Hard detour from regular expressions to prepare for guest speaker

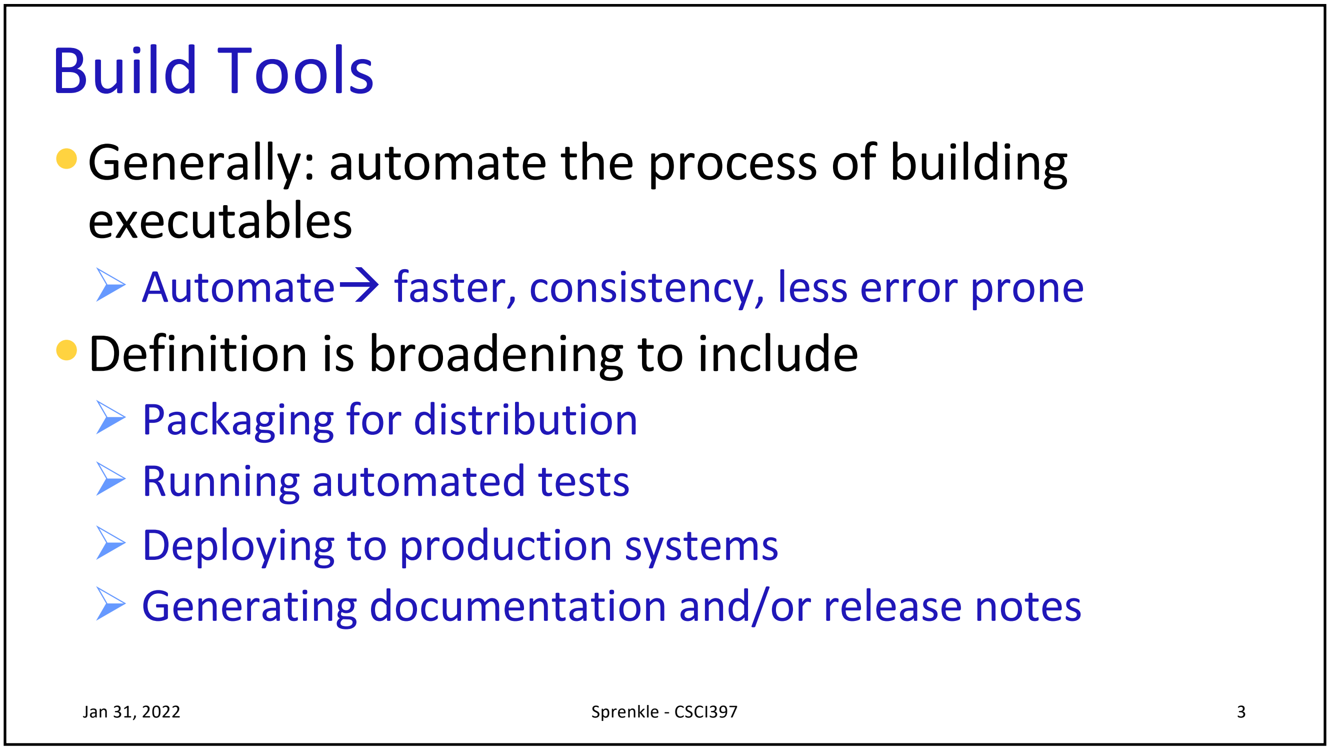
BUILD TOOLS

Distributing Software

- Pieces typically distributed:
  - Binaries/Bytecode
  - Required libraries
  - Data files
  - Documentation

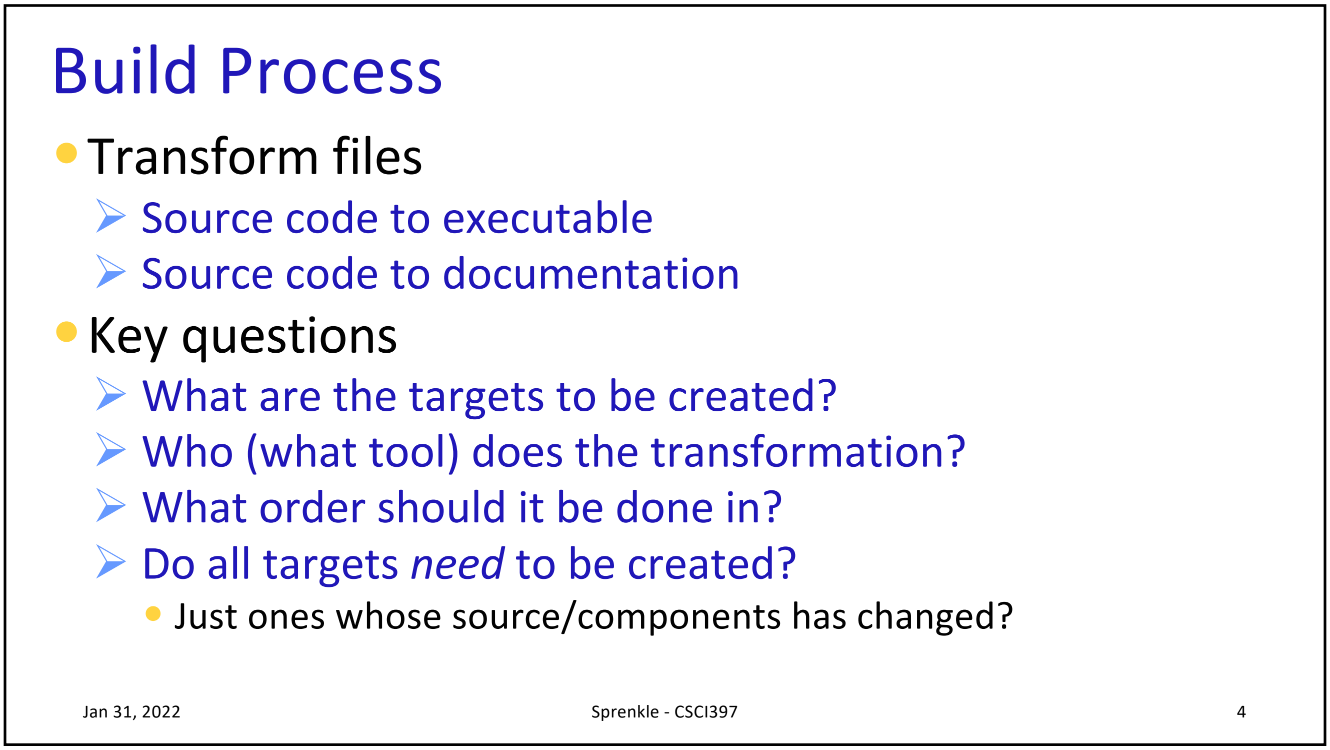
artifacts

- Often packaged in an archive:
  - e.g., tgz, jar, zip, rpm
- May need all of these or some subset of them



Build Tools

- Generally: automate the process of building executables
  - Automate→ faster, consistency, less error prone
- Definition is broadening to include
  - Packaging for distribution
  - Running automated tests
  - Deploying to production systems
  - Generating documentation and/or release notes



Build Process

- Transform files
  - Source code to executable
  - Source code to documentation
- Key questions
  - What are the targets to be created?
  - Who (what tool) does the transformation?
  - What order should it be done in?
  - Do all targets *need* to be created?
    - Just ones whose source/components has changed?

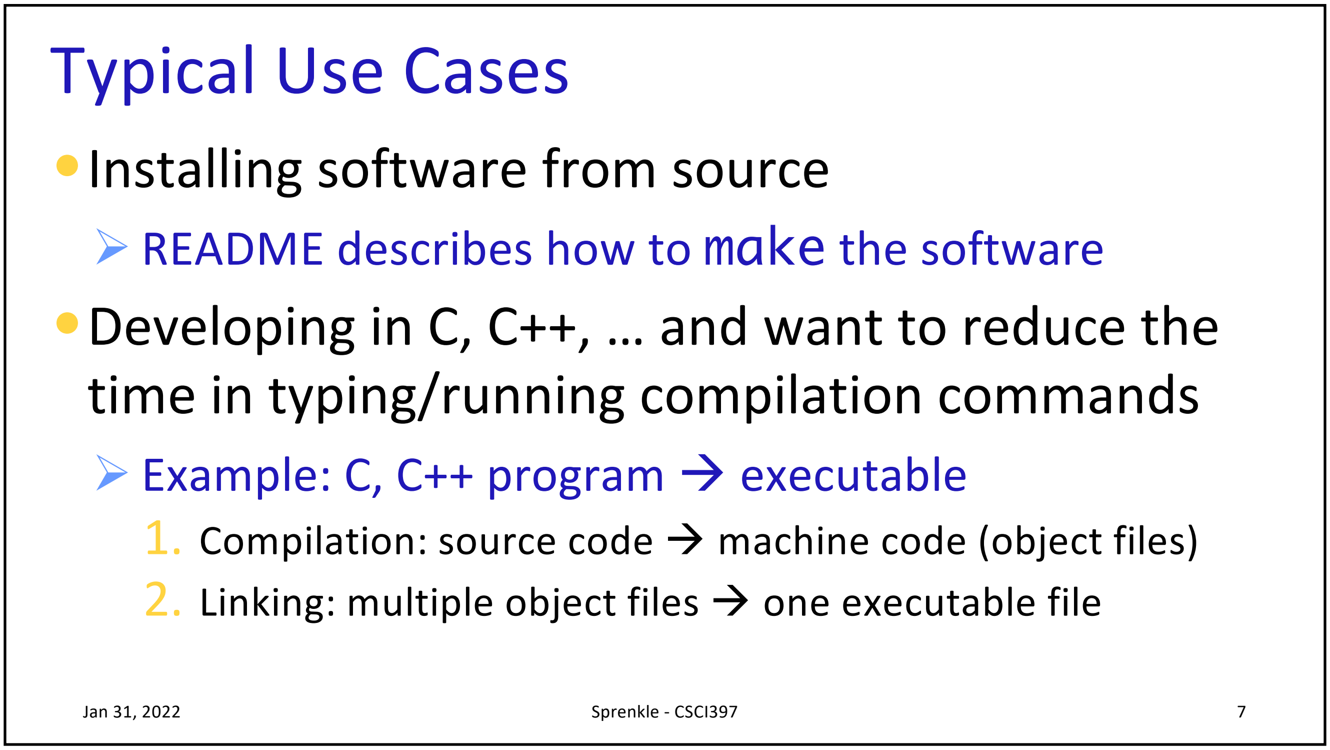
Build Tools

- Sometimes programming-language or domain specific
- Examples
  - Make
  - Ant
  - Part of Maven

make

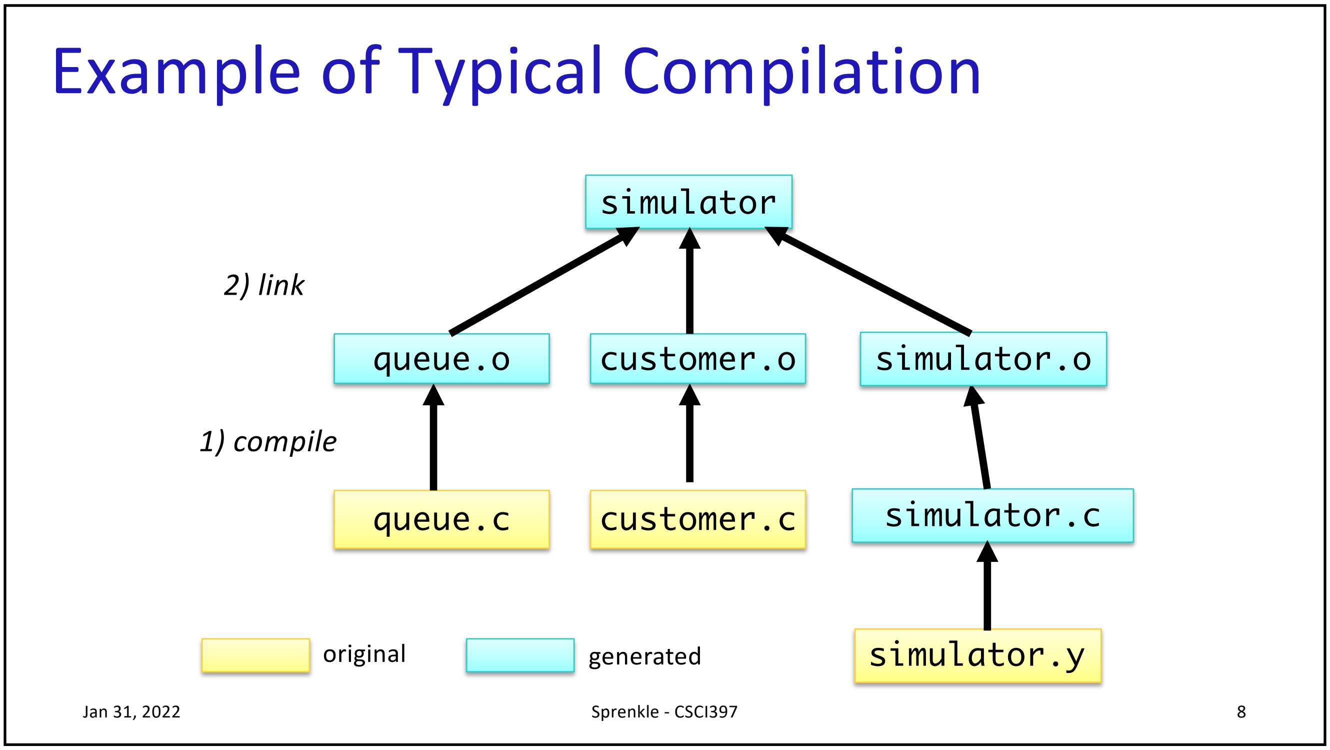
- make: A program for building and maintaining computer programs
- Developed at Bell Labs around 1978 by Stu Feldman
  - Now, head of Schmidt Sciences at the foundation Schmidt Philanthropies
  - Past President of ACM
- Won 2003 ACM System Software Award

`http://www.gnu.org/software/make/`



Typical Use Cases

- Installing software from source
  - README describes how to *make* the software
- Developing in C, C++, … and want to reduce the time in typing/running compilation commands
  - Example: C, C++ program → executable
    1. Compilation: source code → machine code (object files)
    2. Linking: multiple object files → one executable file



Example of Typical Compilation

`simulator`

2) link

`queue.o` `customer.o` `simulator.o`

1) compile

`queue.c` `customer.c` `simulator.c`

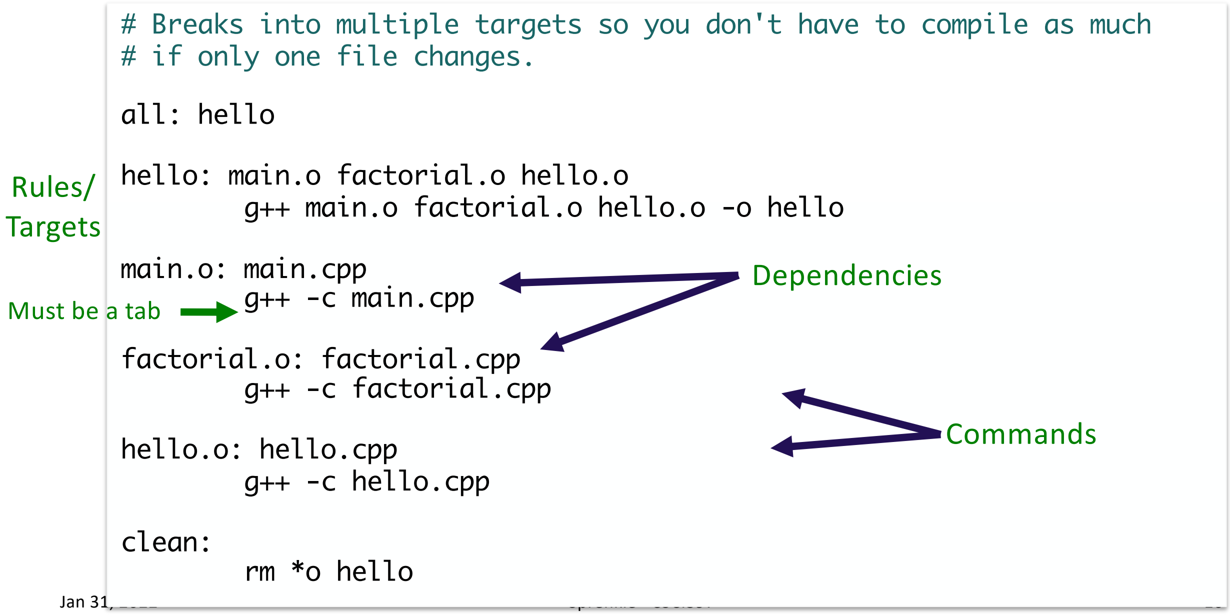
`simulator.y`

🟨 original 🟦 generated

`make` Features

- Contains the build instructions for a project in a *makefile*
  - Automatically updates files based on a series of *dependency rules*
  - Supports multiple configurations for a project
  - Language-independent
- Only re-compiles necessary files after a change (conditional compilation)
  - Major timesaver for large projects
  - Uses timestamps of the intermediate files
- Typical usage: executable is updated from object files which are in turn compiled from source files

Example Makefile



```
# Breaks into multiple targets so you don't have to compile as much
# if only one file changes.

all: hello

hello: main.o factorial.o hello.o
	g++ main.o factorial.o hello.o -o hello

main.o: main.cpp
	g++ -c main.cpp

factorial.o: factorial.cpp
	g++ -c factorial.cpp

hello.o: hello.cpp
	g++ -c hello.cpp

clean:
	rm *o hello
```

Rules/ Targets

Must be a tab

Dependencies

Commands

Example Makefile with Variables

```
# Example Makefile
CC=g++
CFLAGS=-g -Wall -DDEBUG
OBJECTS=customer.o simulator.o queue.o

simulator: $(OBJECTS)
	$(CC) $(CFLAGS) -o simulator $(OBJECTS)
simulator.o: simulator.c
	$(CC) $(CFLAGS) -c simulator.c
customer.o: customer.c
	$(CC) $(CFLAGS) -c customer.c
…
clean:
	rm $(OBJECTS) simulator
```

Variables

Rules/ Targets

Dependencies

Must be a tab

Commands

By default looks for makefile

Runs top target

Running:

```
$ make
$ make clean
$ make -f other_makefile
```

ANT

THE APACHE® SOFTWARE FOUNDATION
ESTABLISHED 1999

- All-volunteer organization
  - The largest open source foundation
- Develops >350 open-source projects

13

THE APACHE® SOFTWARE FOUNDATION
ESTABLISHED 1999

- All-volunteer organization
- Develops >350 open-source projects
  - **Httpd (Web Server)**
  - Tomcat (web application server)
  - Struts, Wicket (web application development frameworks)
  - Hadoop (distributed data processing)
  - Spam Assassin
  - Common library

14

Related Tools: Apache Ant

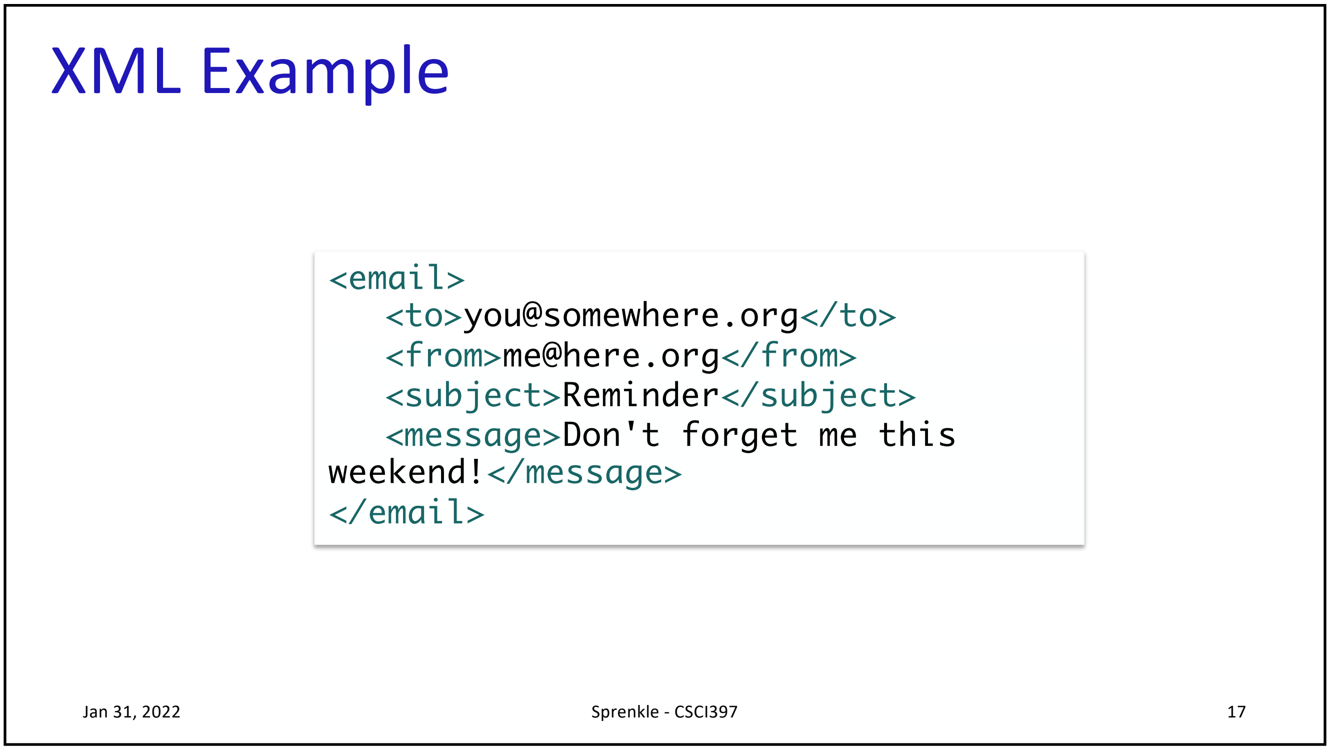
- Java-based build tool
- Similar to `make`

Make	Ant
Shell-based, makefile	Java, XML config files

`http://ant.apache.org/`

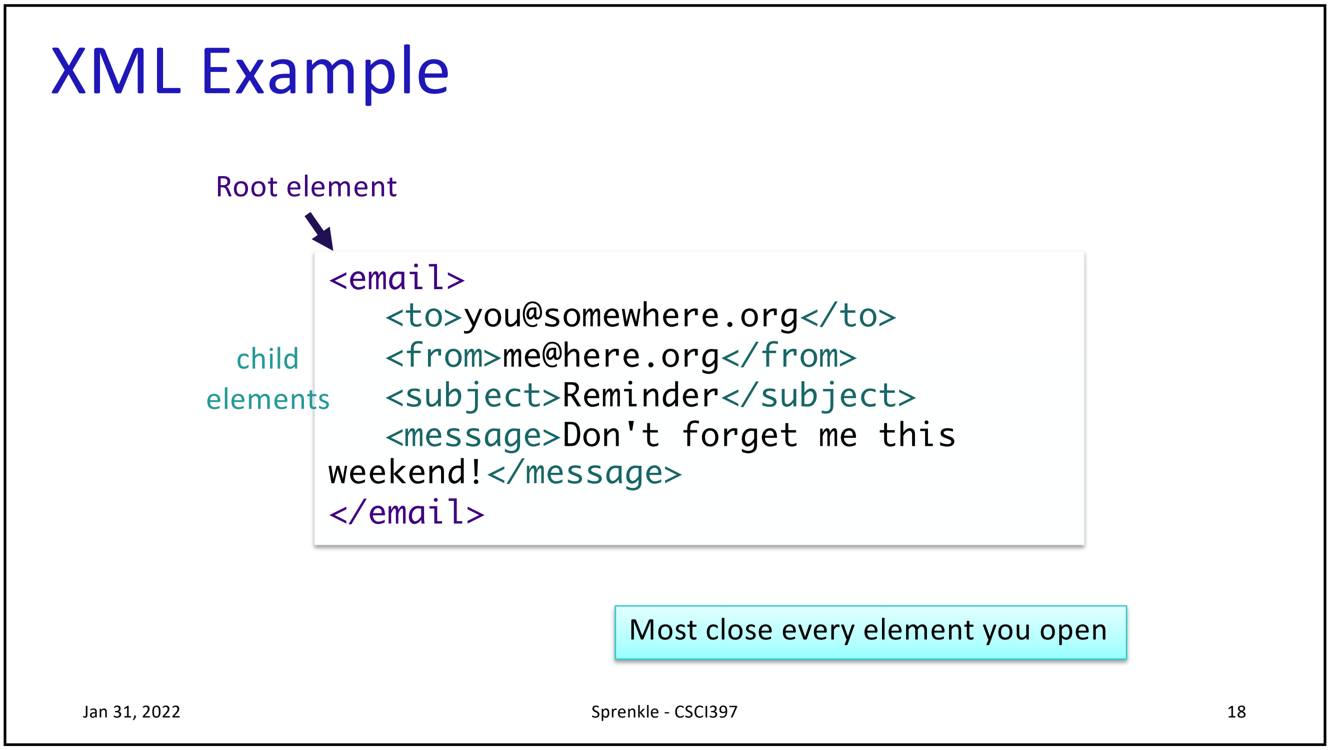
XML: eXtensible Markup Language

- Looks similar to HTML
  - HTML’s stricter sibling
- Designed to structure, store, and transport data
  - Text file → PORTABLE!
- Made up of *nested* elements
  - Hierarchy of data
- Schema
  - *Define* your own tags, tag nesting, tag attributes



XML Example

```
<email>
   <to>you@somewhere.org</to>
   <from>me@here.org</from>
   <subject>Reminder</subject>
   <message>Don't forget me this 
weekend!</message>
</email>
```



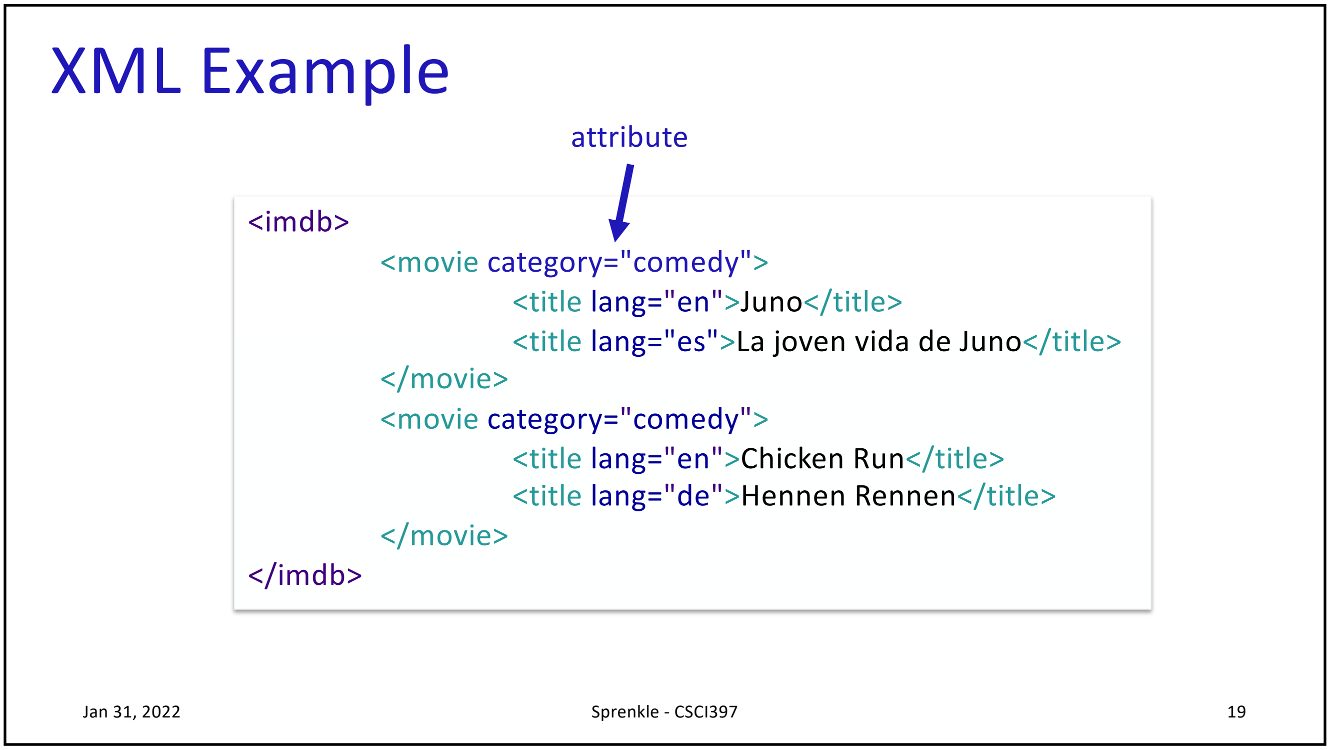
XML Example

Root element

child elements

```
<email>
   <to>you@somewhere.org</to>
   <from>me@here.org</from>
   <subject>Reminder</subject>
   <message>Don't forget me this 
weekend!</message>
</email>
```

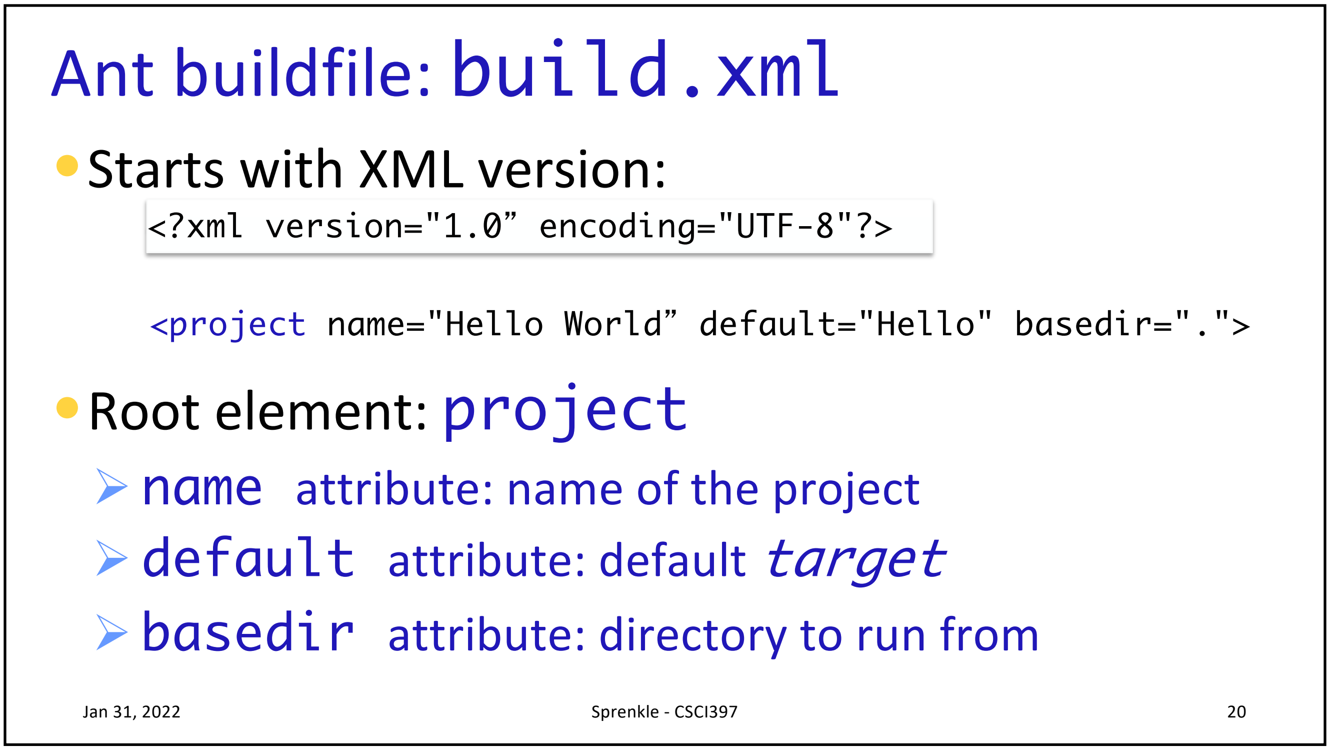
Most close every element you open



XML Example

attribute

```
<imdb>
        <movie category="comedy">
                <title lang="en">Juno</title>
                <title lang="es">La joven vida de Juno</title>
        </movie>
        <movie category="comedy">
                <title lang="en">Chicken Run</title>
                <title lang="de">Hennen Rennen</title>
        </movie>
</imdb>
```



Ant buildfile: build.xml

- Starts with XML version:

```
<?xml version="1.0” encoding="UTF-8"?>
```

```
<project name="Hello World” default="Hello" basedir=".">
```

- Root element: project
  - name attribute: name of the project
  - default attribute: default *target*
  - basedir attribute: directory to run from

Ant target

- Target: has a name, set of tasks to execute
- Can specify which targets to execute
  - If no target given, use project’s default
- Can depend on other targets
- Examples:
  - Compile
  - Distribute
    - Needs compile

Closes open tag

```
<target name="compile"/>
<target name="jar" depends=“compile"/>
```

Example Ant Target

What does this do?

```
<target name="compile"
      description="Compile the source code">
   <mkdir dir="build/classes"/>
   <javac srcdir="src"
          destdir="build/classes"
          debug="on">
      <include name="**/*.java"/>
      <classpath refid="build.class.path"/>
   </javac>
</target>
```

build-replay.xml

Ant property

- Like a variable: defines a name and its value:
  - `<property name="vname" value="vvalue" />`

- To use property, use ${vname}

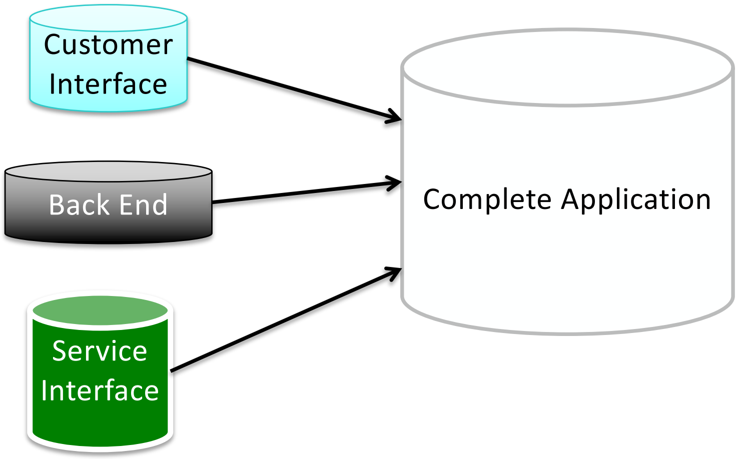
Ant in Eclipse

- Add two new targets
- First:
  - Use ctl-space to auto-complete

```
<target name="Hello”>
    <echo>${HelloText}</echo>
</target>
```

- Second: use Eclipse’s design view
- Run file

Motivating Build Tools: Common Use Cases



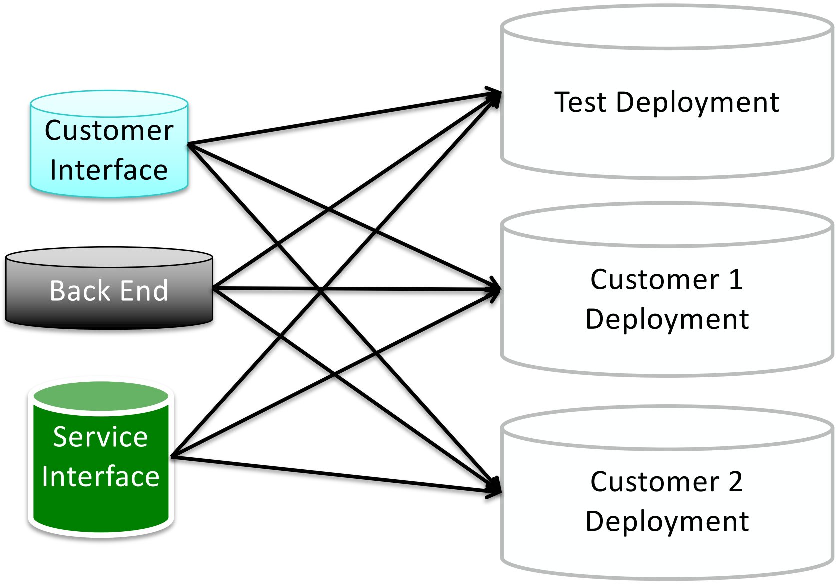
Customer Interface

Back End

Service Interface

Complete Application

Motivating Build Tools: Common Use Cases



Customer Interface

Back End

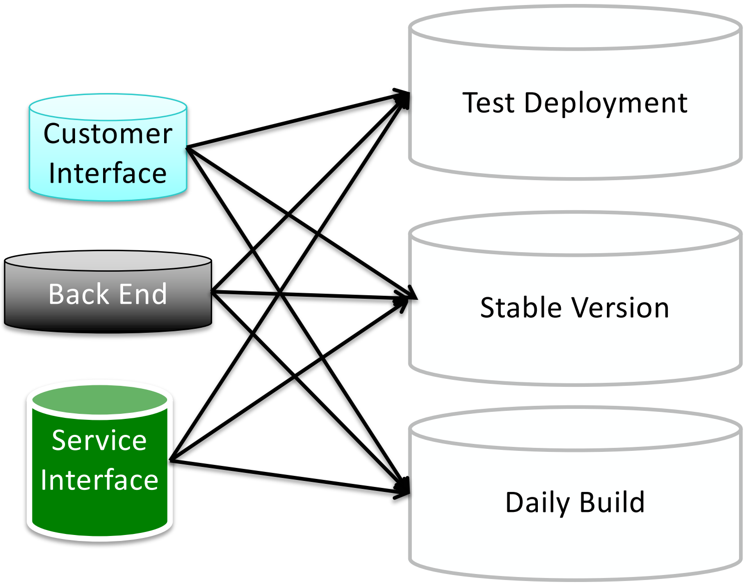
Service Interface

Test Deployment

Customer 1 Deployment

Customer 2 Deployment

Motivating Build Tools: Common Use Cases



Customer Interface

Back End

Service Interface

Test Deployment

Stable Version

Daily Build

Create every night
Then run test cases

Generate at the
end of a cycle

Generate every night
for early adopters

Apache Maven Project
http://maven.apache.org/

Apache Maven™

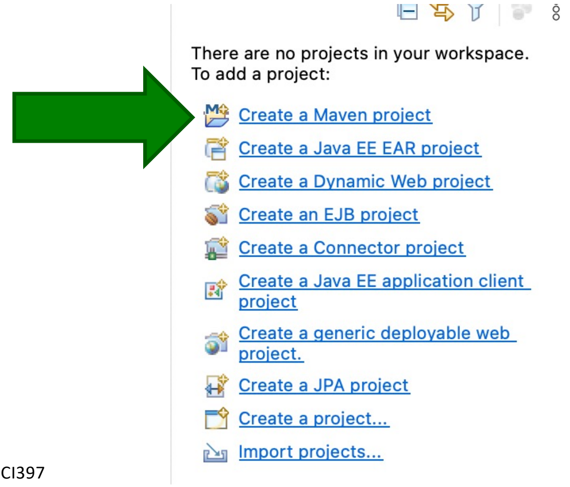
- Maven: Yiddish word meaning *accumulator of knowledge*
- Evolved from struggles in maintaining an Apache project
- For building **and managing** any Java-based project
    - Uses a Project object model (POM)
- Goal: download and build a project quickly

Jan 31, 2022 Sprenkle - CSCI397 `http://maven.apache.org/` 29

29

Maven

- Can be used as standalone tool or within Eclipse (what we'll do)



There are no projects in your workspace.
To add a project:

- Create a Maven project
- Create a Java EE EAR project
- Create a Dynamic Web project
- Create an EJB project
- Create a Connector project
- Create a Java EE application client project
- Create a generic deployable web project.
- Create a JPA project
- Create a project...
- Import projects...

Jan 31, 2022 Sprenkle - CSCI397

30

Maven Philosophy: Convention Over Configuration

- Maven's location assumptions:
  - source code: ${basedir}/src/main/java
  - Resources: ${basedir}/src/main/resources
  - Tests: ${basedir}/src/test
- Other assumptions:
  - Want to produce a JAR file in ${basedir}/target
  - Compile byte code to ${basedir}/target/classes

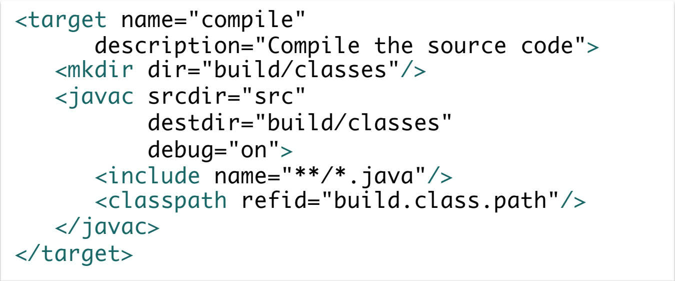
How does this philosophy help us?

Maven Philosophy: Convention Over Configuration

How does this philosophy help us?

- Ant-based builds *define* locations
  - No built-in idea of where source code or resources are
  - **User** has to supply this information → more work for us!!

Could be for any project:



```
<target name="compile"
      description="Compile the source code">
   <mkdir dir="build/classes"/>
   <javac srcdir="src"
          destdir="build/classes"
          debug="on">
      <include name="**/*.java"/>
      <classpath refid="build.class.path"/>
   </javac>
</target>
```

Maven Philosophy: Convention Over Configuration

- Beyond location conventions…
- **Core plugins** apply a common set of conventions for compiling source code, packaging distributions, generating web sites, and many other processes
  - Example: similar to Ant compile target
- Little effort:
  - Put source in the correct directory
  - Maven handles the rest

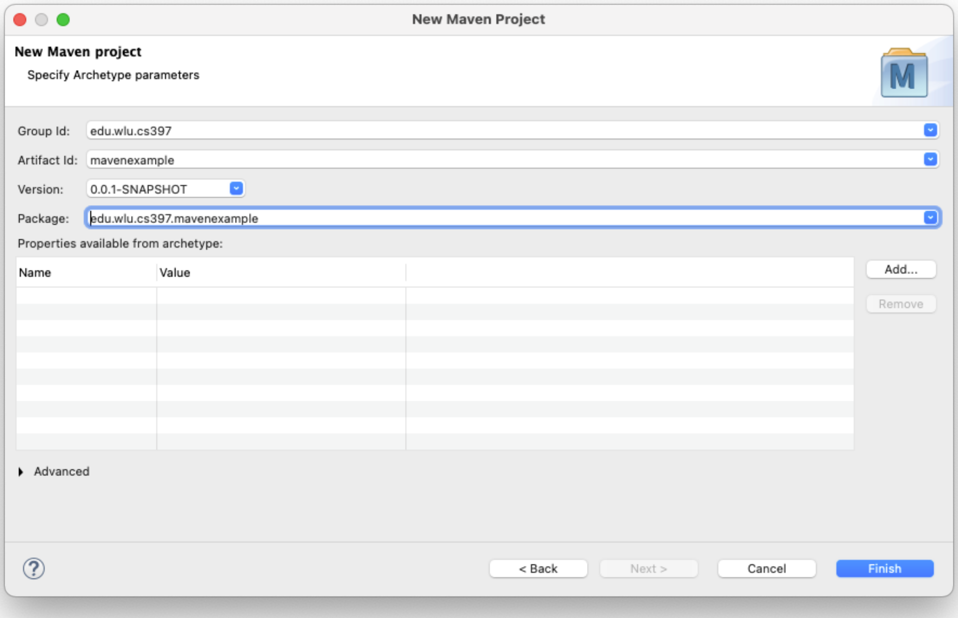
Consequences of Convention Over Configuration

- Users may feel forced to use a particular methodology or approach
- Most defaults can be customized
- Can create custom plugins for your requirements

Maven in Eclipse

- Create a new Maven project
- Filter: maven-archetype-quickstart

Maven



New Maven Project

New Maven project

Specify Archetype parameters

Group Id: edu.wlu.cs397

Artifact Id: mavenexample

Version: 0.0.1-SNAPSHOT

Package: edu.wlu.cs397.mavenexample

Properties available from archetype:

Name	Value

Look at Created Project

- What are the directories, files?
- What are the dependencies?

Project Object Model: pom.xml

- Defines information about your project
  - Which plugins
  - Which dependencies and which versions

Maven Build Lifecycle

- Defined by a list of *build phases*
- Example build phases
  - `compile` - compile the source code of the project
  - `test` - test the compiled source code using a suitable unit testing framework
  - `package` - take the compiled code and package it in its distributable format, such as a JAR
- When execute a phase, executes life cycle's previous phases first, in order
  - E.g., calling package would execute compile and then test

Maven Build Lifecycle

- 3 built-in build lifecycles
  - `default` lifecycle handles project deployment
  - `clean` lifecycle handles project cleaning
  - `site` lifecycle handles the creation of project's site documentation

Updating Project

- In pom.xml, change Java version from 1.7 → 11 (or 14 or 16)
  - maven.compiler.source
  - maven.compiler.target
- Update project
- Run as → Maven build
  - Goals: clean verify

Updating Dependencies

```
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-api</artifactId>
  <version>5.7.2</version>
  <scope>test</scope>
</dependency>
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-engine</artifactId>
  <version>5.7.2</version>
  <scope>test</scope>
</dependency>
```

Maven Repository

https://mvnrepository.com/

- How to use it
- Typically: looking for a stable release
  - rc = release candidate

Looking Ahead

- Guest speaker on Wednesday
- Friday: Assignment 0 due