Software Development Processes: *Rapid Prototyping*

Software Engineering

Millersville University

The Flip Side: Advantages to Being Fast

- In the short-term, we can assume the world will not change
  - At least not much
- Being fast greatly simplifies planning
  - Near-term predictions are much more reliable
- Unfortunately, the waterfall model does not lend itself to speed . . .

Something Faster: Rapid Prototyping

1

Write a quick prototype

2

Show it to users

- Use to refine requirements

3

Then proceed as in waterfall model

- Throw away the prototype
- Do spec, design, coding, integration, etc.

Comments on Rapid Prototyping

- Hard to throw away the prototype
  - Slogan “the prototype is the product”
  - Happens more often than you might think!
- A prototype is useful in refining requirements
  - Much more realistic to show users a system rather than specification documents
- A prototype exposes design mistakes
- Experience building a prototype will improve greatly the accuracy of plans

Opinions on Reality

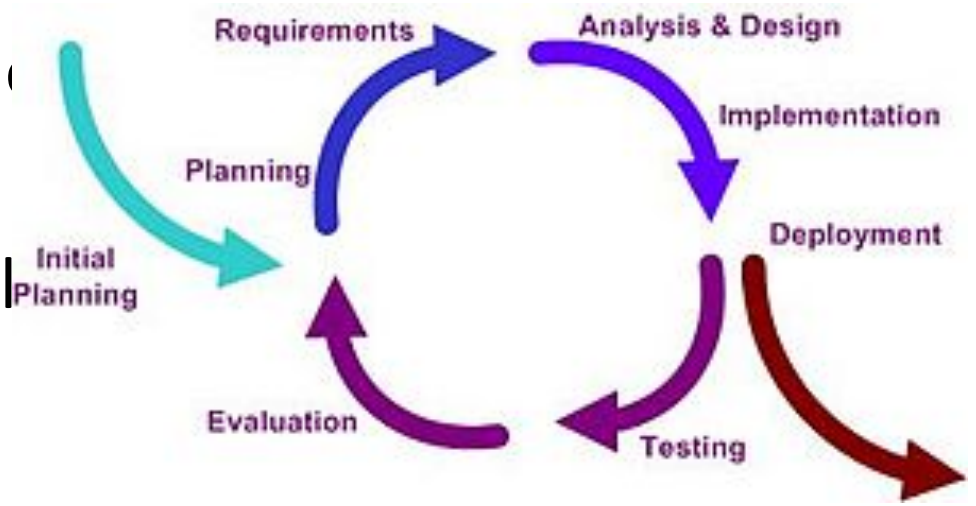
- Neither of these models is true to life
- In reality, feedback between all stages
  - Specifications will demand refined requirements
  - Design can affect the specification
  - Coding problems can affect the design
  - Final product may lead to changes in requirements
    - I.e., the initial requirements weren't right!
- Waterfall model with "feedback loops"

What to Do?

- Accept that later stages may force changes in earlier decisions
- And plan for it
- The key: Minimize the risk
  - Recognize which decisions may need to be revised
  - Plan to get confirmation/refutation as soon as possible

Iterative Models: Plan for Change

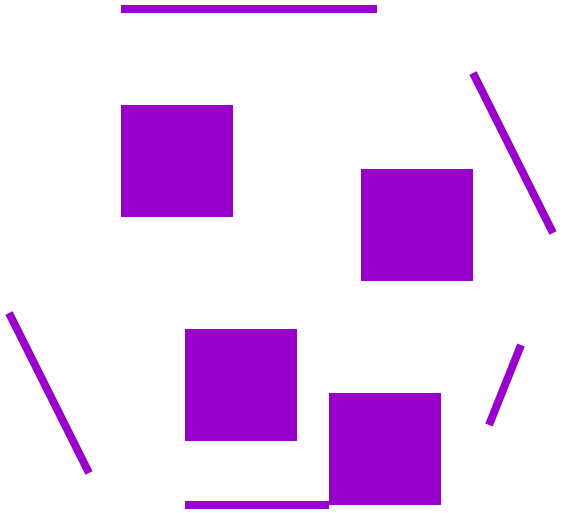
- Use the same stages as the waterfall mod
- But plan to iterate the whole cycle several
  - Each cycle is a “build”
  - Smaller, lighter-weight than entire product



- Break the project into a series of builds which lead from a skeletal prototype to a finished product

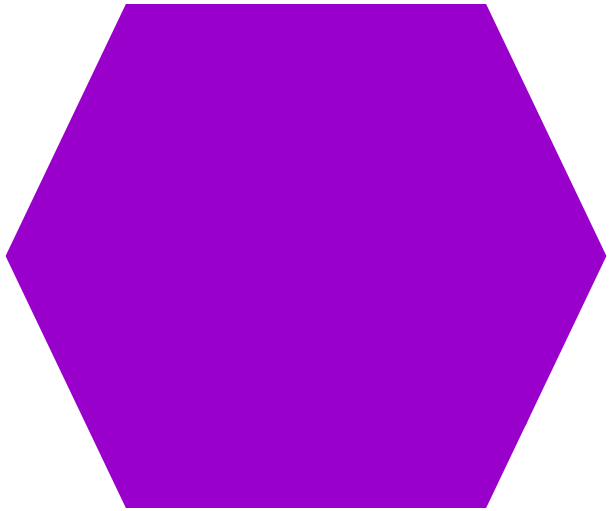
Gather Requirements

- Same idea as before
- Talk to users, find out what is needed
- But recognize diminishing returns
- Without something to show, probably can’t get full picture of requirements on the first iteration



Specification

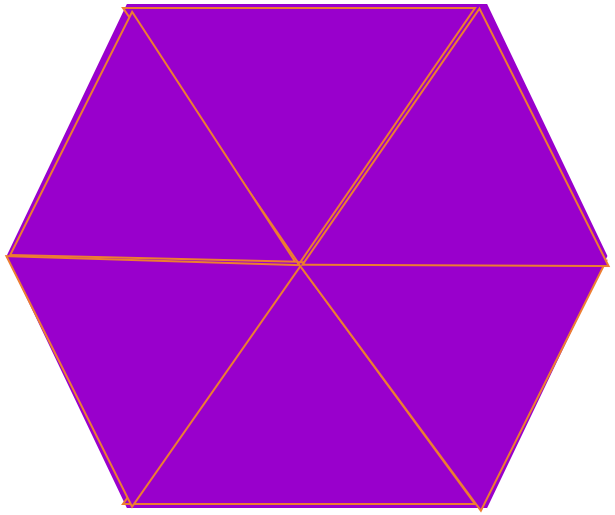
- A written description of *what* the system does
  - In all circumstances
    - For all inputs
    - In each possible state
- Still need this
  - Worth significant time
- Recognize it will evolve
  - Be aware of what aspects are under-specified



Design

- Decompose system into modules and specify interfaces

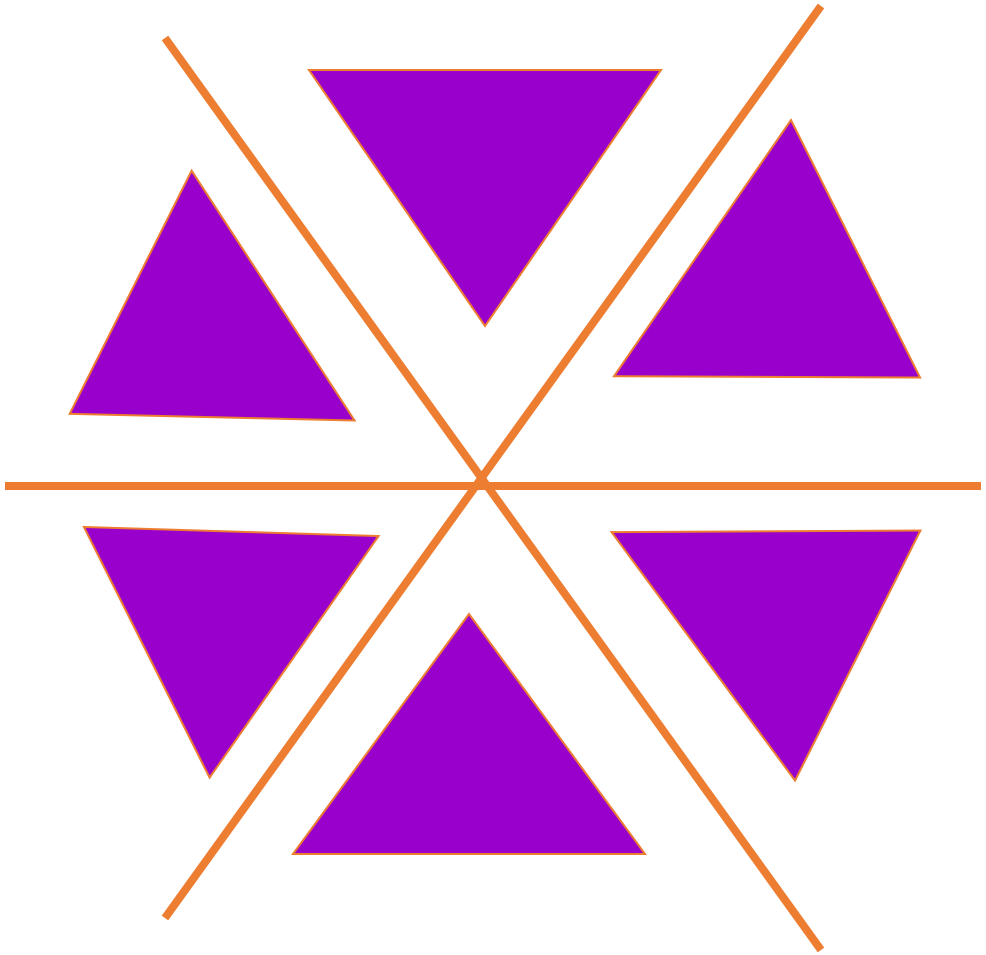
- Design for change
- Which parts are most likely to change?
  - Put abstraction there



Design

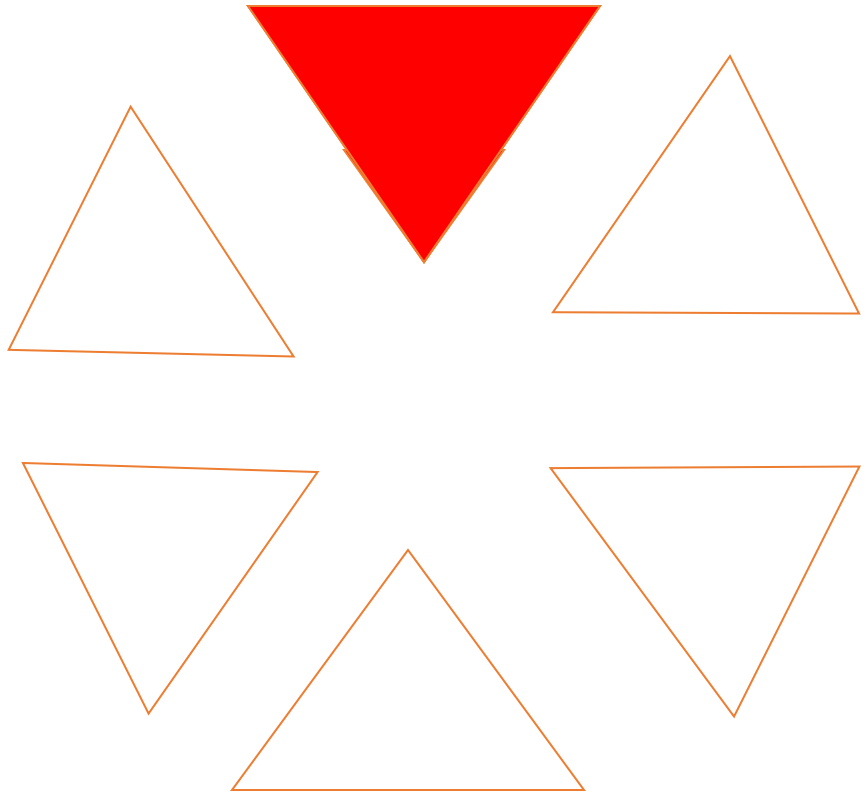
- Decompose system into modules and specify interfaces

- Which parts are most likely to change?
  - Put abstraction there



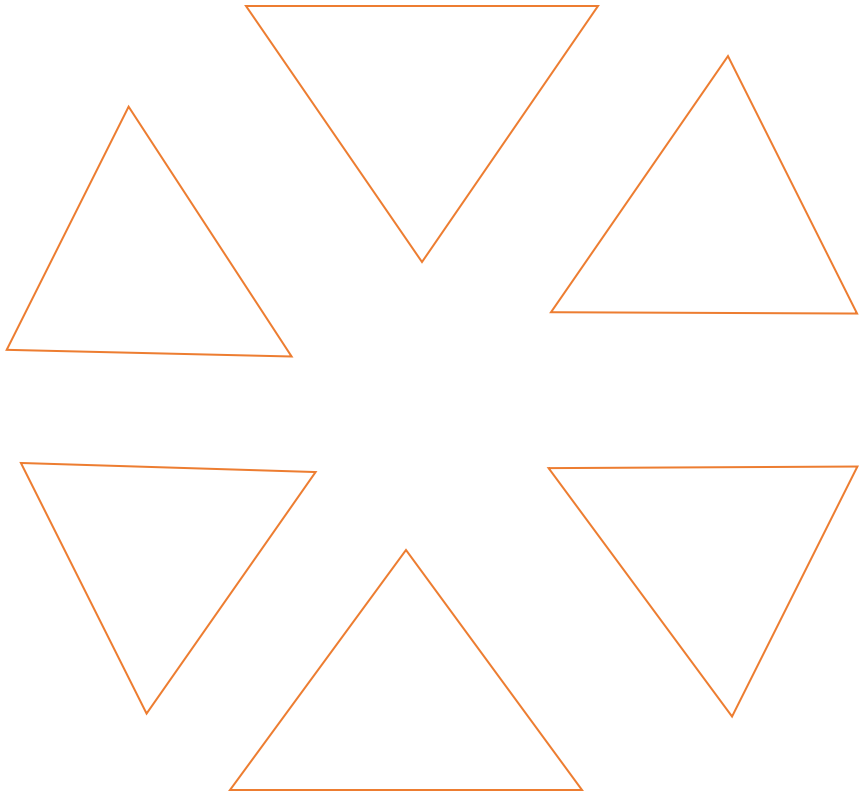
Design

- Plan incremental development of each module
- From skeletal component to full functionality
- From most critical to least critical features



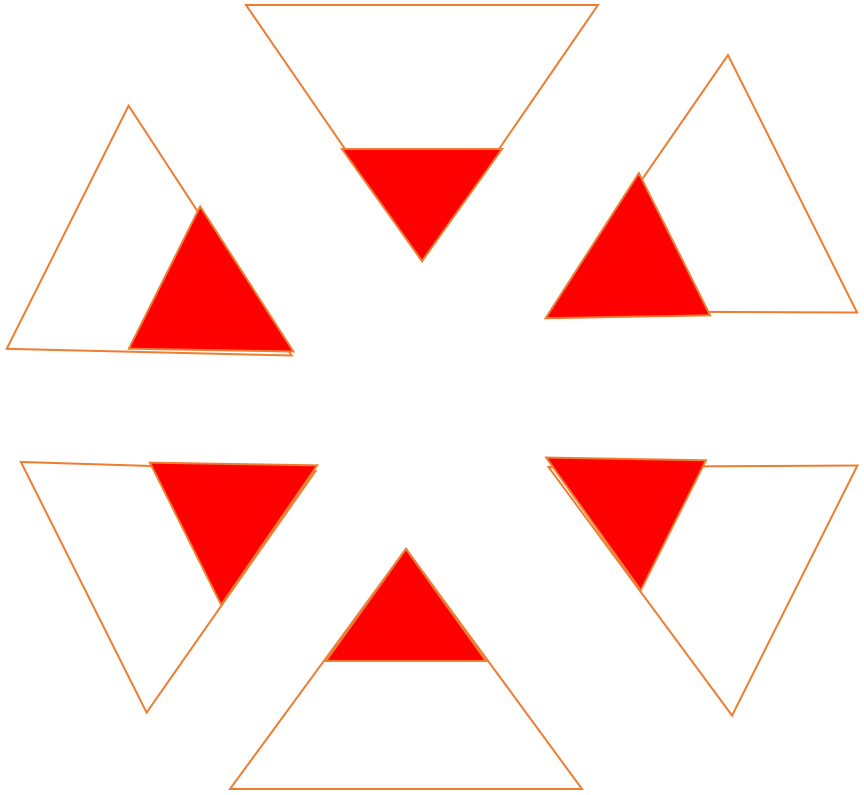
Implementation: Build 1

- Get a skeletal system working
- All the pieces are there, but none of them do very much
- But the interfaces are implemented
- This allows
  - A complete system to be built
  - Development of individual components to rely on all interfaces of other components



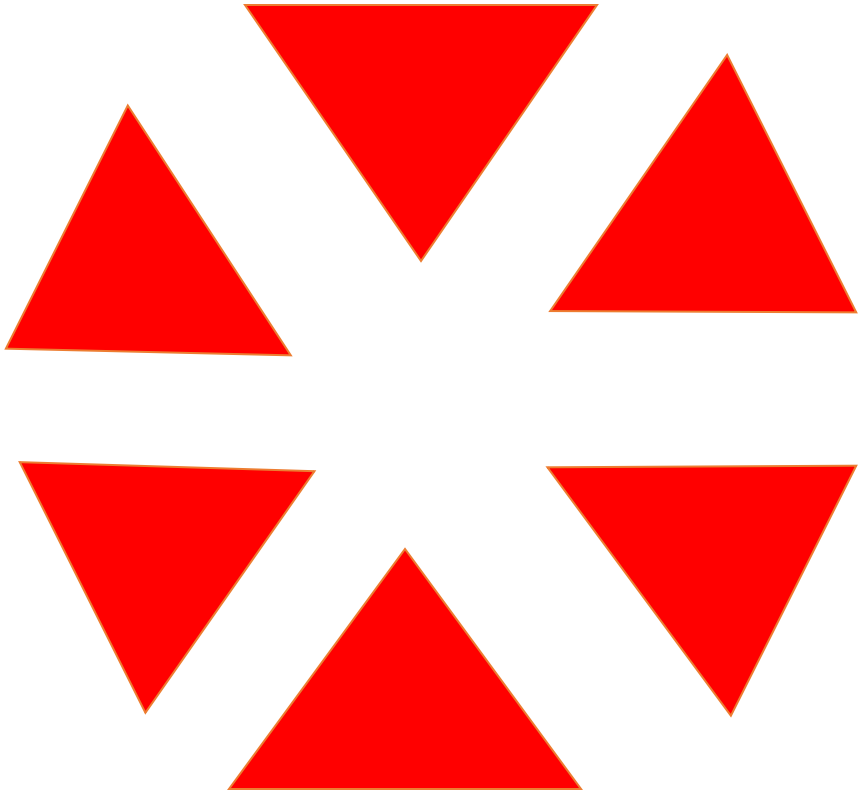
Implementation: Subsequent Builds

- After build 1, always have a demo to show
  - To customers
  - To the team
  - Communication!
- Each build adds more functionality



Integration

- Integration and major test for each build
- Stabilization point
- Continues until last build
  - But may begin shipping earlier builds



Advantages

Find problems sooner

- Get early feedback from users
- Get early feedback on whether spec/design are feasible

More quantifiable than waterfall

- When build 3 of 4 is done, product is 75% complete
- What percentage have we completed at the implementation stage of the waterfall model?

Disadvantages

Making a major Mistake

- In requirements, specification, or design
- Because we don’t invest as much time before build 1
- Begin coding before problem is fully understood

Trade-off against being slow

- Often better to get something working and get feedback on that rather than study problem in the abstract

In Practice

- Most consumer software development uses the iterative model
  - Daily builds
  - System is *always* working
  - Microsoft is a well-known example
  - IBM Rational Unified Process
- Many systems that are hard to test use something more like a waterfall model
  - E.g., unmanned space probes

Summary

- Important to follow a good process
- Waterfall
  - top-down design, bottom-up implementation
  - Lots of upfront thinking, but slow, hard to iterate
- Iterative, or evolutionary processes
  - Build a prototype quickly, then evolve it
  - Postpone some of the thinking
- Extreme programming, Agile process, next …