CSE 70: Software Development Pipeline
Build Process, XML, Repositories

Ingolf Krueger
Learning Goals for Today
Learning Goals

• Be able to identify key elements of the software development pipeline

• Understand a basic Java program and how it is compiled/run

• Be able to change a given program to introduce functions and additional classes that can evolve separately

• Understand the need for a build tool

• See Ant used to automate compilation and other tasks

• Be able to write basic Ant files yourself

• Understand the challenges of maintaining a code base in an individual and team development setting

• Understand the need for and basic principles of version control/configuration management
Key elements of the software development pipeline
Software Development Pipeline: Requirements ➔ System
Stage 0: Individual, using an IDE
Stage 1: Individual, using IDE, Test and Build Tools
Stage 2: Individual, w/IDE, Local Build/Test Tools and Version Control
Stage 3: Small Team w/IDE, Test, Build, Version Control
Stage 4: Small to Medium Teams w/Automated Build Process
Stage 4: Small to Medium Teams w/Automated Build Process

We’ll cover all of these over the next few lectures and labs
Stage 4: Small to Medium Teams w/Automated Build Process

Today’s focus: Build Tools & Basics of Version Control

Diagram showing the flow of tool integration for software development processes.
Stage 0 to Stage 1

Why?

Requirements, Architecture, Design → IDE → Deployed System

IDE → Local Test and Build Tools → IDE → Deployed System
Write a program that formats and outputs a given String

Hello! → H-e-l-l-o-!-
We want to change the Formatter and the main program separately:

Introduce a StringFormatter class
1st attempt: introduce a dedicated class!

```java
package today;

public class StringFormatter {

    StringFormatter(){};

    public String formatString(String in) {
        char[] cv = new char[2*in.length()];
        for(int i = 0; i < cv.length; i+=2) {
            cv[i]   = in.charAt(i/2);
            cv[i+1] = '-';
        }
        return new String(cv);
    }
}
```
1st attempt: introduce a dedicated class!

package today;

public class Printer {
    public static void main(String[] args) {
        StringFormatter sf = new StringFormatter();
        String out = sf.formatString("Hello!");
        System.out.println("formatted: "+out);
    }
}
Looks good! Let’s get it compiled!

By now we have TWO Java source files:

- Printer.java
- StringFormatter.java

In what order do we need to compile them?
Java Compile Command (command line)

1

javac -d bin -cp bin src3/StringFormatter.java

2

javac -d bin -cp bin src3/Printer.java
Introduce symbolic names for special characters
2nd attempt: add a class for symbolic names!

package today;

public abstract class CharConstantHolder {
    public static char SPACE = ' ';  
    public static char DASH  = '-';
    public static char STAR  = '*';
}

public class StringFormatter {
    ...
    cv[i]   = in.charAt(i/2);
    cv[i+1] = CharConstantHolder.STAR;
    ...
}
Looks good! Let’s get it compiled!

By now we have THREE Java source files:

1. CharConstantHolder.java
2. StringFormatter.java
3. Printer.java

In what order do we need to compile them?
Java Compile Command (command line)

1

javac -d bin -cp bin src4/CharConstantHolder.java

2

javac -d bin -cp bin src4/StringFormatter.java

3

javac -d bin -cp bin src4/Printer.java
Can we avoid manual recompiles and automate other tasks as well?

Yes! Enter: Ant
What is Ant?

- Apache Ant is an open source Java-based **Build tool**
- Works across platforms – written in Java, input: XML

**Interlude: XML**
eXtensible Markup Language (XML)

- XML is a set of rules/guidelines for describing structured data in plain text
- XML supports creation of domain-specific languages that add meaning to data

Example

```
<project>
  <name> week2 </name>
  <basedir> . </basedir>
  <default> compile </default>
</project>
```

Start tag

End tag
XML: Tags and Elements

- XML is a set of rules/guidelines for describing structured data in plain text
- XML supports creation of domain-specific languages that add meaning to data

Example

Start tag

Element/Data

End tag
XML Example

- XML is a set of rules/guidelines for describing structured data in plain text
- XML supports creation of domain-specific languages that add meaning to data

```xml
<project>
  <name> week2 </name>
  <basedir> . </basedir>
  <default> compile </default>
</project>
```
XML Example

- XML is a set of **rules/guidelines** for describing **structured data in plain text**
- XML supports creation of domain-specific languages that add meaning to data

```xml
<project>
  <name> week2 </name>
  <basedir> . </basedir>
  <default> compile </default>
</project>
```
In short: *tags* add “meaning” to *data*
Writing Data Structures in XML vs. Java

Example.xml

```xml
<project>
  <name> week2 </name>
  <basedir> . </basedir>
  <default> compile </default>
</project>
```

```
class Project {
  private:
    Name    name;
    Basedir basedir;
    Default default;
  ...
}
```

```
class Name {
  private:
    String s;
  ...
}
```

```
class Basedir {
  private:
    String bd;
  ...
}
```

```
class Default {
  private:
    String d;
  ...
}
```
Alternative Styles

<project>
  <name> week2 </name>
  <basedir> . </basedir>
  <default> compile </default>
</project>

vs.

<project name = "week2" basedir = "." default = "compile">
</project>

vs.

<project name = "week2" basedir = "." default = "compile"/>

Attribute
Write an XML file that holds Customer information using suitable tags and data entries:

The customer’s first and last name, their phone number, and their email address!

Example customer:
Bilbo Baggins, 123.456.789, Bilbo.Baggins@lor.com
<customer>
  <phone> 123.456.789 </phone>
  <first_name> Bilbo </first_name>
  <email> Bilbo.Baggins@lor.com </email>
  <last_name> Baggins </last_name>
</customer>
XML Pros and Cons

- XML files can be “self-documenting” via well-chosen tags
- Easily extendible
- Broadly applicable and applied
  - communication standards
  - B2B data exchange
- For small files: all you need is a text editor

- Tags/tag syntax add complexity to the file
- XML files can become unwieldy; special editing software can mitigate this
- Performance loss
Back to *Ant* and its build files
What is Ant?

• Apache Ant is an open source Java-based **Build tool**
• Works across platforms – written in Java, input: XML
• Basic idea:
  1. Read in a project specification (the “build.xml” file), containing
     • Targets:
       – A set of tasks you want to automate
       – Each target can depend on multiple other targets
     • Tasks (something you want to get done automatically), examples:
       – Compile a set of Java files
       – Create, delete, move a file/directory
       – Create a jar file
       – Email a zipped archive of your source
  2. Determine dependencies among targets and within tasks
  3. Execute all required tasks following the order determined in 2.
Remember

Ant

build.xml

output files

other side-effects
• create/delete files
• create/delete dirs.
• program execution
• ...

.java

.class

other input
Ant – Build File: Project

Base directory, relative to location of build.xml

<project name = "week2" basedir = "." default = "compile">
    <target name = "compile">
        <javac srcdir = "src4" destdir = "bin"/>
    </target>
</project>

build.xml

Default target
Ant – Build File: Target

build.xml

```xml
<project name = "week2" basedir = "." default = "compile">
  <target name = "compile">
    <javac srcdir = "src4" destdir = "bin"/>
  </target>
</project>
```

**Target specification:**
One set of tasks you want to automate
To execute the target named “compile”, type

```
ant compile
```
Add a target that cleans out the "bin" directory
<project name = "week2" basedir = "." default = "compile">

  <target name = "compile">
    <javac srcdir = "src4"
      destdir = "bin"/>
  </target>

  <target name = "clean">
    <delete dir= "bin"/>
  </target>

</project>

To execute the target named “clean”, type

ant clean

Task specification: remove the directory “bin” and all of its contents!!!
Add a target that creates the “bin” directory if it does not already exist
Ant – Build File: Task

build.xml

```xml
<project name = "week2" basedir = "." default = "compile">
    <target name = "compile">
        <javac srcdir = "src4"
             destdir = "bin"/>
    </target>

    <target name = "clean">
        <delete dir= "bin"/>
    </target>

    <target name = "init">
        <mkdir dir = "bin"/>
    </target>

</project>
```

**Task specification:** create the directory "bin" if it doesn't exist already

To execute the target named “init”, type

`ant init`
Add dependencies between compile and init targets
Ant – Build File: Task

```xml
<project name="week2" basedir="." default="compile">
  <target name="compile" depends="init">
    <javac srcdir="src4"
      destdir="bin"/>
  </target>
  <target name="clean">
    <delete dir="bin"/>
  </target>
  <target name="init">
    <mkdir dir="bin"/>
  </target>
</project>
```

What happens now if you enter the following

```
ant clean; ant compile
```
Version Control
(a.k.a Configuration Management)
with *subversion*
Stage 1 to Stage 2

Why?
Scenario

What can go wrong with your source files even if you are the only developer, and you are developing locally on your machine?
Problems for Individual Non-Version Controlled Software

• If you save your work over what you had before, any prior work is gone forever
  – Some help from “backup mode” of IDEs/Editors, but only a crutch
  – Some help from physical backups, but how timely are they?

• No ability to revert back to a prior version of a particular file (the one without the error!)
  – Never happened to you?
  – It will – trust me!
Problems for Individual Non-Version Controlled Software

• How do you freeze one version for deployment and continue to work on a new version?
  – Crucial if you need to deliver software at some point
  – Many ad-hoc solutions that quickly fall apart due to lack of tool support
• What about a hard-disk failure? Are you protected?
  – How often do you back up your development machine?
• What if you are working on a desktop at school/the office, and want to continue to work at your home desktop?
  – Email the code?
  – Carry a USB-stick?
  – Do you have a good naming scheme?
Assume now, you are developing in a small team. The team uses a regularly backed-up file-sharing mechanism (such as a shared network drive). What can go wrong?
Problems for Team Non-Version Controlled Software

• The last one who saves “wins”
  – Blame game: “You wrote over my changes!”
    “All my work is lost!”
  – How do you coordinate who can save a file at what time?
  – How do you coordinate with 5 or 10 or 100 or 1000 developers on the team?

• Work gets artificially sequentialized
  – “I’ll wait before I can touch that file. Who knows what will happen?”
  – Could potentially work on separate parts of a file, but merge by hand is error-prone

• Still no ability to revert back to a prior version of a particular file
Problems for Team Non-Version Controlled Software

• Still difficult to freeze one version for deployment and continue to work on a new version
  – Need “configuration manager”
  – Many ad-hoc solutions that quickly fall apart due to lack of tool support

• What if you are working on a desktop at school/the office, and want to continue to work at your home desktop but have no access to the file share?
  – Now it gets downright dangerous to carry the code home with you: the scope for conflicts/artificial sequentialization grows rapidly
  – How do you merge back your changes into the shared file?
Solution: Version Management with Dedicated Repository

Introduce a Version Management Repository that holds the **master copy** of all artifacts (code, tests, build files, graphics, ...) of your project.

![Diagram of a repository](image-url)
If Alice needs access to the code, she “checks out” a local copy.
Solution: Version Management with Dedicated Repository

If Bob also needs access to the code, he also "checks out" a local copy
Bob and Alice **individually** make changes to their own **local copy**.
Solution: Version Management with Dedicated Repository

When Alice is done with her edits, she “commits” her local copy to the repository. Her changes are automatically merged into the master copy of the file in the repository.
Now Bob is done with his edits, he also wants to "commit." What can happen?
Scenario 1: Alice and Bob have edited in **different** areas of the file – **No conflict**!
Bob’s changes will be merged into the master copy.
Solution: Version Management with Dedicated Repository

Scenario 2: Alice and Bob have edited in the **same** areas of the file – **Conflict**!
Bob’s changes will **NOT** be merged into the master copy.

![Diagram showingConflict due to overlapping edits by Alice and Bob.](image)
Solution: Version Management with Dedicated Repository

To resolve the conflict, Bob first “updates” his local copy.
He gets notifications of the conflicts **inside the affected files**. He makes decisions to resolve the conflicts.
Once he has resolved all conflicts, Bob commits again. This time he succeeds!
Alice updates at the beginning of her next work increment.
SVN Workflow: Always Update **BEFORE** Starting Work
SVN Workflow: Do something!

- Update
- Make Changes
- edit, add, delete files
SVN Workflow: Update **BEFORE EACH** Commit

- Update
- Make Changes
- Update

See if any conflicts have occurred.
SVN Workflow: Resolve Conflicts

1. Update
2. Make Changes
3. Update
4. Resolve Conflicts
   - communicate, decide, modify
SVN Workflow: Commit

1. Update
2. Make Changes
3. Update
4. Resolve Conflicts
5. Commit

be happy!
SVN Workflow

- Update
- Make Changes
- Update
- Resolve Conflicts
- Commit
Let’s revisit these steps with *subversion* commands
Create subversion repository

1> ls ~/java/week2/svenx
branches/ tags/ trunk/

1> ls ~/java/week2/svenx/trunk
F.java G.java ...

1>svnadmin create /var/svn/cse70-labs
Solution: Version Management with Dedicated Repository

A new repository, named “svn” has been created (can link to it via Apache web server)

svn repository @
https://cse70-server.ucsd.edu/svn/
Now Alice **imports** her existing folder (and its files) into the repository

```
1> svn import ~/java/week2/svenx \
https://cse70-server.ucsd.edu/svn/alice/svenx
```
Solution: Version Management with Dedicated Repository

Now Alice needs access to check out her fresh local copy to synchronize with the repository.

```bash
1> svn checkout \\
https://cse70-server.ucsd.edu/svn/alice/svenx .
```

./trunk:

F.java

Alice

check out
Bob also **checks out a local copy** into his local “wow” folder

```
1> svn checkout https://some.server.name/svn/trunk .
```

![Diagram showing Alice and Bob with F.java files]

Alice

Bob

check out
Bob and Alice **individually** make changes to **their own local copy**
Solution: Version Management with Dedicated Repository

Now Alice commits (publishes her edited files)

1> svn commit F.java
Scenario 2: Bob updates – Conflict!

```bash
1> svn update

C  F.java

Updated to revision 233.
```
package today;
public class StringFormatter {
    StringFormatter();
    public String formatString(String in) {
        char[] cv = new char[2*in.length()];
        for(int i = 0; i < cv.length; i+=2) {
            cv[i] = in.charAt(i/2);
<<<<<<<<< .mine
            cv[i+1] = CharConstantHolder.STAR;
        ======
            cv[i] = CharConstantHolder.STAR;
>>>>>>>.r233
        }
        return new String(cv);
    }
}

Bob looks at the file to find and resolve the conflict

I better chat with Alice about this
package today;
public class StringFormatter {
    StringFormatter(){};
    public String formatString(String in) {
        char[] cv = new char[2*in.length()];
        for(int i = 0; i < cv.length; i+=2) {
            cv[i]   = in.charAt(i/2);
            cv[i+1] = CharConstantHolder.STAR;
        }
        return new String(cv);
    }
}
Bob indicates that he has resolved the conflict

```bash
1> svn resolved F.java

Resolved conflicted state of 'F.java'
```
Once he has resolved all conflicts, Bob commits again. This time he succeeds!

```
1> svn commit F.java
```
Solution: Version Management with Dedicated Repository

Alice updates at the beginning of her next work increment and the cycle repeats.

```
1> svn update
```

![Diagram showing version control process](image)
What else can you do with subversion?
Other Features of subversion

- **Create Branches**
  - Isolate changes to separate lines of development
  - Example: one branch is the currently deployed version for maintenance, another branch is the current development version
  - Can merge branches, e.g.

- **Create Tags**
  - Similar to a branch
What have you learned today?
Learning Goals

- Be able to identify key elements of the software development pipeline
- Understand a basic Java program and how it is compiled/run
- Be able to change a given program to introduce functions and additional classes that can evolve separately
- Understand the need for a build tool
- See Ant used to automate compilation and other tasks
- Be able to write basic Ant files yourself
- Understand the challenges of maintaining a code base in an individual and team development setting
- Understand the need for and basic principles of version control/configuration management
Stage 2: Individual, w/IDE, Local Build/Test Tools and Version Control
Stage 4: Small to Medium Teams w/Automated Build Process