Building Software Systems

Idaho State University | Computer Science

Dr. Isaac Griffith

CS 2263
Department of Computer Science
Idaho State University

ROAR

Outcomes

After today's lecture you will be able to:

- Understand why we use build and dependency management tools
- Understand the general process of software build
- Understand the general idea of dependency management
- Understand a brief history of these tools

5 Facts I Know

Take out a sheet of paper or open a text editor, pause the lecture and complete the following:

1. At the top of the page write the topic: **Building Software**
2. Write the numbers 1 - 5 along the left column of the sheet
3. Brainstorm to identify 5 facts that you know about this topic
   - If you are having a hard time, consult the reading material
   - If you are still having a hard time, use the internet.

Thought Experiment

Consider Bob

Bob

IDE: Netbeans
OS: Windows
Compiler: Oracle JDK

Build: IDE Default
Deps: Manual

Thought Experiment

Consider Bob

Bob

IDE: Netbeans
OS: Windows
Compiler: Oracle JDK

Build: IDE Default
Deps: Manual

Is this a familiar approach?

Thought Experiment

Consider Bob

Bob

IDE: Netbeans
OS: Windows
Compiler: Oracle JDK

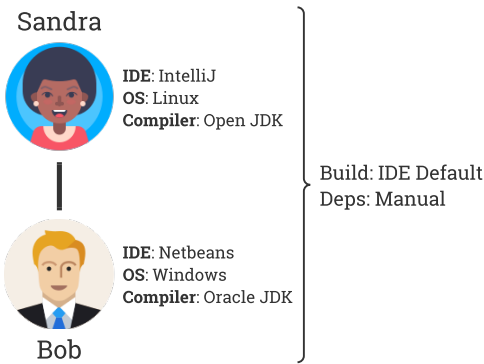
Build: IDE Default
Deps: Manual

Is this a familiar approach?

Is this a good approach?

Thought Experiment

Sandra Joins the Project...



- What can go wrong here?

Thought Experiment

Sandra Joins the Project...

Sandra

IDE: IntelliJ
OS: Linux
Compiler: Open JDK

IDE: Netbeans
OS: Windows
Compiler: Oracle JDK

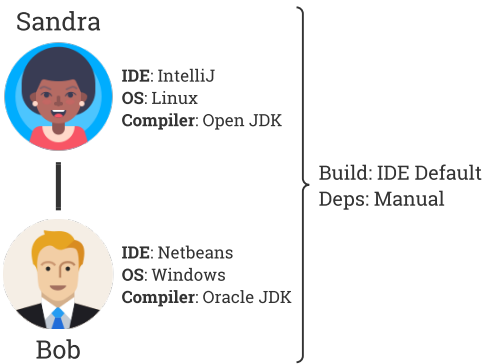
Bob

Build: IDE Default
Deps: Manual

- What can go wrong here?
- What if we add 5 more people?

Thought Experiment

Sandra Joins the Project...



- What can go wrong here?
- What if we add 5 more people?
- How about 5 more teams of 7 people?

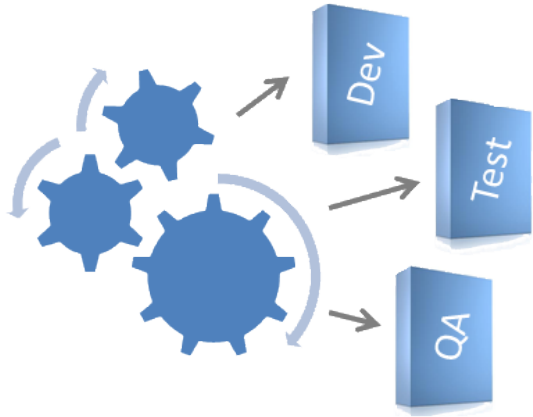
Two Components

- **Build Tools** - Automate the build process
- **Dependency Management** - Automate 3rd-party library management

All About the Automation

Building Software Sucks!

- Software Build Process
  - Develop
  - Test
  - Assemble
  - Deploy
  - Integrate
  - Repeat (again and again and again)



Build Systems

Manual VS **Automated**

- Slow
- Ad-hoc
- Error-prone/Unreliable
- Work Hours Only

- Faster
- Defined Process
- Scriptable/Reliable
- Anytime

Build Systems

Manual VS **Automated**

- Slow
- Ad-hoc
- Error-prone/Unreliable
- Work Hours Only

- Faster
- Defined Process
- Scriptable/Reliable
- Anytime

Building and deploying the project should be as easy as possible.

Automated Build Systems

Automated Build Systems allow us to:

- Download and install required libraries
- Build the software
- Execute test cases
- Package/Deploy Executables

Automated Build Systems

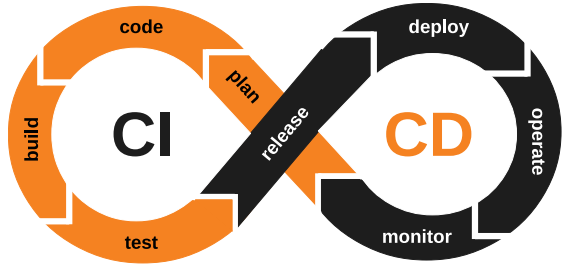
Automated Build Systems allow us to:

- Download and install required libraries
- Build the software
- Execute test cases
- Package/Deploy Executables

Automated Build Systems

Automated Build Systems allow us to:

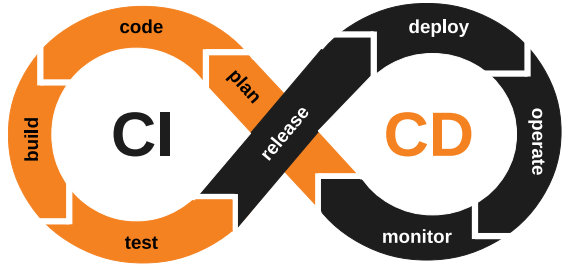
- Download and install required libraries
- Build the software
- Execute test cases
- Package/Deploy Executables



Automated Build Systems

Automated Build Systems allow us to:

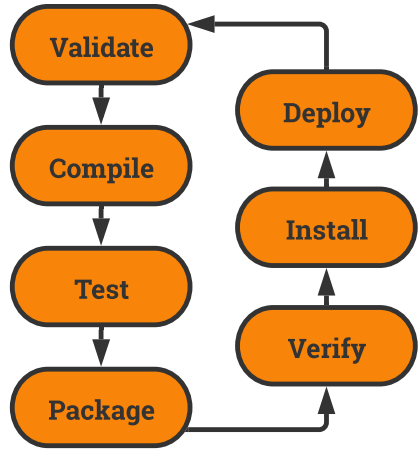
- Download and install required libraries
- Build the software
- Execute test cases
- Package/Deploy Executables



Automating repetitive tasks allows them to be run at-will.

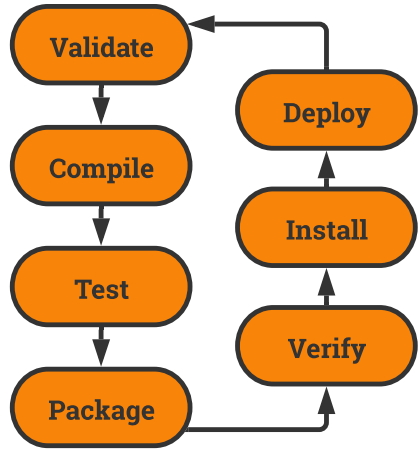
Build Lifecycle

1. **Validate** the project is correct and all necessary information is available
2. **Compile** the source code of the project
3. **Test** the compiled source code using a suitable unit testing framework
   - Run **unit tests** against classes and **subsystem integration tests** against groups of classes
4. Take the compiled code and **package** it in its distributable format (i.e., a JAR file)



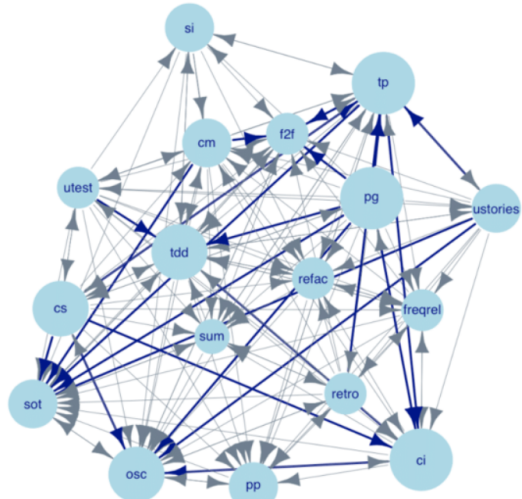
Build Lifecycle

5. **Verify** - run system tests to ensure quality criteria are met
   - System tests require a packaged executable
   - This is also when tests of non-functional criteria like performance are executed
6. **Install** the package for use as a dependency in other projects locally
7. **Deploy** the package to the installation environment



Dependency Management

- All but the simplest programs rely/reuse existing artifacts
- **Dependencies** - represent the relationship between your program and these other artifacts
  - Note, these artifacts are themselves other projects
- Dependencies may be
  - Installed programs (i.e., the `JDK`)
  - System Packages (i.e., `openssl`)
  - Programming Libraries (i.e., `JUnit`)
- Dependency management tools - provide a means to
  - Host dependencies
  - Download dependencies
  - Store dependencies
  - Install dependencies



ROAR

Dependency Repos

- Currently, most dependencies are provided via a **repository**
- Providing
  - A singular location to download from
  - Convenient mechanism for installation
- Examples
  - Java - Maven Central Repository
  - Python - PyPi (via `pip`)
  - Ubuntu - Apt repos
  - Ruby - RubyGems

Identifying Artifacts

- An **artifact** is the component we depend upon
- Typically these are identified by two or three specific components
  - Group or Author identifier (i.e., “org.junit.jupiter”)
  - The artifact identifier (i.e., “junit-jupiter-api”)
  - The artifact version (i.e., 5.7.2)
- The first two may or may not be combined
  - The group/author identifier is for the organization which created it
  - The artifact identifier is for the item depend upon
- The last item, version number, deserves a bit more understanding

Artifact Versions

- Most projects/libraries we depend upon issue a **version identifier** upon release
- Version identifiers are typically numerical, but not always
  - 8.1.3
  - 64.1.20192004
  - 5.8.0-M1
- Version Identifiers serve the following purposes
  - Ensure that the software using them continues to work

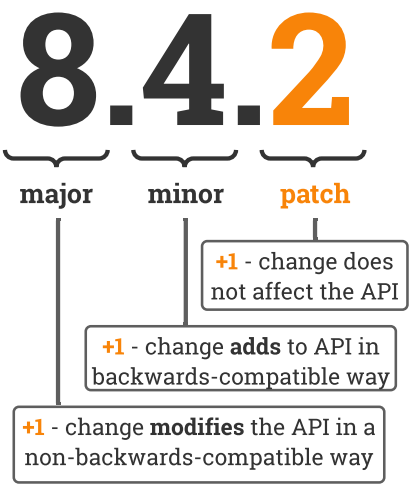
Imagine trying to build after release of a library, but failing because a request method no longer exists

Semantic Versioning

- A relatively common versioning standard
- Every version number is of the form:

What might be some advantages of this?

Semantic Versioning Rules



Lock Files

- Build and dependency management often are separate tools
- Dependency management system typically uses **lock files** to prevent dependency modification
- To upgrade dependencies you must explicitly use the management tool
  - pip in python
  - bundler in ruby

Continuous Integration

- Changing large software projects often requires:
  - Updating project documentation
  - Deploy the compiled system
  - Release the code to a dependency management repo
  - Run the test suite
  - …

Continuous Integration

- Changing large software projects often requires:
  - Updating project documentation
  - Deploy the compiled system
  - Release the code to a dependency management repo
  - Run the test suite
  - ...

- Automated builds can provide all of this
- But how do we activate the automated build?

Continuous Integration

- Changing large software projects often requires:
  - Updating project documentation
  - Deploy the compiled system
  - Release the code to a dependency management repo
  - Run the test suite
  - ...

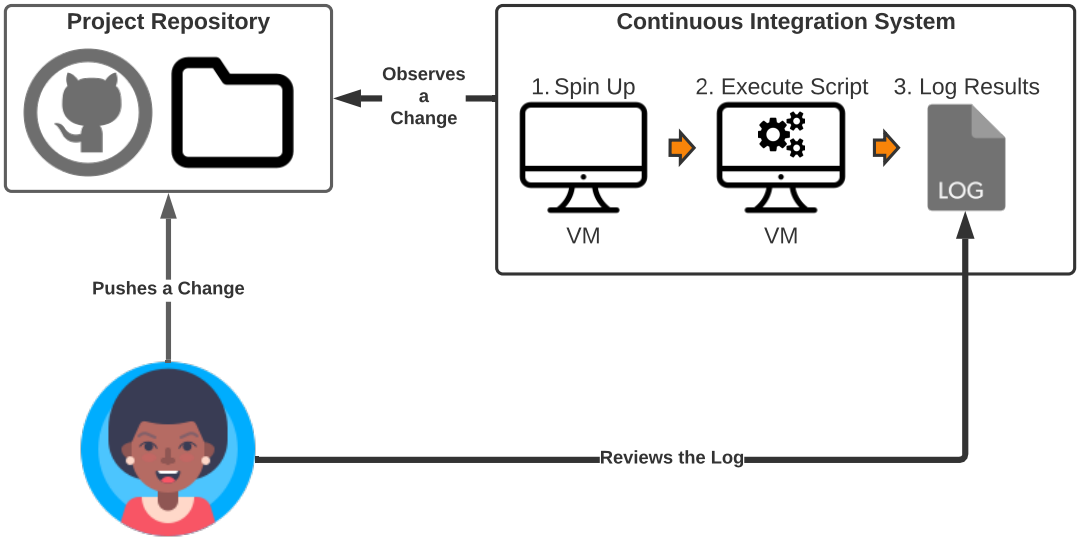
- Automated builds can provide all of this
- But how do we activate the automated build?

This is the purpose of a Continuous Integration system!

Continuous Integration

- Basically, this is an umbrella term for:
  - Stuff that runs automatically anytime a change to the system occurs
- Currently there are many options for CI
  - Travis CI
  - Circle CI
  - GitHub Actions
  - GitLab CI
  - Azure Pipelines
  - Jenkins

Continuous Integration



Examples of Tools

Build Tools

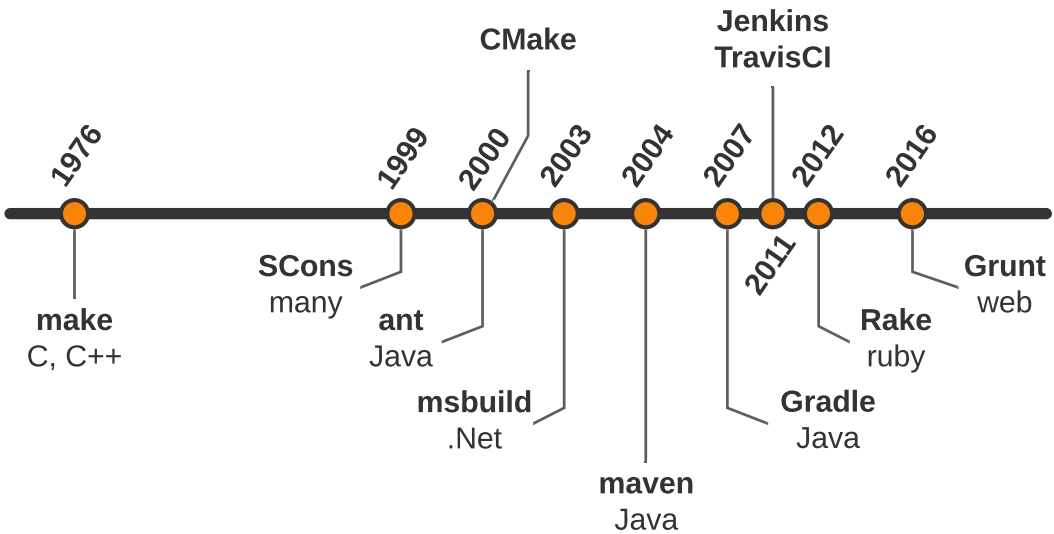
- Java: Ant, Maven, Gradle
- Android: Gradle
- .NET: MSBuild, NAnt
- Scala: sbt
- Ruby: Bundler, Rake
- C/C++: make

Continuous Integration

- GitHub Actions
- GitLab
- Jenkins
- TravisCI
- CircleCI

Idaho State University
Computer Science

Timeline of Tools



Complete A Sentence

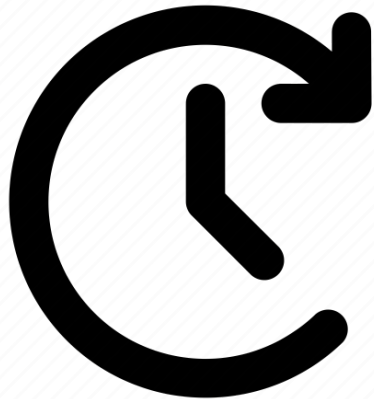
1. Take out a sheet of paper.
2. Take a few minutes to finish the following sentence:

Today, the most important thing I learned was ...

3. Keep this as part of your notes, for when you review the lecture

For Next Time

- Review the InfoQ Article
- Review this Lecture
- Come to Class
- Continue working on Homework 01
- Review the Gradle Readings
- Look at the Gradle Sample Article



Are there any questions?