Week 3 - Wednesday

COMP 3100

Last time

- What did we talk about last time?
- Requirements management
- Requirements modeling
- UML
  - Activity diagrams
  - Use case diagrams
  - State diagrams
  - Sequence diagrams
  - Class diagrams

Questions?

Software Processes

Processes

- A **process** is a collection of actions that turns a set of inputs into a set of outputs
- Describing processes requires:
  - Specifying the inputs to the whole process and the outputs from the whole process
  - Specifying the actions of the process
  - Specifying the inputs to each action in the process and the outputs from each action
  - Specifying the conditions and order for each action
- UML activity diagrams are good ways to model processes

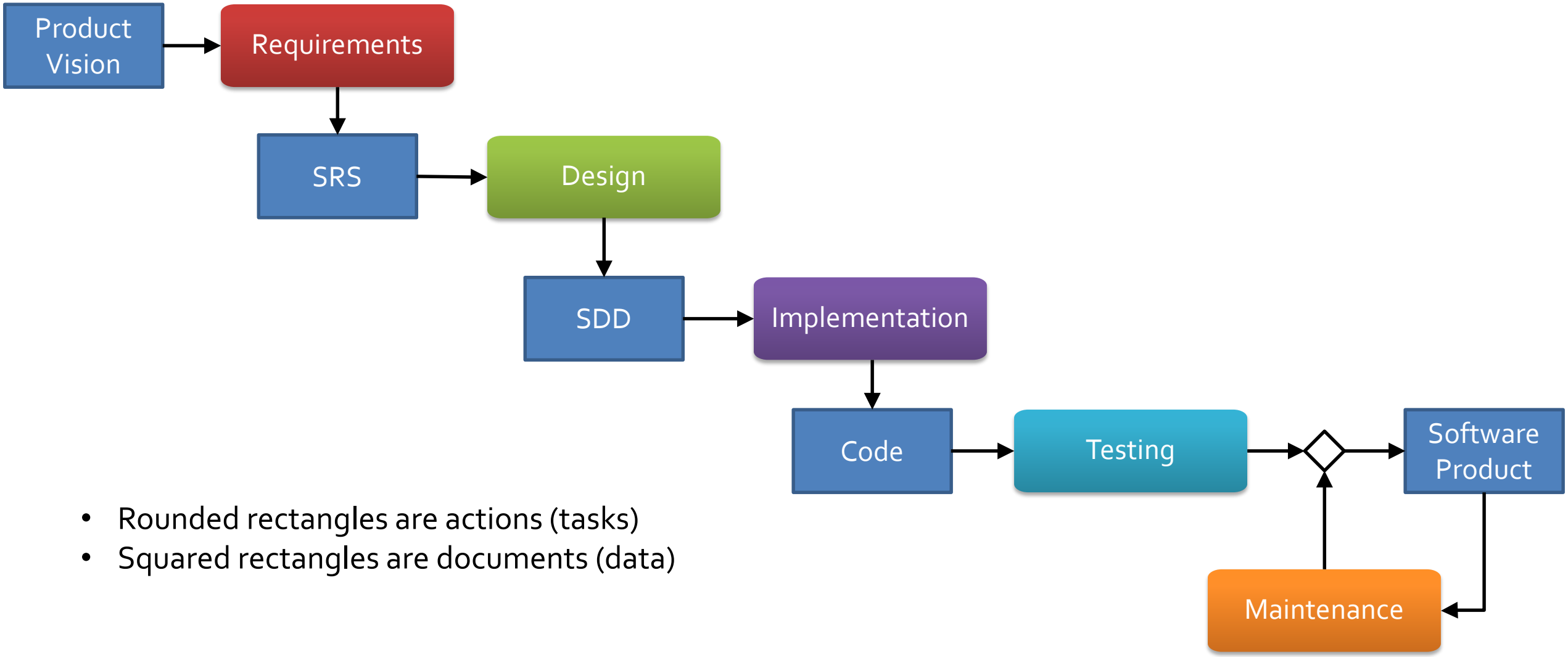
More terminology

- A **software process** is a process used to make or support software
- A **software lifecycle process** shows the steps from product inception to retirement of the product
- A **model** is an entity used to represent another entity (the **target**)
- A **software process model** is a model for a software process
  - Usually a 2D diagram like a UML diagram

Waterfall model

- The **waterfall lifecycle process model** is the oldest description of the tasks in the development of software
- Proposed by Winston Royce in 1970
- It follows similar processes in other engineering disciplines
- It's the usual approach we've been talking about, from requirements to design to implementation to testing to maintenance

Waterfall lifecycle model



- Rounded rectangles are actions (tasks)
- Squared rectangles are documents (data)

More on waterfall

1. Developers get a product vision
2. From it and interaction with stakeholders, they create a software requirements specification (SRS)
3. From the SRS, they create a software design document (SDD)
4. Using the SDD, they implement the code
5. Then they test the software product
6. When the software is in use, problems are found, leading to maintenance and a new release

- The name "waterfall" is because each action flows to the next
  - Like a series of waterfalls
- In principle, developers **never** return to an earlier action
- In practice, earlier actions must always be reexamined because you never get it perfect the first time
- Even so, the goal is to be a thorough as possible the first time

Advantages of waterfall

- The whole product is specified
- The project to create it is planned early
- This approach is important for large and complicated products from a management perspective
  - Size, cost, delivery dates, etc.
- By comparing to the plan, it's easy to tell if a product is on-time and on-budget
- If it isn't, managers can take actions
  - Increase time, increase budget, reduce scope, etc.

More advantages of waterfall

- If each step is done completely and correctly, all mistakes are found before moving on to the next step
  - This ends up being the major disadvantage of waterfall, too, since mistakes usually propagate to future steps
- Good documentation is created for each step
  - This is really important when new people are added to the project
- Each phase is distinct, allowing it to be carried out by teams that specialize in that phase
  - For multiple projects, appropriate teams can be scheduled for maximum efficiency

Disadvantages of waterfall

- Requirements can't change
  - But they usually do
  - If requirements change, all the advantages of waterfall's predictability disappear too
- Even when requirements stay the same, it's hard to be complete and consistent in documenting them
- Creating all the documentation for waterfall is expensive
- If you have separate teams for each phase, each team has to learn what has already been done

More disadvantages of waterfall

- Because there are so many teams, a lot of management is needed
  - Drives up the cost
  - Heavyweight processes are ones with a lot of documentation and management
- There's no product until completion of the entire project
  - Could take years
  - We don't realize the problems until the product is available
  - Clients might not want the product anymore

To waterfall or not to waterfall?

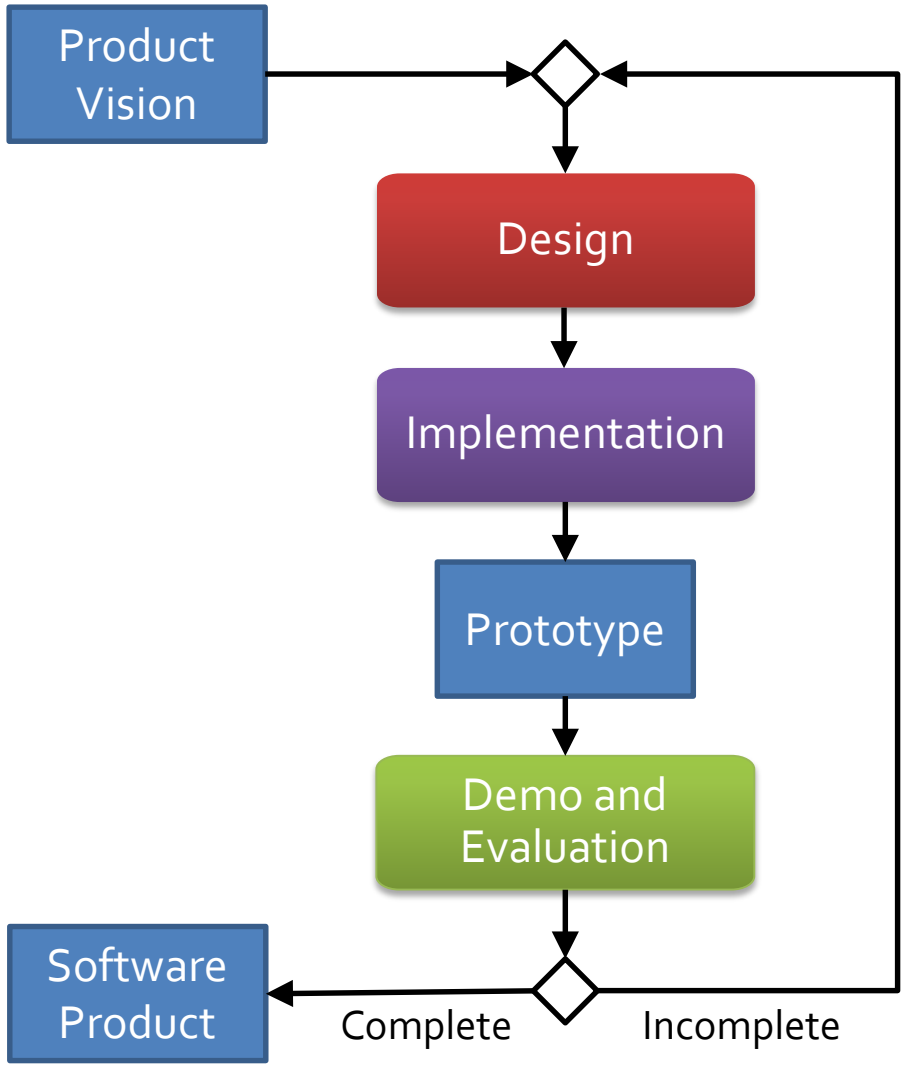
- Waterfall was the only process for a long time
- Its track record isn't great
  - Success only about 25% of the time historically, but the rate is improving
- Waterfall only works when the requirements are stable
- Waterfall has a lot of overhead
  - Might be justified for large projects
  - Isn't justified for small projects
- Use waterfall only for large projects with stable requirements or when there are very high safety, security, or reliability requirements
- … or when your professor makes you

Prototyping

- A **prototype** is a working model of a finished product
  - It can model a part or the whole
- Prototypes can help offset problems with the waterfall model
- Prototypes are particularly helpful with testing out UI decisions
- Prototypes are easy(ish) to make and change
  - Try out several!
  - See which one is the better design
- **Throwaway prototypes** are just used for making specifications and then thrown out
- **Evolutionary prototypes** are modified into the final product

Prototyping process

- Prototypes can be used within the waterfall model
- Or they can be used for an entirely prototype-based lifecycle model
- This idea is what incremental and agile processes are built around



Advantages of prototyping

- Changes to specifications are easy to handle
- Customers are more likely to get what they want (since they get regular opportunities for feedback)
- Customers can get (potentially) useful software quickly
- Not much documentation or management is needed
  - **Lightweight** development process

Disadvantages of prototyping

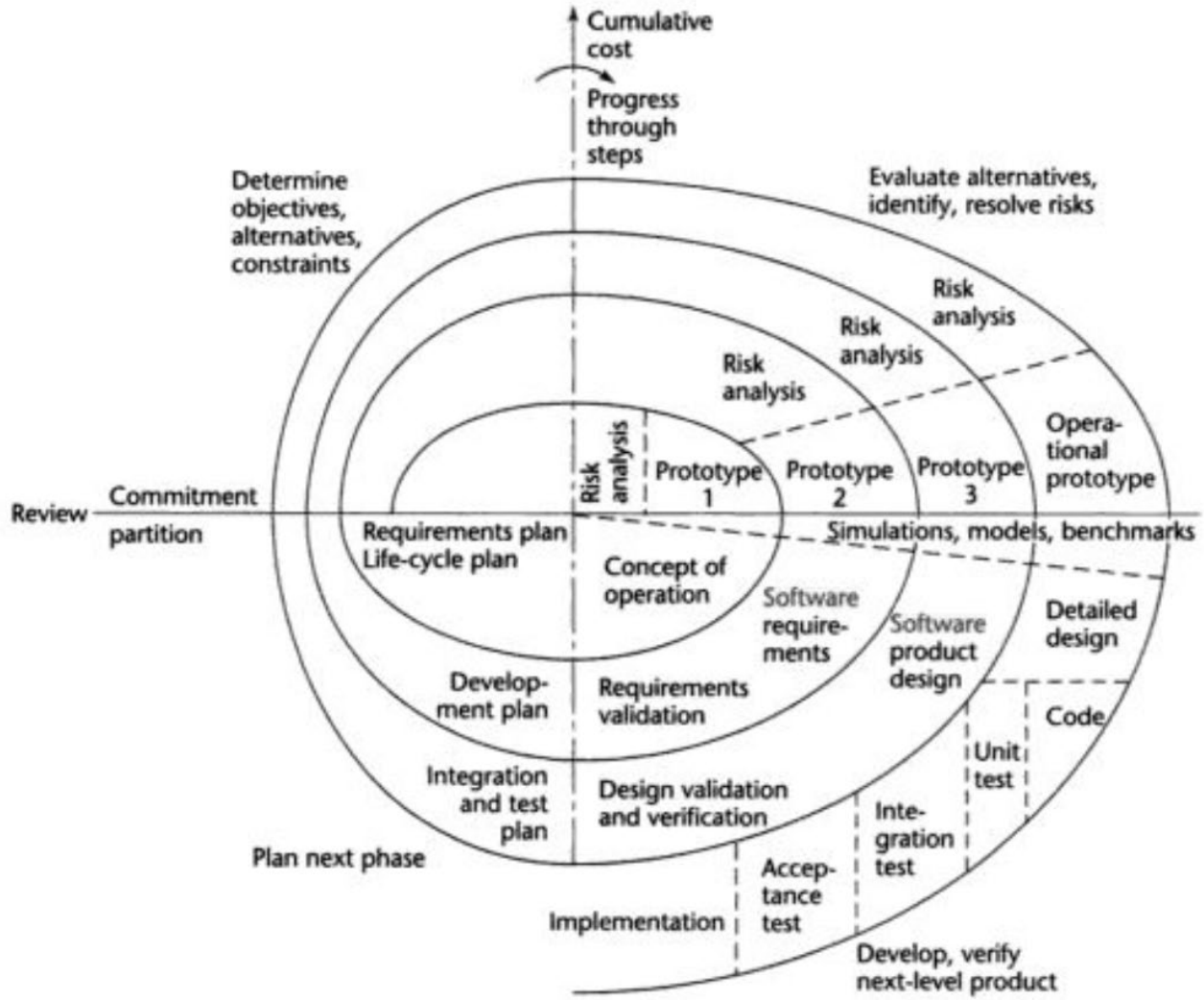
- Without the planning of a process like waterfall
  - It's hard to predict a reasonable deadline for the final product
  - It's hard to predict the budget
- Product design might be bad since the product evolved without following a plan
  - The biggest problem here is maintainability: How can new features be added?
- An undisciplined process can have poor quality control
  - The product might be unreliable or buggy

Risk management

- A **risk** is an event with negative consequences
  - Losing source code
  - Losing a team member
  - Finding an unexpected design flaw
  - Underestimating the time needed to write a piece of code
- Business people think about risk a lot
- Risk management is identifying, analyzing, controlling, or mitigating risks
- Risk management *should* be incorporated into all software lifecycle processes

Spiral model

- The spiral model is built around risk management
- Multiple cycles are used
- Each cycle starts by looking at goals
- Then evaluate different approaches to the goals in terms of risk
- The model on the right shows how the spiral model can be applied to waterfall



Drawbacks of the spiral model

- As with many of these models, the strengths and weaknesses are closely related:
  - The spiral model centers on risk management, but risk management is really hard
  - Few people have the necessary training or skill to properly evaluate risks
  - The spiral model is very general, requiring a lot of knowledge to make it work for software processes

Quiz

Upcoming

Next time…

- Friday is a work day
- Next Monday:
  - Iterative and incremental processes
  - Rational Unified Process
  - Agile processes

CAREER JUMPSTART EVENT

Engineering & Computer Science

THURSDAY, SEPTEMBER 12TH FROM 4:45PM-7PM

Otterbein University @ The Point

Come and network with alumni and recruitment partners and learn how to be successful with your field.

OTTERBEIN
UNIVERSITY

SCAN the QR CODE to REGISTER

Reminders

- Keep reading Chapter 2: Software Processes for Monday
- Finish your draft Project 1 by Friday