CSE 486/586: Distributed Systems

Software Development for Coursework

Ethan Blanton

Department of Computer Science and Engineering
University at Buffalo

Developing for Coursework

Both like and unlike:

- Developing a product
  - Concrete goals
  - Measurable success/failure
  - Schedule
- Research
  - Little to no post-development lifecycle
  - "Good enough" is good enough
    - ... but don't develop to the tests!
  - Few developers

Danger Will Robinson!

I have not tested tested *or even tried to compile* any code in these slides. It is intended to exemplify good practice for our discussion, not to be used in implementation. Be prepared to correct, debug, and extend it.

In particular, it does not handle errors.

(Sort of like Knuth, except I didn't even prove it correct.)

Caveat lector.

Take the Best of Both Worlds

Do:

- Develop with rigor
- Document at least minimally
- Write tests
- Consider idiomatic approaches
- Consider novel approaches
- Use abstraction

Don't:

- "Build it to throw away"
- Document when you should be coding
- Get lost in design patterns
- Neglect code quality

Examples

If you see something that looks like your code here…

don't be embarrassed!

If it's here, I saw multiple students doing it.

Version Control

Use version control!

Not this:

```
// os = new
    ObjectOutputStream(s.getOutputStream());
// os = new
    DataOutputStream(s.getOutputStream());
os = new
    BufferedOutputStream(s.getOutputStream());
// os.writeBytes(data);
os.writeUTF(data);
```

Style

Write good code.

If you hack something in to try it, fix it up when it works!

Comment subtleties.

Document interfaces.

Data Structures

Do this:

```
class Message {
    int priority;
    int sender_id;
    String body;
}
```

Not this:

```
String[] message = new String[]{
    Integer.toString(priority),
    Integer.toString(sender_id),
    body
};
```

Abstraction

```
Socket connectToHost(InetSocketAddress addr)
    throws HostFailedException {
  Socket s = new Socket();
  s.connect(addr,  TIMEOUT);
  s.setSoTimeout(TIMEOUT);
}

void sendToHost(Socket s, Message m) ... {
  s.getOutputStream().write(m.serialize());
}

Message readFromHost(Socket s) ... {
  /* Read and parse message */
  /* Throw HostFailedException on failure */
}
```

Debugging

Brownian motion: “[T]he random motion of particles suspended in a fluid[.]” (Wikipedia)

Brownian motion is not a good debugging strategy.

- Understand why you are making changes
- Change code purposefully
- Iterate

Make Sure of Your State

Make sure you know what you are actually running.

Employ:

- `gradle clean`
- Build | Clean Project in Android Studio
- Build | Build APK(s) in Android Studio
- `adb uninstall`
- `adb install`

More than one inexplicable bug has been “fixed” this way!

Aggregate Logs

Debugging a distributed system is hard.

Emit purposeful log messages.

Include:

- Ordering information
- Logical transitions
- Host/message/etc. identifiers

Consider writing log processing programs.

- Aggregate logs from multiple hosts
- Identify ordering violations
- Verify state transitions
- *etc.*

Debugging Overhead

Remember:

Time spent writing debugging tools is only wasted if it takes more time than ad-hoc methods.

Hint: it often does not.

If you've spent "days" on a problem: start writing tools. (Preferably days ago)

Java Iteration

```
String[] ports = { "11108", "11112", "11116",
    "11120", "11124" };

for (String port : ports) {
    /* Use port */
}
```

Collections using compareTo()

Java collections expect `compareTo()` to be:

- Stable
- Transitive

If `A.compareTo(B)` is -1 now, it should be -1 later.

```
class Foo {
    PriorityQueue pq;

    void frobnicate(SomeType obj) {
        pq.remove(obj);
        obj.changeSortKey();
        pq.add(obj);
    }
}
```

Use Logging Tags and Priorities

```
Log.e(TAG, "Error: debugging statement");
```

Change `TAG` to something suitable, then use:
`adb logcat <tag>:V '*:S'`

Six priorities of logging:

- Verbose (`Log.v()`)
- Debug (`Log.d()`)
- Info (`Log.i()`)
- Warning (`Log.w()`)
- Error (`Log.e()`)
- Fatal (`Log.f()`)

Streams and Buffering

For any given underlying stream (*e.g.*, `Socket`):

- You can connect at most one buffering reader
- You can connect at most one buffering writer

Consider:

```
ois = new ObjectInputStream(s.getInputStream());
dis = new DataInputStream(s.getInputStream());
len = ois.readInt(); // How much did it read?
message = dis.readLine();
```

Formatting

Use robust framing!

For PA 2B, what if I sent:
:-&@;$!%^~*-_=+‘”

How would your implementation handle that?

Better framing:

1. Send length (`short`, `int`, *etc.*)
2. Send UTF-8 bytes