SDLCs and the waterfall model

- Careful management/organization needed for large software projects
- A handful of common models are used for the software development life cycle (SDLC)
- We’ll look at several models, each with advantages and disadvantages
- In this session we’ll start with the simplest model: waterfall

Waterfall model

- In the waterfall model, we regard the SDLC as a sequence of several major phases: you complete each phase in turn and never go back to it
- “Waterfall” refers to the notion of always flowing forward from one phase to the next, no way to go back
- Lots of discussion on the “right” way to decompose your life cycle into phases, we’ll go with a simple, relatively common breakdown

Phases

- Requirements – what is needed?
- Planning – what needs to happen/when?
- Analysis and design – figure out how we’ll build it
- Implementation – actually build/test it
- Deployment – put it into operation
- Maintenance – adjust as necessary for the rest of the product’s lifespan

Requirements

- We need to figure out exactly what it is the clients and end users actually want/need
- Often involves extensive interviewing, researching, discussion, interaction with the clients, users, and people who will be impacted by the product
- By the end of this phase, we want to have a clear, complete, unambigous collection of documents describing what is wanted/needed ... if the end product meets these requirements then it will be satisfactory to all involved

Planning

- We need to establish what needs to happen in the rest of the SDLC to bring the project to completion
- What people/resources are needed, and when
- What tasks need to be carried out (including future planning and management tasks) and when
- How will we establish a budget and timeline, and monitor if we’re on time and on budget

Analysis and design

- Requirements tell us what the clients/users want/need
- We need to figure out how we’ll actually build it
- Often involves many different parts (networking, databases, AI, graphics, sound, etc etc etc) each of which can be very complex
- Need to plan how we’ll decompose the problem into managable units and design them to solve the problem and work together

Implementation

- In analysis and design we mapped out what we were going to build
- Now we need to actually build the parts, test them, debug it, put the parts together into larger components and test/debug those, put those together into still larger components and test those, etc
- By the end of implementation we should have a product we think meets the requirements

Deployment

- Now that it is built, we need to put it into operation
- Might involve installation on various servers (network, database, etc), cloud support, installation of various support software/tools, client-side installations, configuration, data entry and conversion, etc
- Often a very complex task, using a variety of programs, scripts, and tools to automate as much as possible (of course, each such program/script also needs testing)

Maintenance

- Once operational, we need to keep the product operational
- The environment it works in may change frequently (operating system upgrades, changes to software the product interacts with, regulatory changes, etc), which may require alterations to the product
- The client and user needs/wants may change over time, which again may require alterations to the product
- Bugs may be noticed over time, which again may require alterations to the product

Strengths, weaknesses

- Waterfall is a very simple model, easy to manage, and easy to keep track of whether we’re on schedule or not
- Unfortunately, relies on the idea that we get each phase right the first time through, and never need to go back to it
- Not very realistic: in most projects we need to revise/clarify requirements during design, revise/clarify design during implementation, etc
- Works for small, low-risk projects where the team is well experienced with the kind of product they’re creaing