CPSC 491

Lecture 22: Wrapping Up …
Final Exam Overview
Design & Development Principles
MFT Exam

Remainder of semester ...

Today: Finish up with version control

Thursday: Exam 2

Next Tuesday: Design principles + Demo day

Next Wednesday: CEDE presentations

Next Thursday: More on design principles / lab day

Final: Exam 3 + HW due + CEDE status reports due
Final Exam (Thursday at 3:30)

Basics:
- About 6 multipart questions
- Worth 10% of final grade
- Closed book, notes, etc.

Topics:
- Basic Software Engineering Terminology (E1)
- Requirements (E1)
- Software process models and Agile/Scrum (E1)
- Estimation and Planning (E1, E2)
- Code Review and Testing Basics (E2)
- Testing Techniques (E2)
- Principles & Practices (today)

Exercise ...

With a partner, brainstorm what you think are “principles” / “best practices” in:
- Software engineering, including
- Software design
- Software development

Write down what you come up with on a piece of paper with your names
Design & Development “Principles”

Software Engineering methodologies come and go

- E.g., Waterfall, Spiral, RUP, RAD, XP, Kanban, Lean, Agile, Scrum, ...

But many of the underlying “principles” / “best practices” seem to endure / stick around

Examples: Software Engineering (generally)

1. **Figure out what you need to do before you do it**
   - Requirements (disagreements on detail needed)
   - Mock-ups, Prototyping

2. **Visibility is good**
   - keep customer involved, get feedback, iterate/prototype/refine

3. **Be organized and structured, but adapt**
   - Have a plan, follow it, change it as needed
Examples: Software Engineering (generally)

4. Keep track of where you are and where you’re going
   ○ e.g., milestones, roadmaps, wbs, ...
   ○ … and when you’ll get there (estimation)

5. Communicate
   ○ e.g., stand-up meetings, planning/review/retrospective meetings, documentation, etc.

Examples: Software Development (generally)

1. Use useful tools
   ○ Versioning, building, integrating, testing, tracking issues, ...

2. Test and review your software
   ○ Unit, integration, system, usability, stress, etc.

3. Consider maintenance
   ○ Part of “What you produce, others will consume”
   ○ Assume someone else will need to understand your work
Examples: Software Development (generally)

4. Focus on value
   - Decisions should be made based on providing value to users
   - Focus on the reason you are building the software

5. Do what is needed (“You aren’t going to need it”)
   - Don’t implement things until you actually need them
   - You probably won’t need it, or it will be different when you do
   - Won’t save time now by doing something you don’t need now

Examples: Software Design

1. Keep it simple, stupid!
   - As simple as possible, no simpler
   - Simpler is easier to understand, maintain, extend
   - Simpler is harder than “quick and dirty”

2. Have an (intentional) design
   - No design is not the opposite of over-engineered
   - Let the design evolve / improve it as needed (refactoring)
   - Avoid gold-plating
**Examples: Software Design**

3. Be a defensive programmer
   - Fail fast principle
   - Consider and mitigate what can go wrong (e.g., check inputs)
   - Opposite: Design by contract

4. Divide and conquer
   - Modularity (decomposition into smaller “modules”)
   - Avoid duplication (DRY)
   - Avoid unnecessary dependencies (loose coupling)
   - Use standard/vetted approaches (e.g., design patterns)

And lots more …