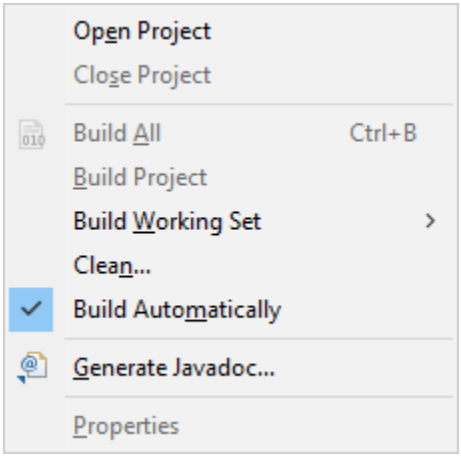
Build Automation Systems

Build Systems Review

- You've built most school projects by clicking an icon

- or even let Eclipse auto-build your project



Build Systems Review

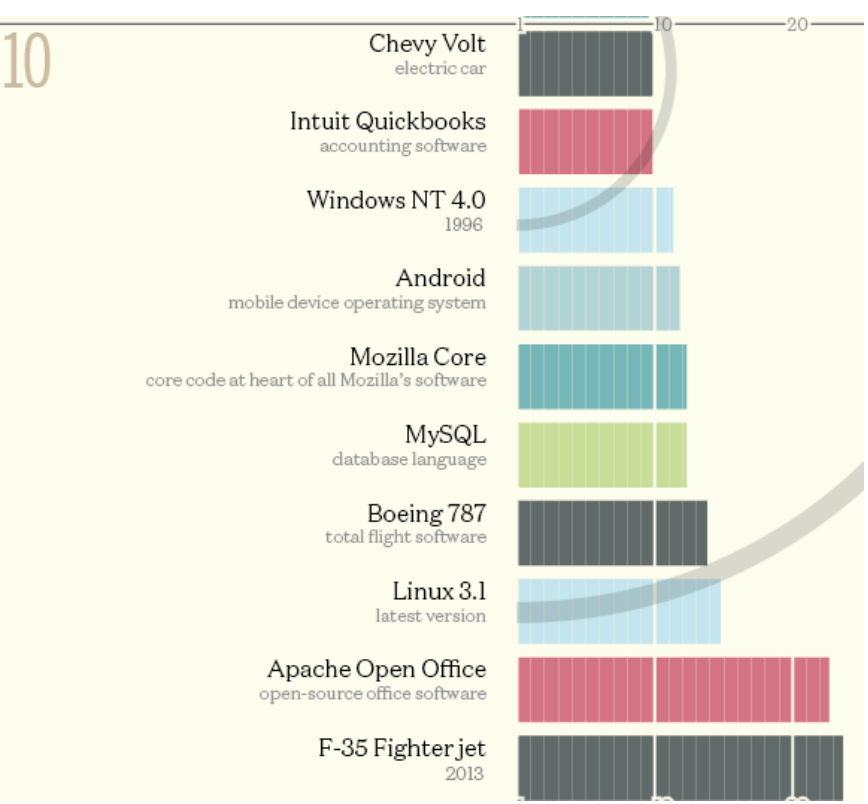
- You've built most school projects by clicking an icon

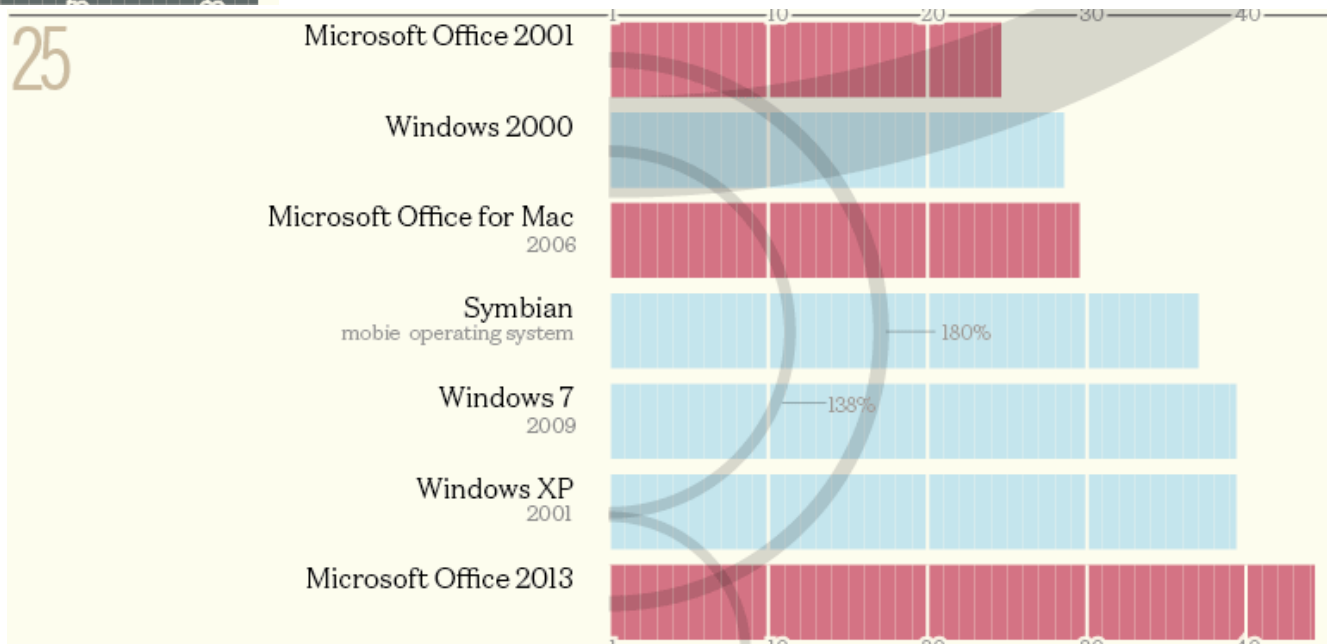
- or even let Eclipse auto-build your project
- Developers call this your private build or sandbox build
- Building commercial software is rarely this simple

How are Commercial Products Different?

Codebases (MLOC)

(Review Slide)





http://www.informationisbeautiful.net/visualizations/million-lines-of-code/

How are Commercial Products Different?

- Large products
- Developed by multiple teams
- Extensive source code management / version control system
- Potentially multiple repositories

Real-World Builds

- Often include multiple programming languages

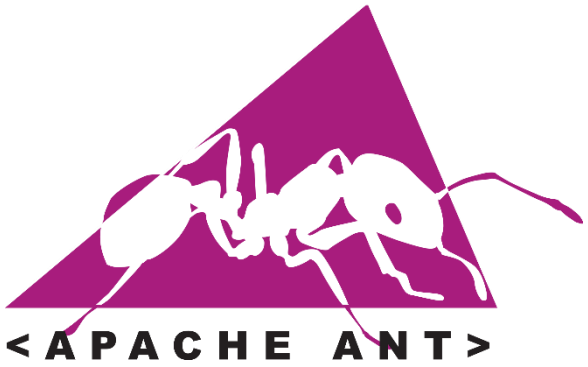
- The likelihood of integration problems is very high
  - Incorporating sub-projects delivered by other teams
  - Open-sourced projects

Development team wants

- consistency and deterministic behavior in building across multiple devices/servers
- to find/test integration problems quickly and avoid *big-bang integration*!
- to be able to back-up to previous working configuration
- Solution: Continuous Integration (CI) Server
  - which require Automatic Builders

Builders

Builders



(2000)

(2012)

make
(1976)

(2004)

(2007)

(2016)

Builders

- The Builder follows instructions/rules regarding:
  - What to build
  - How to build each deliverable and intermediary
  - The order in which things must be built

- Builds the executable binaries and much more

What Might a Builder Do?

- Check-out the latest source code and the supported compilers, libraries and tools
- Compile everything (may be multiple projects)
- Build the libraries and executables
- Execute the, Unit-Level Tests, Integration-Level (Regression) Tests and report code coverage from the automated tests
- minification, linting

What Might a Builder Do?

- May need to deploy a product on a staging server, load a database, install client apps on their platforms, etc.

Popular Builders: make

Popular Builders: make

- The original builder (1976)
- Some bias towards the needs of C/C++ projects
- Instructions provided by rules in a *Makefile*
- Fairly easy to learn how to create *Makefiles*
- Example:

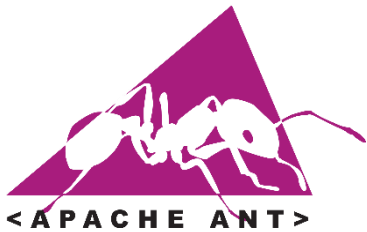
```
hellomake: hellomake.c hellofunc.c
	gcc -o hellomake hellomake.c hellofunc.c -I.
```

Popular Builders: make

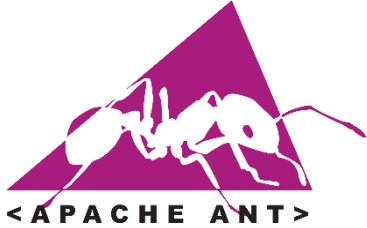
- Platform-dependent command-line instructions

- Platform-dependent path names (e.g., `bin/foo.o` vs. `bin\foo.obj`)

Popular Builders: ant



Popular Builders: ant



- Introduced in 2000
- Platform independent
- But most popular in java development (on any platform)
- Build controlled by an XML build file named, `build.xml`
- Build files tend to be detailed and lengthy
- Some developers find it difficult to learn
- .NET variation, Nant, available
- Integrates with Apache Ivy (dependency manager)
- Popularity waning in favor of maven

Popular Builders: Maven™

- Introduced in 2004

- Also platform independent, most popular for java
  - Can be used for C, C++, C#, Ruby, etc.

Popular Builders: Maven™

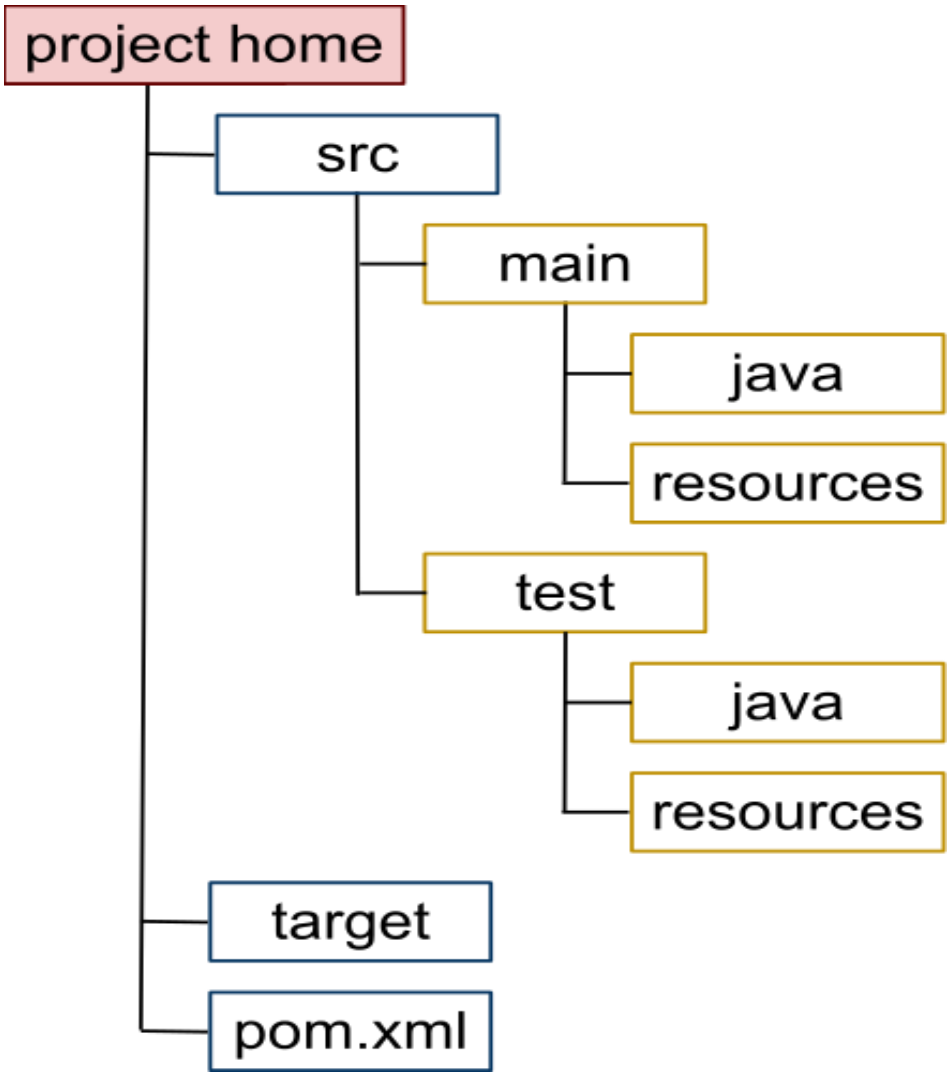
- Plug-ins extend the basic maven functionality
  - Compiling
  - Testing
  - Running code coverage tools*
  - Running mutant analysis for discovering effectiveness of tests*
  - Running static analysis tools
  - Source code management
  - Package for deployment
  - Run a web server
  - Deploy

*Homework towards the end of the semester

Popular Builders: Maven™

- Build instructions placed in Project Object Model file, `pom.xml` (located in the root of the project)
  - Defines who builds what (project components) where, when and how
  - http://maven.apache.org/pom.html
- `pom.xml` is crucial to automatic deployment (e.g., AWS, Heroku, etc.)
- Built-in conventions (i.e., if project is structured as maven expects) simplify configuration of `pom.xml`

Preferred Folder Structure of a maven Project



Maven's Build Lifecycles

- maven's *build lifecycle* ≠ a *software lifecycle* (e.g., waterfall)

- A maven *build lifecycle* defines a sequence of *phases* that control the order of the build activities

- Maven defines three lifecycles:
  - clean: removes remnants of previous builds (e.g., delete "`target`" folder)
  - default: builds the product
  - site: builds documentation

Maven's `Default` Lifecycle Phases

- Sequential *phases* for building the product. Includes:
  - `process-resources`: Check-out product source code from repository and run any required pre-processing
  - `compile`: Compile the product code
  - `process-test-resources`: Check-out test source code and files
  - `test-compile`: Compile the test code
  - `test`: Run unit-level tests (without packaging/deploying product)
  - `package`: Build installation package (e.g. JAR, WAR…)
  - `integration-test`: Run integration tests
  - `install`: Install package in local repository
  - `deploy`: Copy to remote repository for sharing with other teams
- Building `install` will build all *phases* before and including `install`
  - e.g., `mvn install`

- http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html

Maven: Dependencies

- Industry products consist of multiple Maven projects or projects dependent on libraries
- Example: Client-Server Application
  - One server project
  - Mobile client-side application project
  - Windows client-side application project
  - External libraries
- Dependencies between these are common
- Maven ensures dependent projects build/test successfully
- A project's POM.xml file has a `<dependencies>` section

Maven: Dependencies

- Specifying this in the `pom.xml`

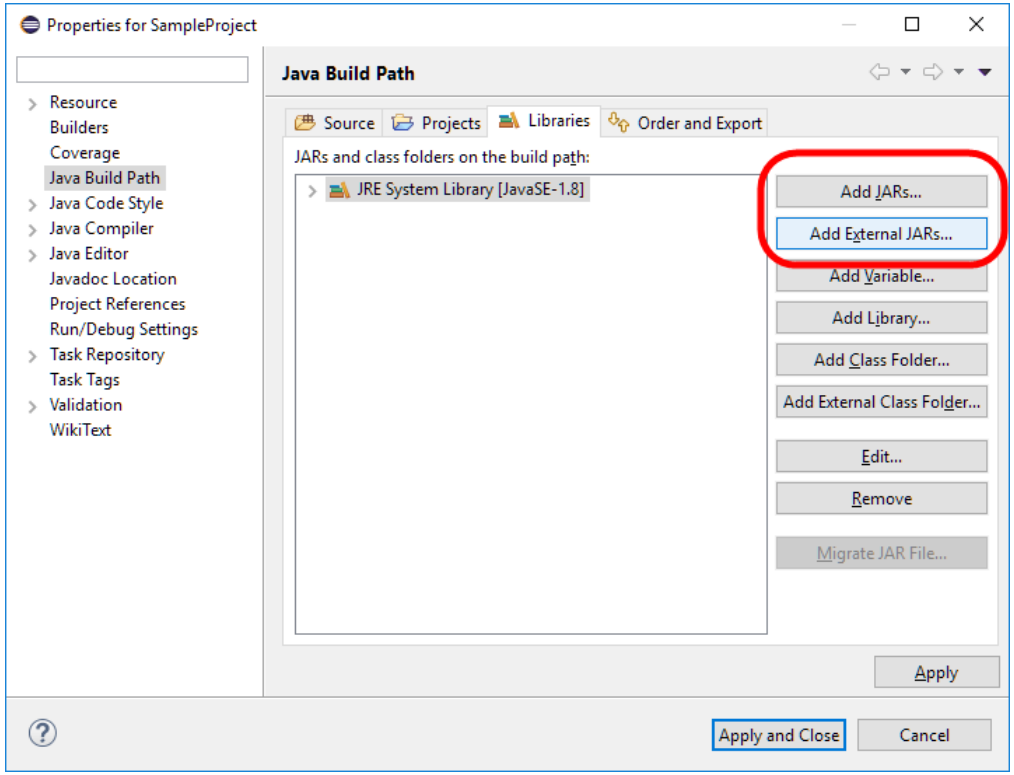
```
<dependencies>
  <dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpclient</artifactId>
    <version>4.5.2</version>
  </dependency>
  <dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.8.0</version>
  </dependency>
</dependencies>
```

Maven: Dependencies

- Specifying this in the `pom.xml`

```
<dependencies>
  <dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpclient</artifactId>
    <version>4.5.2</version>
  </dependency>
  <dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.8.0</version>
  </dependency>
</dependencies>
```

- Avoids manual specifications of these libraries (because maven downloads these specific libraries automatically)



Getting Started with Maven

- http://maven.apache.org
- Maven already installed on onyx
- Introduction: http://maven.apache.org/guides/getting-started/maven-in-five-minutes
- Documentation: http://maven.apache.org/guides/getting-started/index.html
- Eclipse Integration: http://www.eclipse.org/m2e