Continuous Integration Hudson

SENG 403
Tutorial 4
Department of Computer Science
University of Calgary

Agenda

- What is Continuous Integration?
- The benefits of CI
- CI Practices
- Using Hudson as a CI server

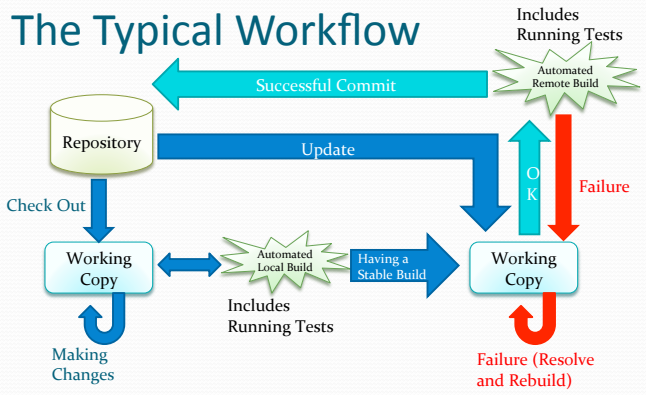
The Integration Dilemma

- If separate groups are working on different parts of a project for a long time, integrating their work will become a nightmare!
  - It could take months or even years!
- Solution: Every one should integrate their work frequently.
  - Any individual developer's work is only a few hours away from a shared project state and can be integrated back into that state in minutes.

So, What is Continuous Integration, Anyway?

- The basic idea is that every developer on the team integrating frequently, usually daily, against a VCS.
  - Wait a minute! Isn't it what we practiced in the first tutorial? *Not Exactly*!
- It is not just about committing changes into the shared repository. It also deals with making sure that the project stays in a stable state.

The Typical Workflow



Why commit frequently?

- If there is a clash between two developers, frequent commits will reveal it when the second developer commits.
- The error will be detected rapidly.
- The most important thing is to fix it ASAP.
- CI is about communication. Developers find inconsistencies and fix them rapidly.
- It guarantees that the latest version of software in the repository stays stable. The working copies will not deviate from it dramatically and integrating them back does not need too much effort.

Practices of Continuous Integration (1)

Maintain a Single Source of Repository

- We should use a Version Control System, like SVN.
- It will contain the source code as well as everything we need to build it, like Ant scripts, DB schemas, test scripts, etc.
- Rule of thumb: You should be able to check out from the repo and build it on a virgin machine.

Practices of Continuous Integration (2)

Automate the Build

- Turning the sources into running systems can be a complicated process. However it can, and should, be automated.
- Use automated environments like Ant, Nant, make, MSBuild, etc.
- We can use IDE built-in build features, but we should not rely on them for the automated build process on the CI server.

Practices of Continuous Integration (3)

Make Your Build Self-Testing

- By compiling and linking the source code, some errors can be found, but most bugs are undetected.
- To catch lots of bugs, we need a suite of automated tests that can check a large part of the source code.
- Do not forget: Good tests can catch bugs, but they do not prove the absence of bugs.

Practices of Continuous Integration (4)

Everyone Commits to the Mainline Every Day

- By committing frequently, we will tell other developers about the changes we've made.
- When we want to commit, we should first update. It will reveal compilation errors. By running tests, we will know about bugs as well.
- The sooner we spot conflicts, the easier is to fix them.
- Rule of thumb: Commit to the repo every day, even multiple times a day.

Practices of Continuous Integration (5)

Every Commit Should Build the Mainline on an Integration Machine

- A commit is considered to be done, if it builds successfully on an integration server.
- It could be either
  - Manual: The build is started by the developer.
  - Automatic: Every time a commit against the repo finishes, the CI server automatically checks out the head and initiates the build process.
- Some organizations do regular scheduled builds.
- The mainline should be kept in a healthy state. If it breaks, it must be fixed right away.

Practices of Continuous Integration (6)

Keep the Build Fast

- The whole point of Continuous Integration is to provide rapid feedback.
- The XP guideline says that builds should finish in ten minutes.
- If a short build is not possible, a staged build (aka build pipeline) can be used.

Practices of Continuous Integration (7)

Test in a Clone of the Production Environment

- The point of testing is to show any problem that the system will have in production.
- So, we should run tests in an environment similar to the production environment.
- Not always possible,
- Consider using Virtualization.

Practices of Continuous Integration (8)

Make it Easy for Anyone to Get the Latest Executable

- Customer will tell you what they want, when they see a running system. :P
- Any team member should be able to get the latest version of the executable.
- Hint: Make sure there's a well known place where people can find the latest executable.

Practices of Continuous Integration (9)

Everyone can see what's happening

- Continuous Integration is all about communication.
- The most important thing to communicate: The state of the mainline build
- Some teams even go further by hooking up the CI display to some fancy lights.
- The CI website is advantages for distributed teams.

Practices of Continuous Integration (10)

Automate Deployment

- We might have multiple machines for the primary and secondary builds. We need to transfer executables between these machines.
- So, the deployment should be automated.
- The consequence is that you should also have scripts that allow you to deploy into production with similar ease.

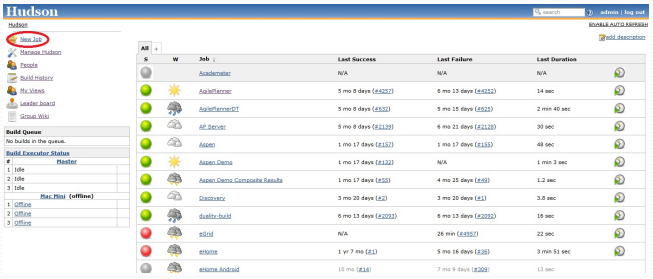
Hudson

- Hudson monitors source control repositories. When changes are committed, Hudson can:
  - Execute automated builds on various platforms
  - Run automated tests
- It supports different VCSs, like SVN, CVS, and git.
- It's expandable by adding plugins.
- It reports the results of automated test run
- Build results can be monitored on the web console or pushed to users via RSS, Email, and IM.

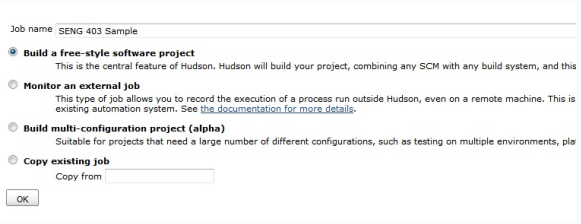
Setting Up a Build in Hudson

- Before we set up a build job in Hudson, the following conditions must be met:
  - We must have an accessible source code repository.
  - The repository must contain the source code we want to build.
  - The repository must contain build scripts that will build the source. These are usually Ant or Maven scripts, although Hudson also supports simple shell scripts, NAnt, and MSBuild.

Create A New Job

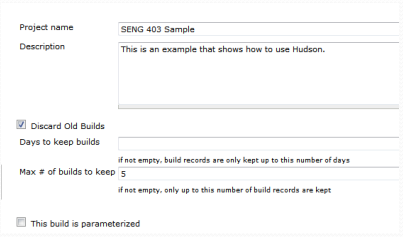


New Job (1 of 6)

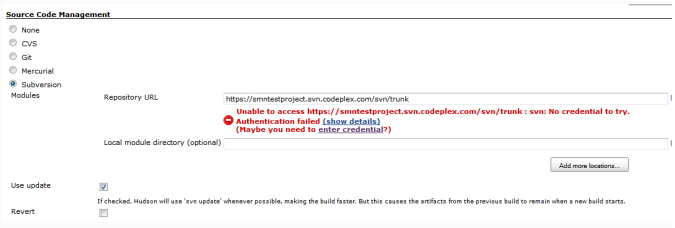


New Job (2 of 6)

- Provide a description
- If we do not check the "Discard Old Builds," Hudson will keep records of all the previous builds.

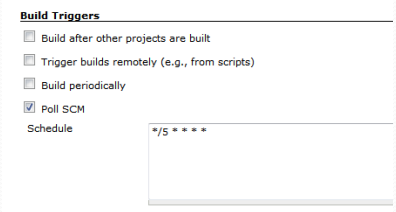


New Job (3 of 6)



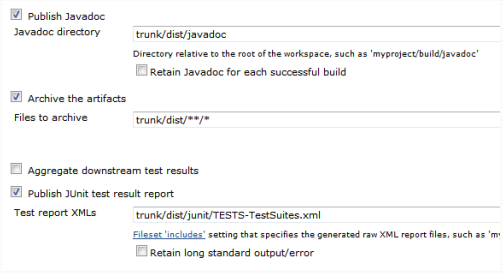
New Job (4 of 6)

- We ask Hudson to poll the VCS (or SCM), in this case SVN, every 5 minutes and start the build if there has been a commit.



New Job (5 of 6)

- Some of the post-build Actions



New Job (6 of 6)

- We can provide a list of email recipients to receive an email every time a build fails.
- It is possible to run other tools to get us some metrics about the project, like code coverage.

Defining a simple Ant script

```
<project name="MyProject" default="all" basedir=".">

    <property name="src" value="./src" />
    <property name="lib" value="./lib" />
    <property name="dist" value="./dist" />
    <property name="classes" value="${dist}/classes" />
    <property name="jar" value="${dist}/jar" />
    <property name="javadoc" value="${dist}/javadoc" />
    <property name="junit" value="${dist}/junit" />
    <property name="test.class.name" value="smn.MyTestSuite" />

    <path id="test.classpath">
        <pathelement location="${classes}" />
        <pathelement location="${lib}/junit-4.8.2.jar" />
        <fileset dir="${lib}">
        <include name="**/*.jar"/>
        </fileset>
    </path>

    <target name="clean">
        <delete dir="${dist}"/>
    </target>

    <target name="compile" depends="clean">
        <mkdir dir="${dist}"/>
        <mkdir dir="${classes}"/>
        <javac srcdir="${src}" destdir="${classes}">
        <classpath>
            <pathelement path="${classpath}"/>
            <fileset dir="${lib}">
                <include name="**/*.jar"/>
            </fileset>
        </classpath>
        </javac>
```


The Source Files

- HelloWorld.java! ;)

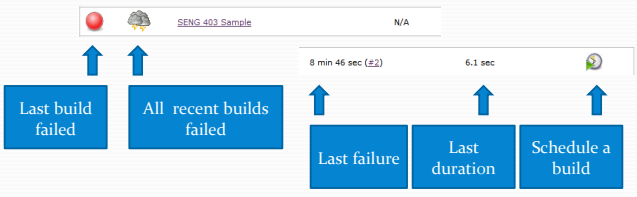
```
package smn;

public class Arithmetic {
    private int a;
    private int b;

    public Arithmetic(int a, int b){
        this.a = a;
        this.b = b;
    }

    public int add(){
        return a+b;
    }

    public int multiply(){
        return a*b;
    }

    public static void main(String[] args){
        System.out.println("100 + 35 = "+(new Arithmetic(100, 35)).add());
    }
}
```


The Test File

```
package smn;

import org.junit.Assert;
import org.junit.Test;

public class TestArithmetic {

    @Test
    public void adding(){
        int x = 10, y = 20;
        Arithmetic a = new Arithmetic(x, y);
        Assert.assertEquals(x+y, a.add());
    }

    @Test
    public void multiplying(){
        int x = 10, y = 20;
        Arithmetic a = new Arithmetic(x, y);
        Assert.assertEquals(x*y, a.multiply());
    }
}
```


Build Status

- The build starts after 5 minutes.
- And it fails! ☹
- The status of builds are shown in the Dashbord.



Project Home

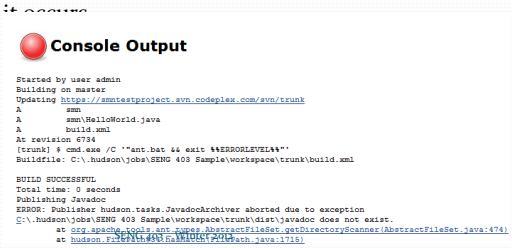
- To see what went wrong, we should go to the project page.

Changes

- By clicking on changes and then on details we can see what has changed in the last commit.

Console Output

- If the job is complete this will display the static output that was generated by the build script; you can click ENABLE AUTO REFRESH to make Hudson periodically refresh the content of the page so that you can see output as it occurs.



Start Build

- We can start a build immediately by clicking on the "Build Now" link in the Dashboard page.
- The left table labelled Build Queue will display jobs currently running.

Hudson Plugins

- Hudson functionality can be extended by installing plugins
- **SCM**: Plugins that implement Hudson support for source control systems other than CVS and Subversion.
- **Build tools**: Plugins that implement additional build tools, such as MSBuild and Rake. These are particularly useful if you would like to build non-Java software in Hudson.

Hudson Plugins

- **Build notifiers**: These plugins supply alternate ways of issuing notifications about job events -- via Twitter, IRC, Google Calendar events, and the like.
- **Build reports**: A series of plugins that create useful reports based on some form of analysis of your source code or generated artifacts. For example, the Cobertura plugin aggregates ongoing coverage reports generated by your build scripts.
- **External site integrations**: Plugins that assist in integrating Hudson with other applications, such as Jira or Bugzilla.

Job Stability

- **Job state**: Figure 27 outlines the symbols for the four possible states for the most recently executed build of a job:
  - *Successful*: The build completed and was considered stable.
  - *Unstable*: The build completed and was considered unstable.
  - *Failed*: The build failed.
  - *Disabled*: The job is disabled.

Figure 27. Job states

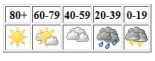
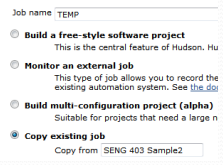


Figure 28. Job stability

- **Job stability**: While a job may build to completion and generate the target artifacts without issue, Hudson will assign a stability score to the build (from 0-100) based on the post-processor tasks, implemented as plugins, that you have set up to implicitly evaluate stability.

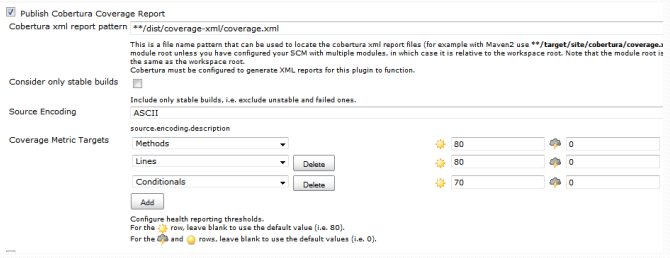
A Plugin Example

- We want to use Code Coverage metrics.
- One plugin that does this Cobertura.
- It is free, but we need to add some stuff into the Ant script.
- To add a Cobertura report as a post build artifact, we need to create a new job.
- We can copy an existing job and change the properties.



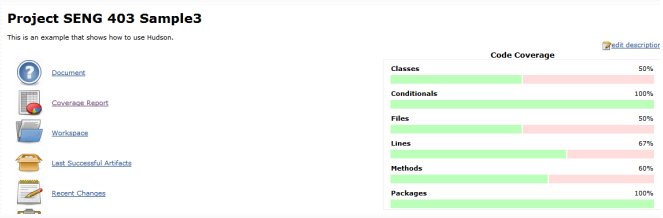
Using Cobertura

- We should check the appropriate checkboxes in the "define a new job" screen.



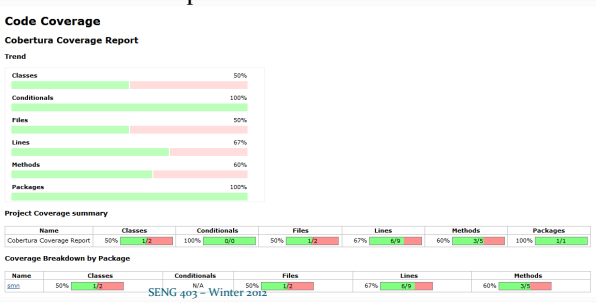
Code Coverage Reports

- Now when we build the project we can see some reports in the project home.



Code Coverage Reports

- By clicking on the "Coverage Report" link we can see a more detailed report.



References and Further Readings

- The classic article on Continuous Integration by Martin Fowler: http://www.martinfowler.com/articles/continuousIntegration.html
- The CI book: http://www.amazon.com/Continuous-Integration-Improving-Software-Reducing/dp/0321336380/ref=sr_1_2?ie=UTF8&qid=1296008075&sr=8-2
- Continuous integration with Hudson: http://www.javaworld.com/javaworld/jw-12-2008/jw-12-hudson-ci.html?page=1