Software development processes; Intro to agile development

Dr. Kosta Damevski

CMSC 355 - Software Engineering: Spec and Design

Fall 2015

School of Engineering | Computer Science

Announcements

- Sign up to Piazza
  - Link via e-mail or on Blackboard
- Assignment 1 is due on Sunday @ midnight
  - I'll aggregate and post the list of apps
  - I'll formally announce group forming (next week)

Last Time

- We need software engineering to manage the complexity of designing software
    - A set of scientific principles and best practices
        - It's more than just programming
- Software fails and can be difficult to engineer
- ***Software development process (model)*** governs how we go from the idea to the product stage
    - mentioned waterfall, spiral and agile models
    - *continue this discussion today…*

Do we need a software process?

- Designing simple software (such as homework assignments) has two steps
  - Step 1: Think!
  - Step 2: Code!
- Both steps are creative
  - Programmers are happy doing them
  - Customers are happy to pay programmers to do this

Do we need a software process?

- The two step process doesn't scale up with complexity, for example:
    - How do we split the work among a team of people?
    - How do we ensure we know what the customer meant when they asked for feature X?
    - How do we ensure we give them only what they pay for?
    - What about cross-cutting concerns, such as security, accountability, performance, upgradeability etc?

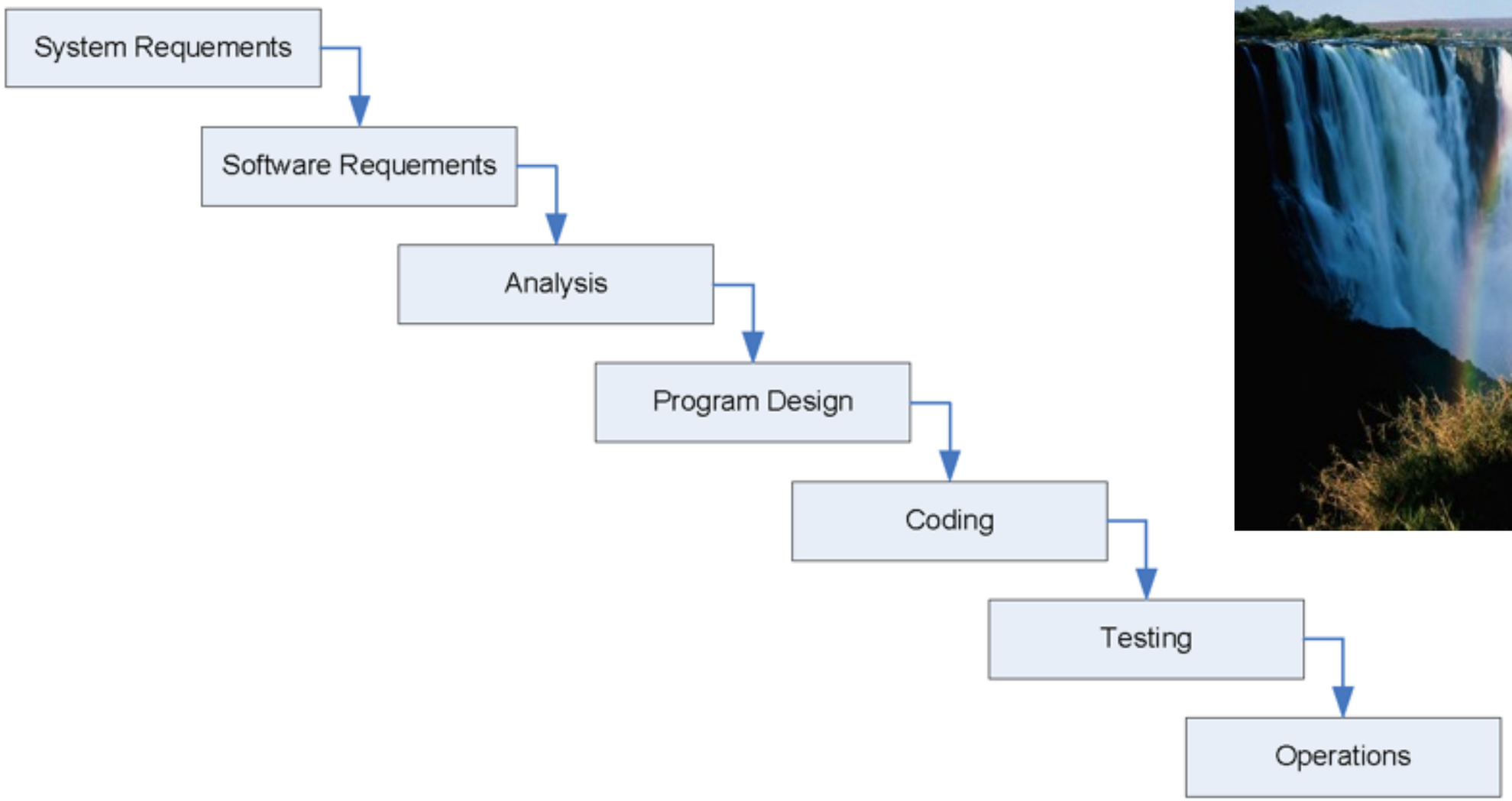
Stages in engineering software

- Typical stages in software engineering:
  - Requirements Analysis
  - Software Design
  - Implementation
  - Testing
  - Maintenance
- Some of these can be further split (e.g. system req's and software req's), and more can be added (e.g. deployment)

What is a software development process?

- Let's revisiting this question in the context of stages
- A software process model determines the order of the stages involved in software development and evolution
- It provides the answer to the following two questions?
    - What shall we do next?
    - How long shall we continue to do it?

1st Development Process: Waterfall (1970)



How well does the waterfall model work?

- *“And the users exclaimed with a laugh and a taunt: ‘It’s just what we asked for, but not what we want.’” — Anonymous*
- The Good
    - Simple!!!
    - Plenty of documentation, which is good (allows for management of project)
    - Still in use since the 70s
- The Bad
    - Testing is towards the end of the model, and it may expose fundamental problems, requiring rework
    - Customer is not involved

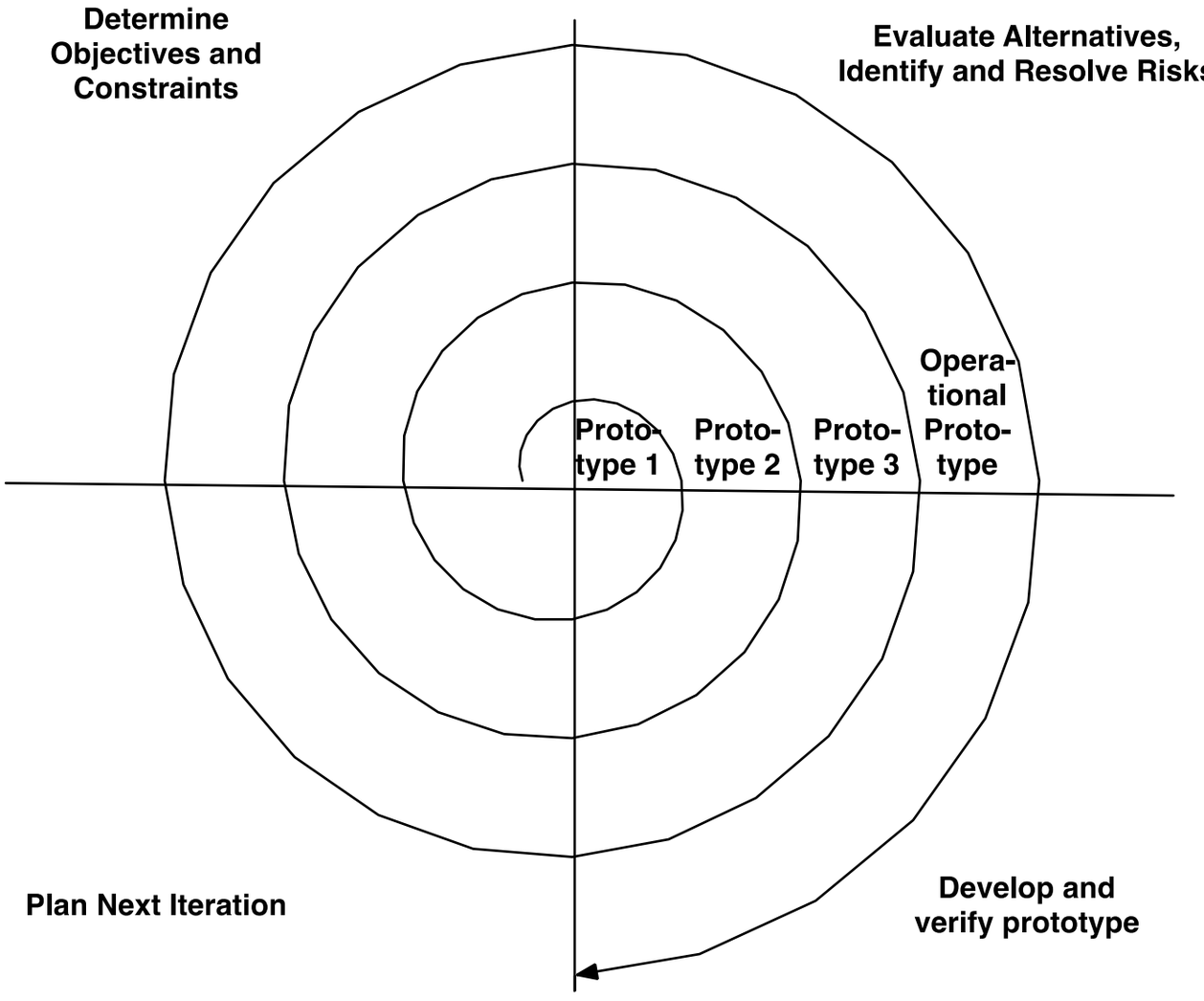
How well does the waterfall model work?

- *"Plan to throw one [implementation] away; you will, anyhow." — Fred Brooks, Jr. (1999 Turing Award winner)*
- Often after building first one, developers learn right way they should have built it

Spiral model (1986)

- Combine Plan-and-Document with prototypes
- Rather than plan & document all requirements 1st, develop plan & requirement documents across each iteration of prototype as needed and evolve with the project

Spiral model



Spiral model, good and bad

- The Good
    - Iterations involve the customer before the product is completed
        - Reduces chances of misunderstandings
    - Risk management part of lifecycle
    - Project monitoring easy
    - Schedule & cost more realistic over time
- The Bad
    - Iterations 6 to 24 months long
        - Time for customers to change their minds!
    - Lots of documentation per iteration
    - Lots of rules to follow, hard for whole project
    - Cost of process is high
    - Hard to meet budget and schedule targets

Agile development processes

Plan and document models failed often

- Often missing the cost, schedule, & quality target
- P&D requires extensive documentation and planning and depends on an experienced manager
  - Can we build software effectively without careful planning and documentation?
  - How to avoid “just hacking”?

Peres' Law

- *"If a problem has no solution, it may not be a problem, but a fact, not to be solved, but to be coped with over time." — Shimon Peres (winner of 1994 Nobel Peace Prize for Oslo accords)*

Agile manifesto

- “We are uncovering better ways of developing SW by doing it and helping others do it. Through this work we have come to value
- ***Individuals and interactions*** over processes & tools
- ***Working software*** over comprehensive documentation
- ***Customer collaboration*** over contract negotiation
- ***Responding to change*** over following a plan

That is, while there is value in the items on the right, we value the items on the left more.”

Agile development model

- Embraces change as a fact of life: continuous improvement vs. strict phases
- Developers continuously refine working but incomplete prototype until customers happy, with customer feedback on each Iteration (every ~1 to 2 weeks)
- Agile emphasizes Test-Driven Development (TDD) to reduce mistakes, written down User Stories to validate customer requirements, Velocity to measure progress

Extreme programming (version of agile programming)

- If short iterations are good, make them as short as possible (weeks vs. years)
- If simplicity is good, always do the simplest thing that could possibly work
- If testing is good, test all the time. Write the test code before you write the code to test.
- If code reviews are good, review code continuously, by programming in pairs, taking turns looking over each other's shoulders.

Agile Then and Now

- Controversial in 2001
  - *"… yet another attempt to undermine the discipline of software engineering… nothing more than an attempt to legitimize hacker behavior." — Steven Ratkin, "Manifesto Elicits Cynicism," IEEE Computer, 2001*
- Accepted in 2013
  - 2012 study of 66 projects found majority using Agile, even for distributed teams

Yes = Plan and Document; No = Agile

- Is specification required?
- Are customers unavailable?
- Is the system to be built large?
- Is the system to be built complex (e.g., real time)?
- Will it have a long product lifetime?
- Are you using poor software tools?
- Is the project team geographically distributed?
- Is team part of a documentation-oriented culture?
- Does the team have poor programming skills?
- Is the system to be built subject to regulation?

Question? Which statement is true?

1. A big difference between Agile and P&D is that Agile does not use requirements
2. A big difference between Agile and P&D is measuring progress against a plan
3. You can build Android apps using Agile, but not with Plan-and-Document
4. A big difference between Agile and P&D is building prototypes and interacting with customers during the process

References

- *“Engineering Software as a Service” by Armando Fox and David Patterson (2nd Edition)*
- *“Software Engineering” by Ian Sommerville (10th Edition)*

Questions