* tag named MySystem:

```
<project name="MySystem" default="build">
```

default="build" states that the default <target> *task* is 'build' and that will be executed if no task is specified on the command line when running this NAnt script. Everything else added to the build script exists under this <project> tag.

Properties are used to set some of the values used later in the script. Add the following to MyWinAppSolution.build.

```
<!-- Properties -->
<property name="buildbase" value="D:\MyBuildSolutions\MySystem" />
<property name="solutionName" value="MyWinAppSolution" />
<property name="basename" value="MyWinAppSolution" />
<property name="basepath" value="${buildbase}\${basename}" />
<property name="build.config" value="DEBUG"/>
<property name="build.dir" value="${basepath}\MyWinApp\bin" />
<property name="verbose" value="true" />
<property name="fxcop.location" value="D:\Program Files\Microsoft FxCop 1.30" />
```

Each name can be used as a handle to its value by referring it as ${property-name} for example ${buildbase}.

The <target> *task* is an executable task that can do one or more things.

The **build** <target> *task* is the default task that uses <solution> *task* to compile the application along with calling other <target> *tasks* to perform other operations.

```
<target name="build">
    <call target="getLatestFromVss"/>
    <call target="performVersioning"/>
```

```
        <call target="clean"/>

        <solution outputdir="${build.dir}\${sys.version}"
                  configuration="${build.config}"
              solutionfile="${buildbase}\${solutionName}\${solutionName}.sln"
                  verbose="true">
        </solution>
        <call target="runNUnitTests" />
            <call target="copyToLatest"/>
        <call target="runNDoc"/>
        <call target="runFxCop"/>
        <call target="copyResultsToLatest"/>
    </target>
```

Refer to http://nant.sourceforge.net/help/tasks/solution.html for description of <solution> task and its parameters.

Use <csc> *task* for C# Projects or <vbc> for VB.NET Project instead of <solution> *task* to compile the build. The <solution> task is preferred because it automatically determines project dependencies from inter-project references unlike <csc> task.

All the properties used in the <solution> task are defined in properties section except for sys.version property. We will discuss about this property along with performVersioning <target> task.

Now lets create each of the <target> called from within build <target>.

The **getLatestFromVss** <target> task gets the latest code from VSS by using <vssget> task.

```
    <!-- Get latest code version from VSS -->
    <target name="getLatestFromVss">
        <vssget user="Admin"
                password=""
                localpath="${basepath}"
                recursive="true"
                replace="true"
                writable="true"
                dbpath="D:\MyTestVSSDB\srcsafe.ini"
                path="$/MyBuildSolutions/MySystem/${basename}/" />
    </target>
```

Refer to http://nantcontrib.sourceforge.net/help/tasks/vssget.html for description of <vssget> task and its parameters.

The **performVersioning** <target> task uses <version> task of NAntContrib to add versioning to MyWinAppSolution.

Create a text file called build.number in D:\MyBuildSolutions\MySystem\MyWinAppSolution, and add 1.0.0.1 to the file. The script below will cause the build.number file to have its file version incremented by one, and the value of the version to be put into the NAnt property sys.version.

```
<!-- Increments the version number -->
<target name="performVersioning">
    <version buildtype="noincrement"
             prefix="sys."
             revisiontype="increment"
             path="${basepath}\build.number"/>
</target>
```

Refer to http://nantcontrib.sourceforge.net/help/tasks/version.html for description of <version> task and its parameters.

The **clean** <target> task uses <delete> task to delete all the files in the delete 'Latest' folder.

```
<!-- Delete 'Latest' version -->
<target name="clean" description="cleans Latest build directory">
    <delete>
        <fileset basedir="${build.dir}\Latest\">
            <includes name="*.*"/>
        </fileset>
    </delete>
</target>
```

Refer to http://nant.sourceforge.net/help/tasks/delete.html for description of <delete> task.

After clean <target> task is executed the <solution> task compiles the source code and stores the build files to ${build.dir}\${sys.version} where ${sys.version} is the number set by <version> task. Remember that output of all the Projects within the MyWinAppSolution goes to this same directory.

The **runNUnitTests** <target> task uses <nunit2> task to perform unit testing. Test dll's to run are specified under the <test> tag. The result output path and type is specified in <formatter> tag.

```
<!-- run the nunit task on the test dlls -->
<target name="runNUnitTests" description="Runs unit tests on specified dlls">
        <nunit2 failonerror="false" verbose="true">
            <formatter outputdir="${build.dir}\${sys.version}\"
                       usefile="true"
                       type="XML"
                       extension=".xml"/>
            <test>
                <assemblies basedir="${build.dir}\${sys.version}\">
                    <includes name="MyMathProjectTest.dll"/>
                </assemblies>
            </test>
        </nunit2>
    </target>
```

Refer to http://nant.sourceforge.net/help/tasks/nunit2.html for description of <copy> task.

The **copyToLatest** <target> task copies the build files from the current version number directory to the 'Latest' directory.

```
<!-- Copies the sys.version build to Latest -->
<target name="copyToLatest" >
    <copy todir="${build.dir}\Latest\">
        <fileset basedir="${build.dir}\${sys.version}\">
            <includes name="*.*" />
        </fileset>
    </copy>
</target>
```

Refer to http://nant.sourceforge.net/help/tasks/copy.html for description of <copy> task.

The **runNDoc** <target> task uses <ndoc> task to create MSDN type API documentation for the specified application (exe or dll) and its XML documentation. The facility to create this documentation is directly built into NAnt and it is not required to install NDoc on to the system.

In this case documentation is created for MyWinApp.exe.
MyWinApp.xml is created by <solution> task on source code compilation.

```
<!-- Creates msdn type API documentation for the application -->
<target name="runNDoc" description="Will create documentation for Buidling Solution">
  <ndoc>
    <assemblies basedir="${build.dir}\${sys.version}">
      <includes name="MyWinApp.exe" />
    </assemblies>
    <summaries basedir="${build.dir}\${sys.version}">
      <includes name="MyWinApp.xml" />
    </summaries>
    <documenters>
      <documenter name="MSDN">
<property name="OutputDirectory" value="${build.dir}\${sys.version}\doc" />
      <property name="HtmlHelpName" value="MyWinAppHelp" />
      <property name="HtmlHelpCompilerFilename" value="hhc.exe" />
      <property name="IncludeFavorites" value="False" />
      <property name="Title" value="An NDoc Documented Class Library" />
      <property name="SplitTOCs" value="False" />
      <property name="DefaulTOC" value="" />
      <property name="ShowVisualBasic" value="True" />
      <property name="ShowMissingSummaries" value="True" />
      <property name="ShowMissingRemarks" value="True" />
      <property name="ShowMissingParams" value="True" />
      <property name="ShowMissingReturns" value="True" />
      <property name="ShowMissingValues" value="True" />
      <property name="DocumentInternals" value="False" />
      <property name="DocumentProtected" value="True" />
      <property name="DocumentPrivates" value="False" />
      <property name="DocumentEmptyNamespaces" value="False" />
      <property name="IncludeAssemblyVersion" value="False" />
      <property name="CopyrightText" value="" />
      <property name="CopyrightHref" value="" />
      <property name="OutputTarget" value="HtmlHelp" />
      <property name="CleanIntermediates" value="True" />
```

```
        </documenter>
      </documenters>
    </ndoc>
  </target>
```

Refer to http://nant.sourceforge.net/help/tasks/ndoc.html for description of <ndoc> task.

The **runFxCop** <target> task executes FxCopCmd.exe, a command line version of FxCop, through <exec> task. (At the time of writing this guide NAnt does not have a task for FxCop like <ndoc> for NDoc).

The FxCopCmd command takes in .FxCop project file as an argument which it uses to validate against the default FxCop standards. Create an .FxCop file for MyWinAppSolution by following these steps:

i Open FxCop
ii Click Project, Add Targets (Ctrl+Shift+A)
iii Browse to D:\MyBuildSolutions\MySystem\MyWinAppSolution\MyWinApp\bin\MyWinApp.exe
iv Click file, save, and save as D:\MyBuildScripts\MyWinAppSolution.FxCop

Add the runFxCop <target> task to the build script.

```
<!-- Runs the command line ver of FxCop passing in the .fxcop file of this Solution. -->
<target name="runFxCop">
   <exec basedir="${fxcop.location}"
         program="FxCopCmd.exe"
         commandline="/p:D:\MyBuildScripts\MyWinAppSolution.FxCop
        /o:${build.dir}\${sys.version}\MyWinApp-FxCop-Results.xml"
         failonerror="false" />
</target>
```

This task outputs result of code analysis against MyWinApp application as an XML file called MyWinApp-FxCop-Results.xml created in the same directory as the build files (current version directory).

Refer to http://nant.sourceforge.net/help/tasks/exec.html for description of <exec> task.

The **copyResultsToLatest** <target> task copies the result files from the 'sys.version' folder to the latest folder. The result files can be any file ending in '-results.xml'. In this case these are result files from FxCop (MyWinApp-FxCop-Results.xml) and unit testing by NUnit, (MyMathProjectTest.dll-results.xml).

```
<!-- Copies the result files from the sys.version folder to the latest folder -->
<target name="copyResultsToLatest">
        <copy todir="${build.dir}\Latest\">
            <fileset basedir="${build.dir}\${sys.version}\">
                <includes name="*-Results.xml" />
            </fileset>
    </copy>
</target>
```

5.5.3 Creating MyWebAppSolution.build

Build script for Web Application is similar to non-Web Application build script with few exceptions. The URL path also needs to be declared that will be used by 'webmap' sub task within 'solution' task.

The script below just shows how <solution> task is different in Web Application from that of non-Web Application NAnt script. Other tasks for this script will be created same as that for MyWinAppSolution.build except for changing the paths specific to this application.

```
<!-- Properties -->
<property name="buildbase" value="D:\MyBuildSolutions\MySystem" />
<property name="solutionName" value="MyWebAppSolution" />
<property name="basename" value="MyWebAppSolution" />
<property name="basepath" value="${buildbase}\${basename}" />
<property name="Host" value="http://localhost" />
<property name="Path" value="D:\MyBuildSolutions\MySystem\MyWebAppSolution" />
<property name="MyWebApp.Host" value="${Host}/MyWebApp/MyWebApp.csproj" />
<property name="MyWebApp.Path" value="${Path}\MyWebApp\MyWebApp.csproj" />
<property name="build.config" value="DEBUG"/>
<property name="build.dir" value="${basepath}\MyWebApp\bin" />
<property name="verbose" value="true" />
<property name="fxcop.location" value="D:\Program Files\Microsoft FxCop 1.30" />
```

Target task for building the Solution:

```
<target name="build">
    <call target="getLatestFromVss"/>

    <solution outputdir="${build.dir}"
              configuration="${build.config}"
              solutionfile="${buildbase}\${solutionName}\${solutionName}.sln"
              verbose="true">

            <webmap>
                    <map url="${MyWebApp.Host}" path="${MyWebApp.Path}" />
            </webmap>
    </solution>

    <call target="runNDoc"/>
    <call target="runFxCop"/>
</target>
```

As mentioned above versioning system is not applied to the Web Application. This is because a Web Application has many different files such as Web.Config and .aspx files apart from its executable file. And since execution of Web Application is dependent on the virtual paths of these files they will have to be kept in one place.

5.5.4 Testing the scripts

Run each script by passing their name to NAnt application on command line:

D:\MyBuildScripts>nant -buildfile: MyWinAppSolution.build
D:\MyBuildScripts>nant -buildfile: MyWebAppSolution.build

5.6 Configuring Draco.NET and Draco.NET Web Viewer

5.6.1 Configure Draco.NET to use NAnt and VSS

File that requires to be edited in order to auto build is Draco.builds.config. This file can be found in C:\citools\Draco.NET Service\bin.

Following is the Draco.builds.config script used for MyWinAppSolution and MyWebAppSolution.

```
<draco xmlns="http://www.chive.com/draco">
 <pollperiod>10</pollperiod>
 <quietperiod>10</quietperiod>
 <timeoutperiod>3600</timeoutperiod>
 <rootsourcedir></rootsourcedir>
 <mailserver>mail.espusa.com.com</mailserver>
 <fromaddress>Mr.Draco@espusa.com</fromaddress>
 <builds>
     <build>
            <name>MyWinAppSolution</name>
            <pollperiod>120</pollperiod>
            <quietperiod>10</quietperiod>
            <timeoutperiod>3600</timeoutperiod>
            <notification>
                  <email>
                        <recipient>whitepapers@espusa.com</recipient>
                  </email>
                  <file>
                        <dir>D:\MyBuildSolutions</dir>
                  </file>
                  </notification>

                  <nant>
                   <buildfile>D:\MyBuildScripts\MyWinAppSolution.build</buildfile>
                  </nant>

                  <vss>
                    <project>$/MyBuildSolutions/MySystem/MyWinAppSolution/ </project>
                    <username>Admin</username>
                    <password></password>
                    <ssdir>D:\MyTestVSSDB</ssdir>
                  </vss>
            </build>

            <build>
                  <name>MyWebAppSolution</name>
                  …
                  …
                  …
            </build>
      </builds>
```

```
</draco>
```

- The key elements in the configuration file are the <build> elements, which allow Draco.NET to poll multiple Visual SourceSafe databases using different settings including the <pollperiod>.

- The <pollperiod> is assigned a global position outside the <builds> node, but it can also be set inside each <build> element.

- The <build> elements have a <nant> node that specifies the NAnt build file to use.

- On each build two types of notification are sent out by Draco.NET: one as email to all the recipients in the <email> element and secondly as an XML file that is saved to D:\MyBuildSolutions specified under <file>.

After modifying Draco.NET script, Draco.builds.config, restart Draco.NET service from Control Panel\Administrative Tools\Services applet.

Note: Every time the Draco.NET build file is modified the Draco.NET service needs to be restarted from the Control Panel\Administrative Tools\Services applet.

5.6.2 Configure Draco.NET Web Viewer

The XML build logs used by Draco.NET Web Viewer are created by Draco.NET (path of which is specified under <notification> tag in <file> tag of Draco.NET's build file). The path of these logs needs to be specified in the Web.Config file of Draco.NET web viewer.

Open Web.Config file placed in C:\citools\Draco.NET Service\DracoWebViewer. Modify the <DracoWebViewer> tag to look like this:

```
<DracoWebViewer>
    <logViewers allowGlobalSyndication="true">
            <logViewer name="MyWinAppSolution" title="MyWinAppSolution Build"
            path="D:\MyBuildSolutions" />
            <logViewer name="MyWebAppSolution" title="MyWebAppSolution Build"
            path="D:\MyBuildSolutions" />
    </logViewers>
</DracoWebViewer>
```

Draco.NET Web Viewer should now be accessible through the URL http://[server-name]/DracoWebViewer and see MyWinAppSolution and MyWebAppSolution menu on the left hand of the page.

5.7 Implementing TrueUpdate

The TrueUpdate design environment consists of two separate programs: the Client Configuration Utility, and the Server File Editor.

The Client Configuration Utility is used to customize the client-side "update.exe" program that will coordinate the update from the user's machine. The most important setting in the Client Configuration Utility is the list of Server File locations. This tells the client executable where to download the server file from.

The Server File Editor is used to edit the server-side update information file. This is where the version identifiers are defined used to identify which version of the software installed on a user's system. It's also where actions are specified to perform when a particular version of software is detected on the user's system.

This setup is not in detail because it can be a very complex setup, please reference the pdf below for more informatoin. To create the Client Configuration Utility and the Server File Editor for the Windows Application developed earlier refer to one of the following URLs:

http://www.indigorose.com/webhelp/tu10/index.htm Or
http://www.indigorose.com/files/tu10/tu10_userguide.pdf

6.0 Build Promotion Process

Every development team may have a slightly different process for promoting a build from Test through to Production, nevertheless, it is very important to have one! A Build Promotion Process (equivalent to the Release Process shown in Figure 2 and Figure 3), is simply a specific set of rules for how a version of a build of a project gets moved from one location to another for a different set of clients to access.

As stated in Section 3.5, promoting a build means copying it to the target server. The build must not be rebuilt but rather moved. Therefore there is the need to identify and archive builds in a build promotion process (Section 6.0 Build Promotion Process) as builds are created.

A level of complexity is added with the requirement for a Build Manager to not only promote (move) the build, but tentatively manage the movement of multiple projects, for multiple clients, Test, QA, and Production, for TrueUpdate delivery (See Section 5.7 Implementing TrueUpdate).

Section 5.5.2 demonstrates how ESP determines version build numbers at the time of writing this document. The following diagram is an overview for implementing TrueUpdate to work with three separate application clients for Test, QA, and Production:

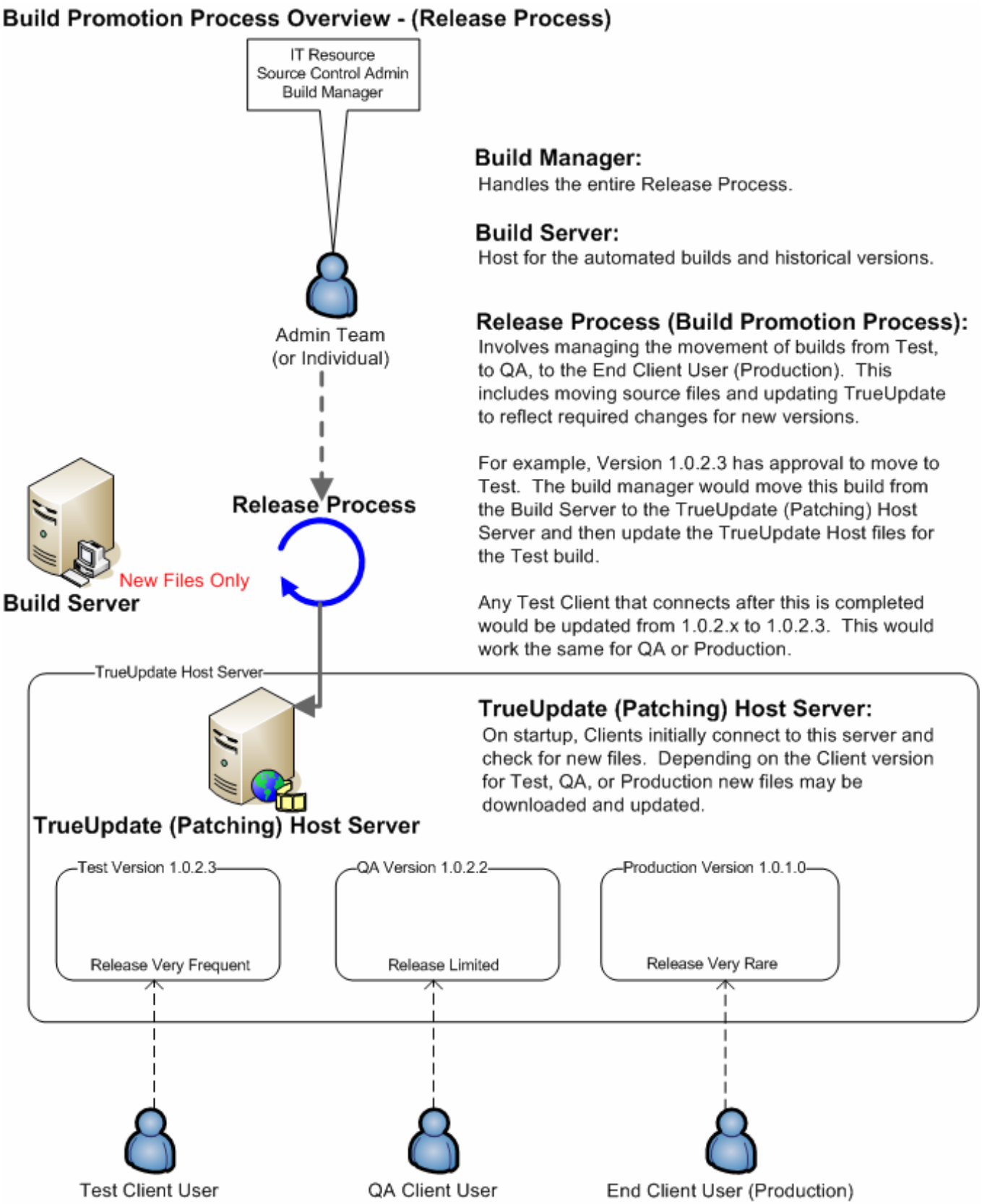

Figure 7 – Build Promotion Process Overview - (Release Process)

7.0 Future Concept

Future concepts are theorized and researched, but has not yet used in practiced.

7.1 Transactional Builds

Transactional Builds involves the process before Build Promotion.

In database systems the concept of a transaction guarantees that a set of actions are atomic, which means that they either all succeed or they all fail. In CI, this concept of transactions is not yet realized for multi developer environments.

The basic concept involves a smart application framework tracking an unlimited number of developers on a single project as well as multiple resources that a code base interfaces with (Web Services, Databases, and Config Files). At the same time a project would have the ability to dynamically roll back all code to a previous successful build until the latest is fixed.

William C. Wake's article below discusses concepts in transactional continuous integration; what is missing from these concepts is an application framework or guide to a corresponding proven solution.

http://www.xp123.com/xplor/xp0203a/index.shtml

In the mean time it is possible to "fake out" transactional builds through simply having Build Promotion process defined tightly to move the most recent successful versioned build directory to the Test area for Auto Updating.

8.0 References and Suggested Reading

8.1 Web References for CI Tools and Articles

8.1.1 Continuous Integration

- Continuous Integration
  http://www.martinfowler.com/articles/continuousIntegration.html

- Team Development with Visual Studio .NET and Visual SourceSafe
  http://msdn.microsoft.com/library/en-us/dnbda/html/tdlg_rm.asp

- Continuous Integration with Cruise Control .NET and Draco.NET
  http://www.theserverside.net/articles/showarticle.tss?id=ContinuousIntegration

8.1.2 NANT

- http://nant.sourceforge.net

- http://nant.sourceforge.net/help/index.html

- Automating Your .NET Development With NAnt
  http://www.informit.com/articles/printerfriendly.asp?p=30076

- More NUnit and NAnt Tricks, Tips and Examples
  http://www.informit.com/articles/printerfriendly.asp?p=30335

8.1.3 NAntContrib

- http://nantcontrib.sourceforge.net/help/tasks/index.html

- http://nantcontrib.sourceforge.net

8.1.4 Draco.NET

- http://sourceforge.net/projects/draconet

- http://draconet.sourceforge.net/wiki

8.1.5 Draco.NET Web Viewer

- http://blogs.biasecurities.com/jim/archive/2004/02/16/337.aspx

8.1.6 NDoc

- http://ndoc.sourceforge.net

8.1.7 NUnit

- http://nunit.sourceforge.net

- http://nunit.org

8.1.8 NUnitForms

- http://sourceforge.net/projects/nunitforms

8.1.9 FxCop

- http://www.gotdotnet.com/team/fxcop

- Microsoft .NET Framework Design Guidelines
  http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/cpconnetframeworkdesignguidelines.asp

8.1.10 CruiseControl.NET

- http://cruisecontrol.sourceforge.net
- http://confluence.public.thoughtworks.org/dashboard.action

8.1.11 Build Promotion

- How do you handle your build/promotion process?
  http://weblogs.asp.net/davebost/archive/2004/06/04/148806.aspx
  http://weblogs.asp.net/davebost/archive/2004/07/08/176022.aspx

- Implementing and Promoting Daily Builds
  http://www.15seconds.com/issue/040810.htm

8.1.12 Functional Testing

- http://www-106.ibm.com/developerworks/library/j-test.html

8.1.13 Dynamic Software Update and Patch System

- TrueUpdate
  http://www.indigorose.com/tu/index.php

- Tools: Patches & Updates

- http://www.installsite.org/pages/en/tt_patch.htm

- Automatically Upgrade Your .NET Applications On-the-Fly
  http://www.devx.com/dotnet/Article/10045

- Write Auto-Updating Apps with .NET and the Background Intelligent Transfer Service API
  http://msdn.microsoft.com/msdnmag/issues/03/02/BITS/default.aspx

8.2 Other References

8.2.1 Visio

- http://office.microsoft.com/en-us/FX010857981033.aspx

8.2.2 Coding Techniques

- Joel on Software - The Joel Test 12 Steps to Better Code
  http://www.joelonsoftware.com/articles/fog0000000043.html

- Extreme Programming Practices in Action
  http://www.informit.com/articles/printerfriendly.asp?p=30187

- Building Software the Extreme Programming Way
  http://www.informit.com/articles/printerfriendly.asp?p=30295

- Web Development Model
  http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/tdlg_ch2.asp

8.3 Books

8.3.1 Open Source .NET Development: Programming with NAnt, NUnit, NDoc, and More

http://www.amazon.com/exec/obidos/tg/detail/-/0321228103/002-1292323-7959221?v=glance

8.3.2 Pragmatic Unit Testing in C# with NUnit (Pragmatic Programmers)

http://www.amazon.com/exec/obidos/tg/detail/-/0974514020/ref=pd_sim_books_3/002-1292323-7959221?v=glance&s=books

8.4 Suggested Readings

8.4.1 Draco.NET vs. CruiseControl.NET

A nice comparison of these two tools is given at the end of this article:
http://www.theserverside.net/articles/showarticle.tss?id=ContinuousIntegration.

Draco.NET has a similar dashboard.

8.4.2 A web viewer for Draco.NET output

http://blogs.biasecurities.com/jim/archive/2004/02/16/337.aspx.
It shows same content as the email sent out by Draco.NET for each build and an archive of all outputs in one place.

9.0 Known Issues / Bug Appendix

9.1 NAnt Version 0.84 Issue

1. *The latest stable build of NANT 0.84 has an issue while compiling Web Application (Web Services). Following error is shown while build is performed on a Web Application:*

   "Error checking whether 'http://localhost/webproject1/webproject1.vbproj' *is an enterprise template project"*

   After some research on the internet I found that this issue is fixed in the build 0.85 nightly build.

   Ref: http://www.mail-archive.com/nant-users@lists.sourceforge.net/msg04645.html

   Used the latest version of NAnt (0.85 – nightly build) for compiling Web Application and web service. Both work fine now.