Continuous (Unit) Testing

- Test your code *early*, *automatically* and *repeatedly*.
  - To maximize the benefits of unit testing.

- Early testing
  - You as a programmer do coding and unit testing at the same time.
- Automated testing
  - Run ALL test cases in an automated way.
    - Never think of selecting and running test cases by hand.
- Repeated testing
  - Run ALL test cases whenever changes are made in the code base.

Benefits of Continuous Testing

- Performing *regression tests* through continuous unit tests
  - Regression
    - A bug that emerges as a by-product in making changes in the code base
      - e.g., adding new code to the code base or revising existing code in the code base.
  - Regression testing
    - Uncovering regressions after changes are made in the code base
  - Seamlessly integrate unit testing and regression testing

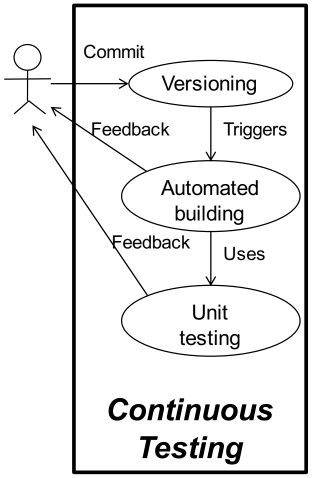
- *Immediate*ly getting feedback on regressions to development project members and fix them.
  - DO: Code → unit test → small regression fixes → unit test
  - DON'T: Code → code → code → big regression fixes
    - The amount of regressions (and the cost to fix them) can exponentially increase as time goes without continuous testing.

3 Key Elements for Continuous Testing

- Unit testing
  - Regression testing through repeated unit testing

- Automated building
  - Automate the *entire* build process
    - e.g., compilation of all source code, unit testing, coverage measurement, other testing (if any), packaging, deployment and Javadoc generation.

- Versioning
  - When a regression is found upon a change in the code base, you need to know the difference/delta in b/w the current version and a previous version that has passed all unit tests.
    - The regression comes out from that delta.



Automated Building

- Build tools: Ant, Maven, Gradle, etc.

- Ant
  - Basic build tool
    - Sufficient for solo projects and small-scale team projects.
    - Need to write up a project specification (build.xml) from scratch
    - Need to download all external libraries and set them up (e.g., specify the build paths and environmental variables for those libraries).
      - Need to be careful about library-to-library depenency
      - It could take a half day or even a whole day to just set up your development environment.

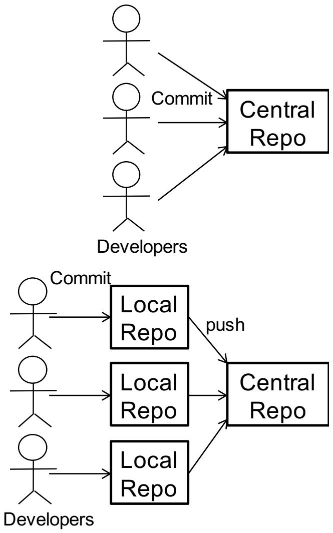
- Maven
  - Provides basic Ant-like features, plus extra useful features
    - Project templates
      - No need to always write up a project specification from scratch
    - Library management
      - No need to download and set up external libraries. Just specify the libraries you want to use. They will be automatically downloaded.
        - No need to download a library when it's newer version is available. Just specify a version number you want to use.
      - No need to check library-to-library dependency. Required libraries will be automatically downloaded.
    - It can be very quick to set up your development environment.
    - A large number of plug-ins are available.
      - Good extensibility. Easy to use plug-ins.
      - Not easy to modify existing plug-ins. Not that straightforward to make a new ones.
    - Integration with JUnit, JaCoCo, Covertura, etc.
    - Concurrent/parallel builds
    - Potential death by XML (POM: Project Object Model)!
      - Need a POM meister in a large-scale project

- Gradle
  - Provide basic Ant-like features, plus extra useful features
  - Library management
  - Groovy-based project specifications
    - You can program whatever you want to do in Groovy.
      - No need to find necessary plug-ins.

```
» task HelloWorld{
    doLast{
      println 'Hello World'
    }
  }
```

Versioning w/ Version Ctrl Systems

- Fully centralized: CVS and Subversion
  - Commit code to the central repository.
  - *Fine-grained* (e.g., per-class or per-method) commits are required to avoid potential conflicts among different commits by different developers
  - Unit/regression testing per commit
  - Too many conflicts could occur in each commit.
- Distributed: Git and Mercurial
  - Commit code to the local repository first and then push it to the central repository
  - *Coarse-grained* (e.g., per-feature/functionality or per-use-case) pushes
  - Unit/regression testing per commit to the local repo and per push to the central repo



Code Inspection (Static Code Analysis)

- Analyzing code without actually executing it.
  - Dynamic analysis: analysis performed by executing code
- Static code analyzers
  - Can automatically detect particular (bad) code smells
  - Various tools available for various languages
    - http://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis

Static Code Analyzer: FindBugs

- Looks for bugs in Java programs based on bug patterns, which are code idioms that are often error.
  - http://findbugs.sourceforge.net/
  - http://findbugs.sourceforge.net/bugDescriptions.html
  - Can analyze code compiled for any version of Java, from 1.0 to 1.8
  - Can be used from Ant and Eclipse
  - Plug-ins are available for IntelliJ, Maven, Gradle, Jenkins, etc.

Static Code Analyzer: PMD

- Looks for potential problems such as:
  - Possible bugs
    - Empty try/catch/finally/switch statements
  - Dead code
    - Unused local variables, parameters and private methods
  - Suboptimal code
    - Wasteful String/StringBuffer usage
  - Overcomplicated expressions
    - Unnecessary if statements, for loops that could be while loops
  - Duplicate code
    - Copied/pasted code means copied/pasted bugs
  - http://pmd.sourceforge.net/
  - Requires JRE 1.6 or higher to run
  - Can be used from Ant
  - Plug-ins are available for Maven, Eclipse, etc.

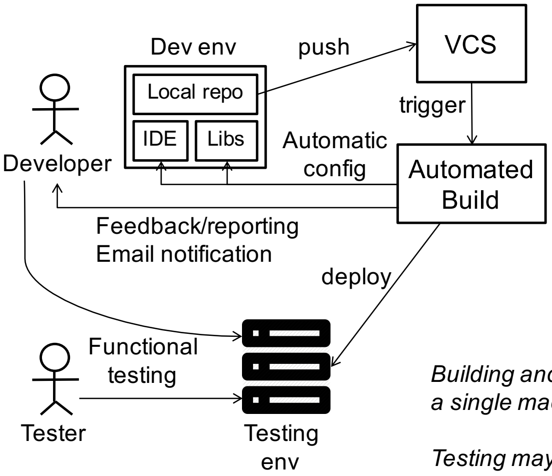
Other Static Code Analyzers

- CheckStyle
  - Checks coding conventions
    - http://checkstyle.sourceforge.net/

HW 7-1

- Run FindBugs on your HW 6-1 and 6-2 programs
- Write build.xml to run JUnit and FindBugs
  - Extra points if you run PMD as well as FindBugs.
  - Required to run FindBugs only in this HW
    - Extra point if you run FindBugs in other HWs in the future.

Continuous Testing



Compilation
Unit testing
Coverage measurement
Inspection (static code analysis)
Packaging
Deployment
Document (Javadoc) generation

Building and testing may be carried out on a single machine.

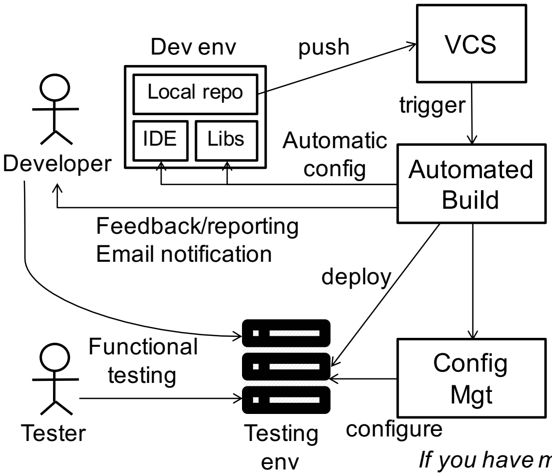
Testing may require more than one machines in network systems and multi-platform systems.

Automated Deployment

- Fabric
  - http://www.fabfile.org/
- Capistrano
  - http://capistranorb.com/
- Rundeck
  - http://rundeck.org/

- jclouds
  - https://jclouds.apache.org/
  - VM management for major clouds
    - Amazon EC2, Azure, OpenStack, etc.
  - Automates staring multiple VMs at once and installing software on them.

Continuous Testing w/ Config Mgt



Compilation
Unit testing
Coverage measurement
Inspection (static code analysis)
Packaging
Deployment
Document (Javadoc) generation

If you have many machines for testing, it would be time-consuming and error-prone to manually configure the machines with particular settings.

A configuration mgt tool can be useful.

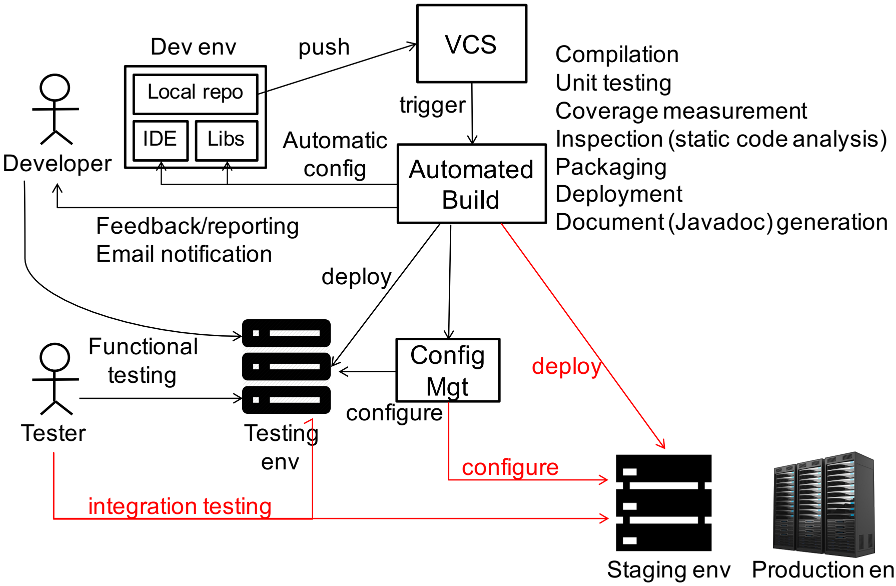
Configuration Mgt Tools

- a.k.a. System management tools

- Automate system configurations (e.g., OS settings, app installations) for given machines.
  - May apply the identical configurations to all of the machines
  - May apply different configurations to different machines.

- Open-source tools
  - Puppet
    - http://puppetlabs.com/
    - Jason-like DSL (domain specific language)
  - Chef
    - https://www.getchef.com/
    - Ruby-based DSL

Continuous Integration (CI)



Dev env
Local repo
IDE
Libs
Developer
push
VCS
trigger
Automatic config
Automated Build
Compilation
Unit testing
Coverage measurement
Inspection (static code analysis)
Packaging
Deployment
Document (Javadoc) generation
Feedback/reporting
Email notification
deploy
Functional testing
Tester
Testing env
Config Mgt
configure
deploy
configure
integration testing
Staging env
Production env

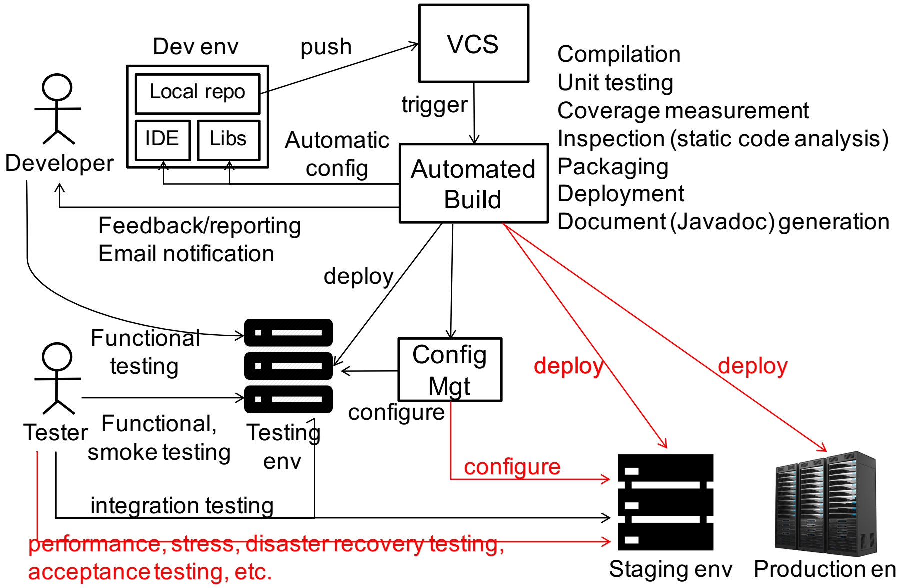
Integration Testing

- Testing the integration of multiple modules/packages.
  - e.g. An output from a module goes to the other module correctly?
  - A module's task starts after the other's task is completed?
- GUI testing
  - e.g., Selenium
    - Automates Web browsing and Web app operations
      - » Page loading, page transitions, data entries, etc.
    - Test cases can be written in HTML, Java, Python, Ruby, C#, etc.
    - Selenium IDE can partially generate test cases.
  - e.g., Android SDK testing framework
    - Automates user activities on the Android emulator
      - » Clicking and other finger gestures, data entries, etc.
- Request-response testing for web apps
  - e.g. AsyncHttpClient

Jenkins: a CI Tool

- Jenkins (http://jenkins-ci.org/)
  - Implemented as a Servlet app
    - Cron-like (timer-based) periodic build and integration testing
  - Advanced GUI-based reporting
  - Not only for Java-based development
    - C/C++, Ruby, Python, JavaScript, etc.
  - Automated process for continuous testing
    - Build with Ant, Maven, Gradle, etc.
    - Unit testing with Junit
    - Coverage measurement with JaCoCo, Cobertura, etc.
    - Code inspection with Findbugs, PMD, CheckStyle, Coverity, etc.
    - App deployment to Tomcat, other Servlet engines, clouds
  - Automated integration testing with Selenium, etc.

Continuous Delivery



Dev env
Local repo
IDE
Libs
Developer
push
VCS
trigger
Automatic config
Automated Build
Compilation
Unit testing
Coverage measurement
Inspection (static code analysis)
Packaging
Deployment
Document (Javadoc) generation
Feedback/reporting
Email notification
deploy
Functional testing
Tester
Functional, smoke testing
Testing env
Config Mgt
configure
deploy
deploy
configure
integration testing
performance, stress, disaster recovery testing, acceptance testing, etc.
Staging env
Production env

- Automated release/delivery
  - Release any versions of the product
    - Testers can verify the changes between different versions
    - Support engineers can deploy the product on a testing/staging environment and see if a reported defect is reprodusible.
    - System operators can deploy the product on an operational environment to perform disaster recovery testing, etc.
    - Customers can be satisfied
  - No more manual procedures for releases
    - Procedure documents
      - Copy this step-by-step time consuming
  - Can rollback to a previous version immediately when critical issues are found in the current version.

HW 7-2

- J. Spolsky, "The Joel Test: 12 Steps to Better Code," In Joel on Software, Chapter 3, Apress, 2004.
  - http://www.joelonsoftware.com/articles/fog0000000043.html
- J. Spolsky, "Daily Builds are your friends"
  - http://www.joelonsoftware.com/articles/fog0000000023.html
- J. Spolsky, "The Joel Test: 12 Steps to Better Code"
  - http://www.joelonsoftware.com/articles/fog0000000043.html

In CS 680/1 and CS 682/3

- In CS 680/1
  - Ant is sufficient.
  - No need to use any VCS
    - Small-scale, solo work
- In CS 682/3
  - Ant? Gradle??
  - Use a VCS: Subversion? Git?