Software Lifecycle

*CIS*2750*

Professional Aspect of Software Engineering

Software Lifecycle

- What is it? (aka “life cycle” “life-cycle”)
- Why introduce it now?
  - Helps put our work {design, coding, testing} into the “big picture” of the modern professional SW engineering process
  - Topic “factored out” of 3750/3760 so you can take them in either order
  - Preparation for group projects & co-op context

Compare “Human Lifecycle”

- Society has many institutions to care for us from cradle to grave:
  - Maternity hospitals, midwives
  - Daycare, nursery school, kindergarten
  - Elementary, high school, colleges, universities
  - Doctors, nurses, nursing homes
  - Retirement communities, assisted living
  - Hospices
  - Funeral homes

and many more!

Does *software* have a "life"?

- **Programmers** tend to focus on one event:

  ☞ **"birth"**

- But **SW engineers** know that there is money to be made caring for *every* phase of a program's life, not just birth
  - Pre-birth: requirements, design
  - Post-birth: customer support, maintenance
- A program can live a long time! (Y2K problem)
  - Being fixed up, enhanced, finally replaced

Comparison to other products

- Software does not “wear out”
  - Other products eventually wear out
  - And, the more complex, the *faster* they wear out (in general)
- Software is very complex, yet never wears out! But may become…
  - incompatible, obsolete, unsupported, unfashionable

“Legacy” Software

- People tend not to throw out old software
- Some people keep running old software (and expect compatibility and support)
- In some domains, the value of extensively tested code outweighs possible future innovation (NASA, banks, …)

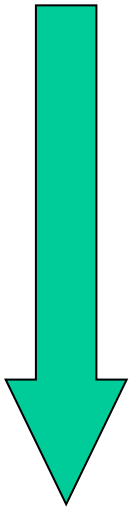
Many *views* of SW lifecycle

- Each view of lifecycle coupled with strategies for *managing* the different stages
- Classical "waterfall" model
  - Identifies very distinct stages
  - Simple, satisfying, but often impractical
- *Alternative models:* incremental, evolutionary, prototyping, spiral, Agile, OO (Rational Unified Process), embedded system process, many more

Waterfall Model

- Still good for pedagogy
  - Strongly documents-based

1. Requirements analysis & specification
2. System design & specification
3. Coding & module testing
4. Integration & system testing
5. Delivery & maintenance



Fundamentals of SW Eng, Ghezzi et al

1. *Requirements* analysis & specification

- **What does ___ want the SW to do?**
- **Analysis** = finding out and writing down
  - What manual work flow will be automated?
  - What data will be input, stored, and output?
  - What operations must be performed via the SW?
  - Constraints (performance, cost, schedule, etc.)
- **Specification** = documenting the requirements
  - Systematically, using precise language/diagrams
  - Critical systems employ **formal specifications**

2. *System* design & specification

- **How do we propose to build the SW?**
- **Design** = planning how to use SW technology to meet the **requirements**
  - Language, components (GUI, database, web server, etc.), web services
  - "Buy or build?" "Serial or parallel?"
- **Specification** = documenting the design
  - Procedural: structure charts, pseudocode, etc.
  - OO: UML, class diag., collaboration diag.

Pause here: bidding on a job

- If you want to win a customer's "call/request for proposals"(RFP) …
  - Need to understand enough about their requirements to put together a **rough design**
  - You can "cost out" the design by estimating:
    - Cost of buying components, coding, integrating, etc.
    - Cost of personnel, consultants, subcontractors, tools
    - Cost of training, warranty support & maintenance
  - Your design proposal + price + schedule form **your proposal** (your bid)

3. Coding & module testing

- Building the SW according to the **design** to meet the **requirements**
- Programmers must follow design spec precisely, so that their module interfaces are respected
  - Preconditions, postconditions, error handling
- Unit testing → your modules in isolation
  - White box testing to exercise all logical paths

4. Integration & system testing

- Assembling the pieces according to the **design**
  - Top-down vs bottom-up integration
- Testing against the **requirements**
  - Independent test team can construct black box test cases purely from the requirements spec
- Formal acceptance testing
  - Often, company does not get paid until customer signs off on approval

5. Delivery & maintenance

- Turning over the SW
  - May require physical installation and configuration of equipment and media
  - May involve training in its use
  - Customer support: on site, telephone, e-mail
- Maintenance needs excellent records!
  - Logging and investigating reported defects
  - Negotiating about enhancements
  - Assigning the work to developers
  - Carefully testing any changes (regression testing, too)

Configuration Control & Quality Assurance

- Background activities that help *all* stages of SW development process run sanely
- Version control (subversion, git, etc.)
- Bug/change tracking system
- Formal releases of versions
- Managers can glean stats from above to monitor progress
  - Defects/month reported vs. fixed

Waterfall Model: where are *we?*

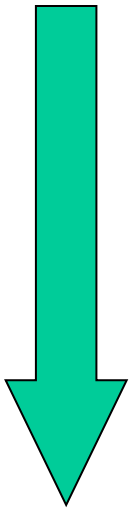
my part

1. Requirements analysis & specification
2. System design & specification

3. Coding & module testing
4. Integration & system testing

your part

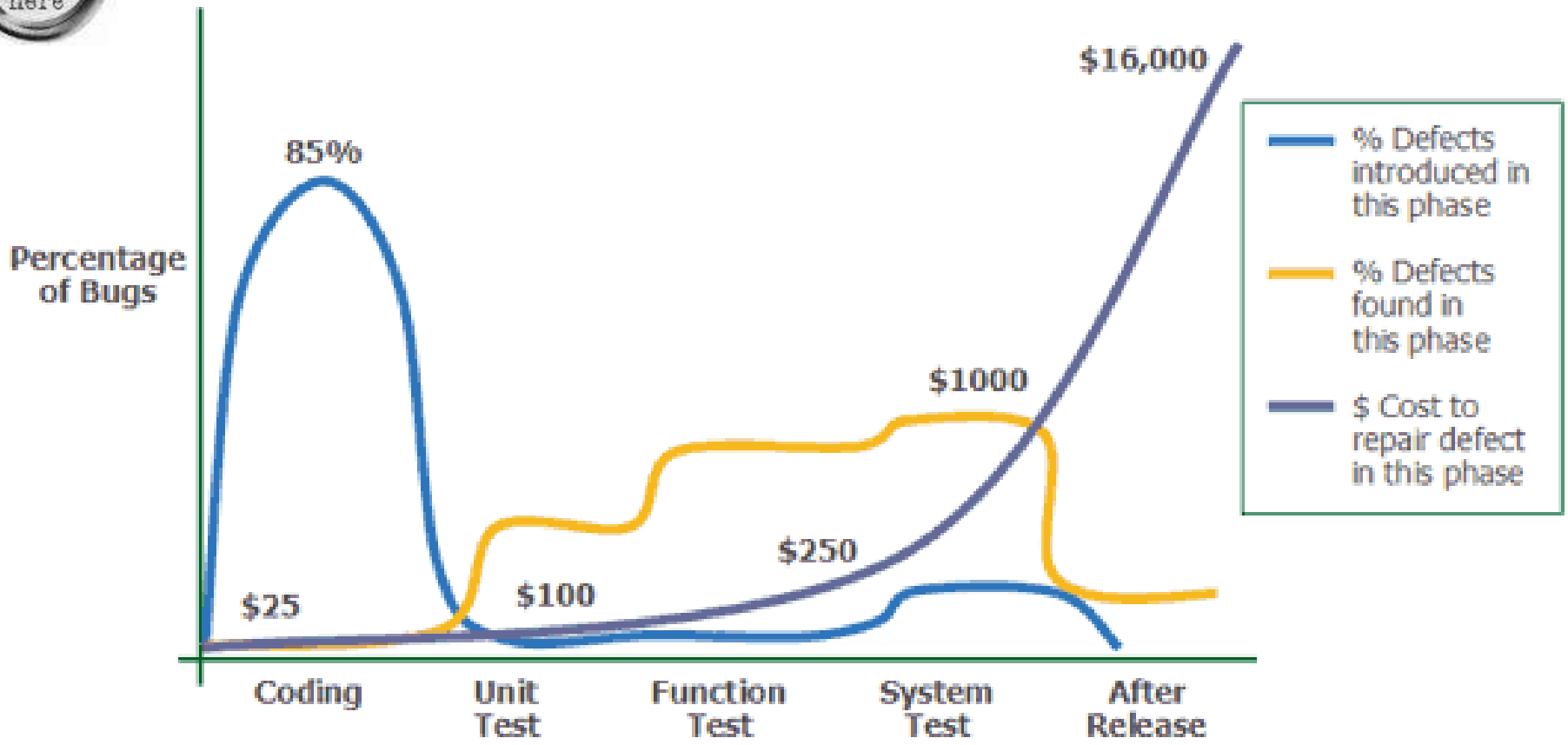
5. Delivery & maintenance



Waterfall's feedback loops

- H_2O flows strictly downhill, but in practical lifecycle, each stage can *feed back* to earlier
  - Constantly refining what the customer wants
  - Discovering gaps in knowledge about the data
  - Finding limitations in the components we decided to buy
  - Improving inefficient algorithms
  - Refactoring OO classes for prospective reuse

Why *quality* important at each stage



from Agitar Technologies (unit testing products)

Bottom line

- Do you want to be *proud* of the software systems you develop?
- Or go into hiding?

- Our mission in SOCS is to train *competent* professionals who can contribute to *quality* in all phases of the SW lifecycle, working in *teams*
  - *whether it's video games, antilock brakes, consumer electronics, telecommunication satellite, banking, healthcare database ...*

Parting (Scary) Thoughts

"Hackers [with laptops] find ways to hijack car computers and take control"
Financial Post, Sep. 3, 2013

- Demoed slamming on brakes, jerking steering wheel, shutting down engine
- Invasion route: cell phone, Bluetooth, CD player, tire pressure monitoring system
- Cars have 20-70 integrated computers!

“Hacking expert” cracked US Obamacare website in 4 mins.

Washington Times, Jan. 19, 2014

- David Kennedy gained access to 70,000+ personal records with standard browser
- Says Healthcare.gov is “100% insecure”
- Program size “**500 million** lines of code” reported in Congressional hearings