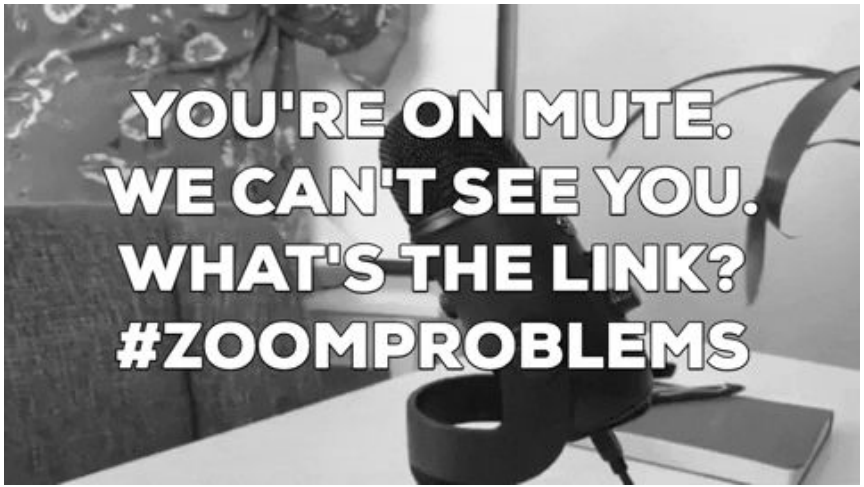
COMS W4156 Advanced Software Engineering (ASE)

September 8, 2022

Important Notice



All my classes will be held on Zoom, probably “forever” not just this semester.

If you do not want to take this course on Zoom, then please drop the course.

Prof. Yang’s W4152 Engineering Software as a Service is a reasonable substitute for most students (except not for MS Computer Security track, since this course addresses some security issues and 4152 does not).

There are a lot of students on the waitlist who would like to take this course and may not hate Zoom as much as you do!

Lecture Notes linked on Courseworks Calendar

"Live" links into google docs, so possibly changing up to and even during class

Will generate pdfs and post in Courseworks->Files->lecture notes later

Questions and Comments during Class

Zoom, either voice ("raise hand") or chat (type whenever you like)

Questions and Comments outside of Class

Ed Discussion (supports anonymous and private posts)

Agenda

1. Upstream vs. Downstream Software Engineering
2. Software Build

Upstream vs. Downstream Software Engineering

This course teaches essentially nothing about requirements analysis (i.e., upstream)

This courses teaches a little bit about software design (e.g., design principles and design patterns)

Our emphasis is on the rest of the software lifecycle (i.e., downstream)

Some implementation (coding practices, e.g., code smells, coding style)

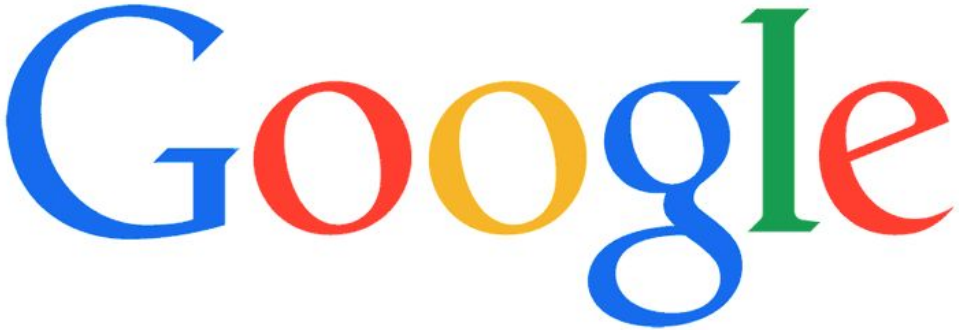
Mostly concerned with verification and maintenance (finding and fixing bugs in old and new features)

How Do I Learn More About Upstream SE?

- Take COMS W4152 Engineering Software as a Service
- Take Coursera courses on requirements analysis and software design
- Read any conventional SE textbook (e.g., Pressman, Sommerville)

There is NO Textbook for this Class

If you cannot use google or another web search engine to find tutorials and other documentation about software tools and techniques to use in the team project, and expect a textbook to tell you everything you need to know for this class, do not take this class!



Agenda

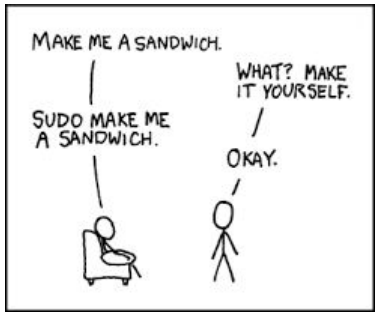
1. Upstream vs. Downstream Software Engineering
2. Software Build

Software Build

Most programming languages have some notion of '**build**', where all the project's own source code files, resources, etc. are gathered together with libraries and any other dependencies, to put the project's code into a form where it can be executed

Build has historically been associated with compile and link, as in C/C++, but even non-compiled languages have multiple project files and multiple external dependencies for all except the most trivial programs

Make



The earliest widely-used build tool was ‘make’, which was developed in mid-1970s for compiling/linking C, Fortran, and other languages in the original Unix ecosystem

Variants of make are in wide use today, e.g., GNU Make, CMake

What does this ‘makefile’ do?

What does ‘make clean’ do?

```
edit : main.o kbd.o command.o display.o \
       insert.o search.o files.o utils.o
        cc -o edit main.o kbd.o command.o display.o \
                   insert.o search.o files.o utils.o
main.o : main.c defs.h
        cc -c main.c
kbd.o : kbd.c defs.h command.h
        cc -c kbd.c
command.o : command.c defs.h command.h
        cc -c command.c
display.o : display.c defs.h buffer.h
        cc -c display.c
insert.o : insert.c defs.h buffer.h
        cc -c insert.c
search.o : search.c defs.h buffer.h
        cc -c search.c
files.o : files.c defs.h buffer.h command.h
        cc -c files.c
utils.o : utils.c defs.h
        cc -c utils.c
clean :
        rm edit main.o kbd.o command.o display.o \
           insert.o search.o files.o utils.o
```

Build Recipes

“here is the file I want to create, it depends of these other files, and there are the commands to run to build it”

Shared build configurations (scripts) enable repeatable, reproducible, standard results

Developers do not need to remember a series of steps with the appropriate inputs, options, etc. and everyone uses the same series of steps

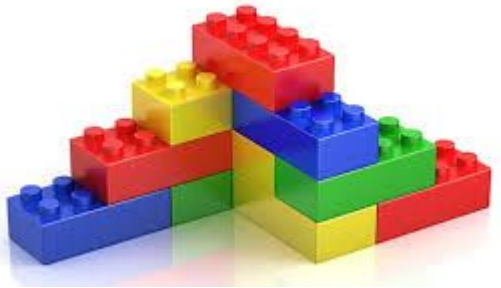
If I run build, and then you run build 12 hours later, and nothing in our shared codebase or external dependencies has changed in between, shouldn’t both builds produce bit-by-bit identical copies?

Well… not always, why not? See reproducible builds

Amaretto and Kahlua Coffee

1/2 oz Amaretto almond liqueur
1/2 oz Kahlua coffee liqueur
5 oz black brewed coffee
1 tsp sweet 'n low or equal

Package Management



Most codebases are divided into multiple components (code units) that are dependent on other components

Some codebases are entirely self-contained, but most modern software imports standard libraries (e.g., I/O, math, string processing) and employs third-party APIs (e.g., embedded OAuth client and external OAuth service)

Who are the three parties?

1. Your team
2. Your customers and users
3. Any other source of software besides your team (or your organization)

Package Management 2

The original make assumed all imported code is local, but modern package managers download external packages from other repositories and place them in the right spot in your file system

The package manager may maintain a local cache of the most recently downloaded versions of those packages, rather than re-fetching for every build

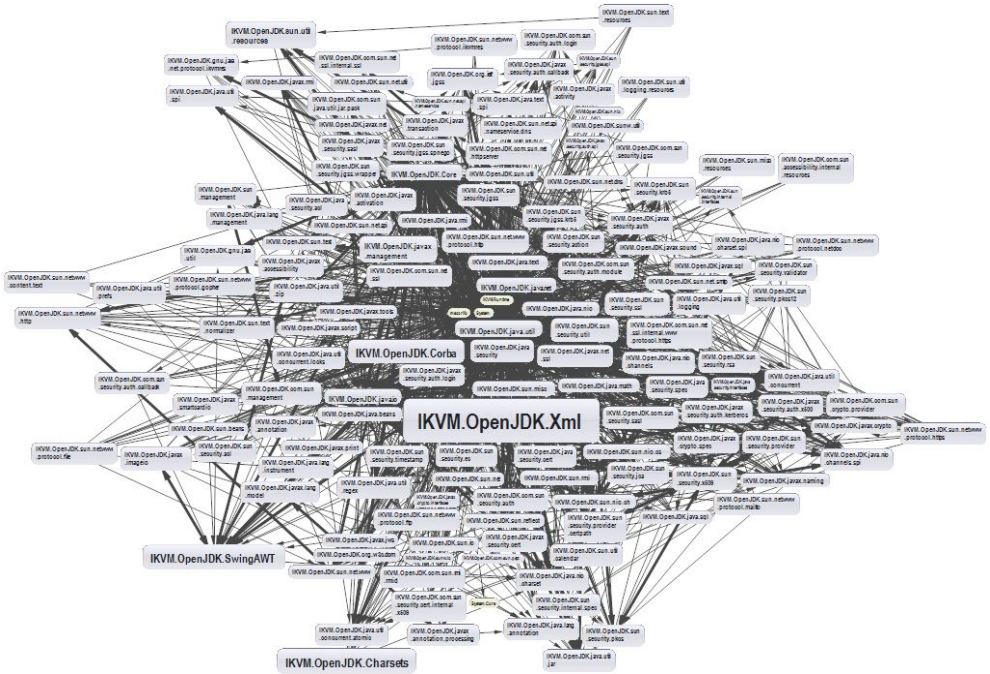
It checks for newer versions before using the locally cached version and updates the cache accordingly (unless a specific older version is required)

Dependency Management

Package dependencies may be *transitive*, A depends on B depends on C depends on D

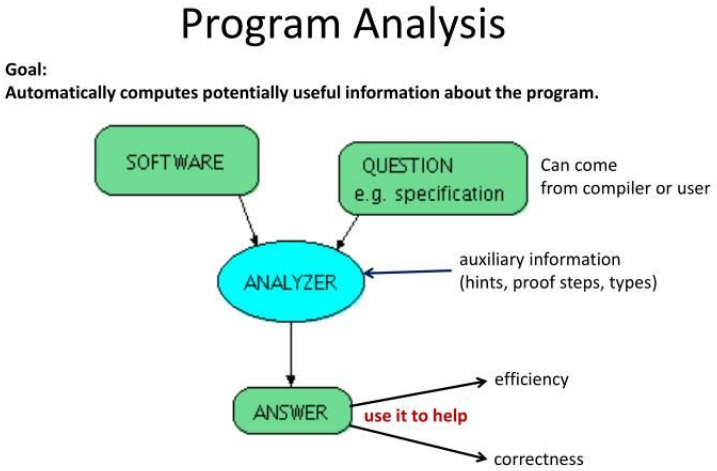
Dependency Management 2

Highly interconnected dependencies and/or reliance on specific versions of external components is **bad**, why?



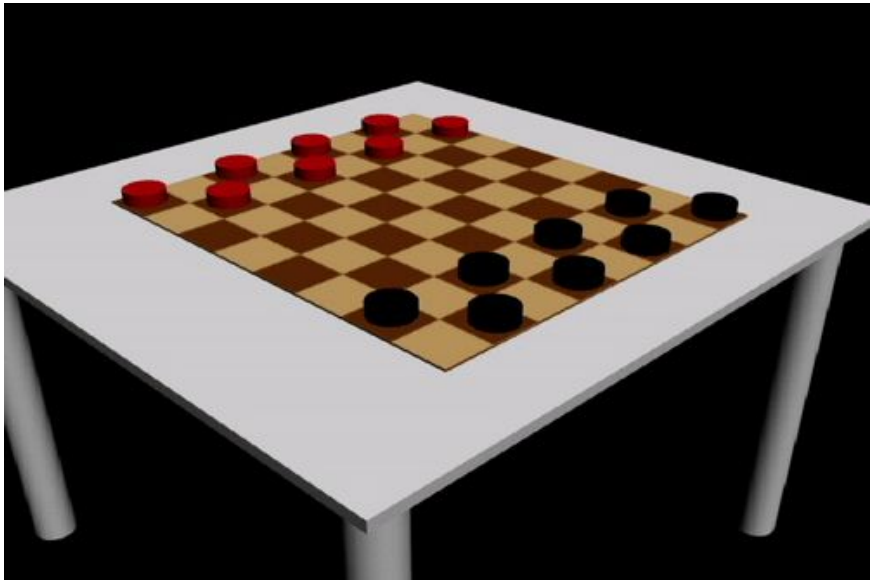
Build 2

Besides compile/link and package management, the build process often includes running a test suite and various “checkers” for desired and undesired patterns, e.g., for coding style compliance, code smells, common security vulnerabilities and bugs, etc.



Many forms of static and dynamic program analysis will be discussed later in semester

Checkers



Some checkers run during testing, such as coverage measurement, test generation to achieve higher coverage, mutation testing

Sometimes the build process is “accelerated” via test suite optimization, minimization, prioritization, and/or parallelization

Continuous Integration (CI)

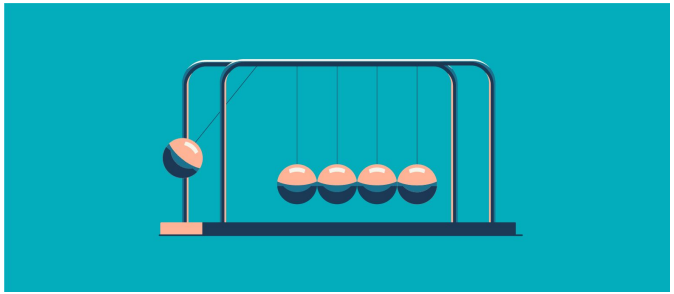
Build ideally runs automatically on every commit to shared repository, when it's easy to rollback the most recent commit(s) - but Google has 16k commits per day!

If build takes too long, then nightly build includes all the new commits from that day

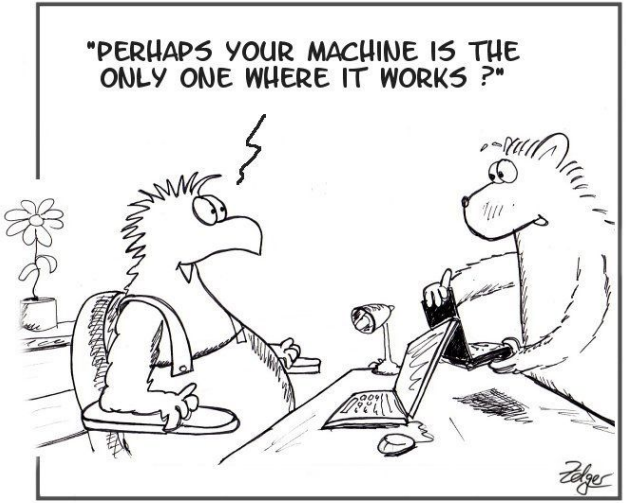
When build runs automatically on some trigger like this, known as "Continuous Integration" (CI)

Goal is to catch errors as soon as possible, why?

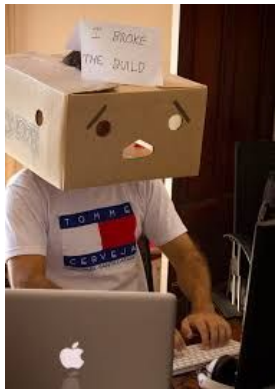
You may use any github-based CI for your team projects (e.g., circleci), but we recommend github actions workflows (covered later in semester)



Don’t Break The Build



It works on my machine



I will not break the build.
I will not break the build.
I will not break the build.
I will not break the build.
I will not break the build.
I will not break the build.
I will not break the build.
I will not break the build.
I will not break the build.

CODESMACK

Build for Java

Maven

Ant

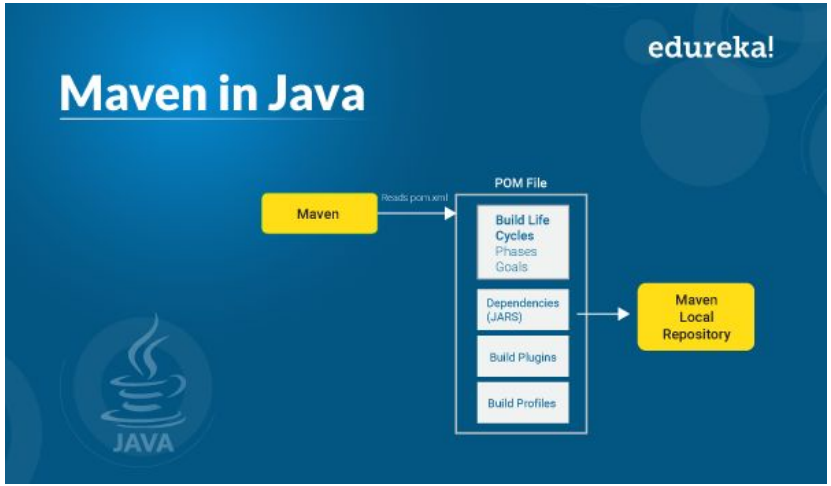
Gradle

sbt

Your team project will need to use some build tool, either for Java or C++

Maven

Maven means “accumulator of knowledge” in Yiddish



Default build recipe consists of a “project object model” (pom.xml combined with standard directory structure

Also supports numerous specialized tools and plugins via Maven Central

“Convention over configuration” → creating a Maven project results in default structure, developer only has to place files accordingly and basic pom.xml is automatically setup with corresponding configuration

Maven’s Default Locations for Project Files

item	default
source code	${basedir}/src/main/java
Resources	${basedir}/src/main/resources
Tests	${basedir}/src/test
Compiled byte code	${basedir}/target
distributable JAR	${basedir}/target/classes

Maven Demo: Nihar Maheshwari (Head TA)

Postponed
until next Tuesday

Learning More About Maven

There are numerous tutorials and lots of other documentation about Maven online

Posting useful (non-obvious) tips to the class on Ed Discussion counts towards your Participation grade. You can post anonymously wrt other students, but do not make your post anonymous to staff because then we cannot give you credit!

Learning More About Build and Package Management Tools

You will need to learn some build/package tool for your team project

Software engineers spent an immense amount of time consulting documentation

Next Class

Maven build tool demo

Unit testing

Ask Me Anything