CS314 Software Engineering
Configuration Management

Dave Matthews

Configuration Management
Configuration Management

- Management of an evolving system in a **controlled** way.
  - **Version control** tracks component changes as they happen.
  - **System Building** assembles components for testing and release. Frequently is better.
  - **Change Management** addresses stakeholder proposals.
  - **Release Management** plans and prepares for distribution.

Version Management

- Two models
  - **Centralized** – master repository maintains all versions (SVN)
  - **Distributed** – multiple copies exist at the same time (Git)
- Features of both
  - Version and release identification
  - Change history recording
  - Independent Development
  - Project Support
  - Storage Management
Distributed Model Benefits

• A backup mechanism for the master repository.
• Allows developers to work offline
  – commit changes without a network connection
  – Developers can compile and test locally.

System Building

• Build script generation (configuration file)
• Build system integration with version control system
• Minimal recompilation (what changed or affected)
• Executable system creation
• Test automation (check build not broken by changes)
• Report success or failure of build and test
• Documentation (release notes) automatically generated
Change Management

• Ensure changes are applied in a controlled way.
  – requirements, bugs, …
• Consider factors in decisions to changes
  – Consequences
  – Benefits
  – Number of users affected
  – Cost
  – Product release cycle

Release Management

• Plan the release
• Prepare the system for release
  – Configuration files
  – Data files
  – Installation program
  – Electronic and paper documentation
  – Packaging and associated publicity
• Document the release
GitHub / Git

https://github.com
https://help.github.com/

GitHub (server)
- On the internet, holds the master repo, issues, releases, ...
- Code on master should always build/test/run with no problems
- No changes are made directly in master
- Proposed changes are reviewed, approved, then merged

Git (client)
- On a remote/local machine
- Holds a clone of the master
- Changes made in branches pushed back to the GitHub master and merged via a pull request
### Using GitHub and Git

**CREATE A BRANCH**
Create a branch in your project where you can safely experiment and make changes.

**Git**
- Select, estimate, and assign a pending issue issue.
- Refresh (pull) or clone repo if needed. Create and checkout a new local branch in your repo clone.
- Modify and add files and directories in the repo. Build and test before you commit.
- Add your changes to the branch. Commit the branch with the #issue. Push branch to master.
- Open a pull request for the commit for review by others.
- Address any merge conflicts or comments. Merge the pull request and confirm.

**GitHub**
- Open a pull request for review by others.
- Address any merge conflicts or comments.
- Merge the pull request and confirm.

**DISCUSS AND REVIEW**
Use a pull request to get feedback on your changes from people down the hall or ten time zones away.

**MERGE AND DEPLOY**
Merge your changes into your master branch and deploy your code.

### Git Commands - local setup

```
# install git on your local system
# configure git username, email

git config --global user.name "[firstname lastname]"
git config --global user.email "[valid-email]"

# clone your team repo or the class repo

git clone [masterURL]
```
Git Commands - start a new branch

# update your local copy before you start

git pull origin master

git branch [newbranchname]

git checkout [newbranchname] # never master!

git branch # verify just to be sure

Git Commands - commit and push

# create, edit, rename, move, or delete files under “.”.
# build and test to verify changes work

git add . # add all changes to branch

git status # verify proposed changes are listed

git commit –m “closes #999”

git push origin [branchname]
Git Commands - merge conflicts

# find files with merge conflicts on your local repo (so you can build and test)
git status # list files with merge conflicts

# edit files to resolve the conflicts between <<<<<<< HEAD and >>>>>>> BRANCH-NAME
# build and test

git add .
git commit -m "resolved merge conflict by ..."
git push origin [branchname]

GitHub Etiquette

• No changes made directly to master branch, never checkout master.
• All changes made in local/separate branches and merged via pull requests.
• All pull requests associated with an issue.
• Never break master. It should always build/test/run successfully.
Java Packages

- Hierarchical namespace that organizes related classes and interfaces
  - easier to find and use
  - avoid naming conflicts
  - control access
- The Java platform provides enormous class library known as the Application Programming Interface (API)
- `java.lang.Object`
  - `java.text.Annotation`
  - `java.text.AttributedCharacterIterator.Attribute` (implements `java.io.Serializable`)
    - `java.text.Field`
      - `java.text.SimpleDateFormat`
      - `java.text.MessageFormat.Field`
      - `java.text.NumberFormat.Field`
    - `java.textAttributedString`
  - `java.text.Bidi`
  - `java.text.Breakiterator` (implements `java.lang.Cloneable`)
  - `java.text.CollationElementIterator`
  - `java.text.CollationKey` (implements `java.lang.Comparable<T>`)
  - `java.text.Collator` (implements `java.lang.Cloneable`, `java.util.Comparator<T>`)
    - `java.text.RuleBasedCollator`
  - `java.text.DecimalFormatSymbols` (implements `java.lang.Cloneable`, `java.io.Serializable`)
  - `java.text.FieldPosition`
  - `java.text.Format` (implements `java.lang.Cloneable`, `java.io.Serializable`)
    - `java.text.SimpleDateFormat`
    - `java.text.MessageFormat`
    - `java.text.NumberFormat`
      - `java.text.ChoiceFormat`
      - `java.text.DecimalFormat`
  - `java.text.Normalizer`
  - `java.text.ParseException`
  - `java.text.StringCharacterIterator` (implements `java.text.CharacterIterator`)
  - `java.lang.Throwable` (implements `java.io.Serializable`)
    - `java.lang.Exception`
      - `java.text.ParseException`
Maven Objectives

- allow developers to understand the complete state of a development effort in the shortest period of time
  - make the build process easy (shields from details)
  - provide a uniform build system (object model, plugins)
  - provide quality project information
  - provide guidelines for best practices development
  - allow transparent migration to new features

Maven – some key features

- simple, consistent setup
- dependency management, central repository
- large repository of libraries and latest releases
- model based builds, no scripting
- generate web site or PDF with project information
- release management and distribution
Maven in CS 314

- pom.xml for local and Continuous Integration build/test
- mvn commands for local build/test
- Travis CI for master build/test

Packages and Maven
TripCo packages

TripCo.java
package []; import [];

Model.java, View.Java
package []; import [];

TestTripoCo.java
TestModel.java
TestView.Java
package []; import [];

TripCo - Maven configuration

TripCo.java /src/main/java/edu/csu2017fa314/t99/
package []; import [];

Model.java, View.Java /src/main/java/edu/csu2017fa314/t99/Model/
package []; import [];

TestTripoCo.java /src/main/java/edu/csu2017fa314/t99/view/
TestModel.java /src/main/resources/
TestView.Java /src/test/java/edu/csu2017fa314/t99/
package []; import [];
/web/
data/
TripCo

TripCo.java
package edu.csu2017fa314.T99;

Model.java
import [];

View.Java
import [];

TestTripCo.java
TestModel.java
TestView.Java
package [];
import [];

Where do I put my files?

TripCo.java
package edu.csu2017fa314.T99;

Model.java
import [];

View.Java
import [];

Destination.java
import [];

Itinerary.java
import [];