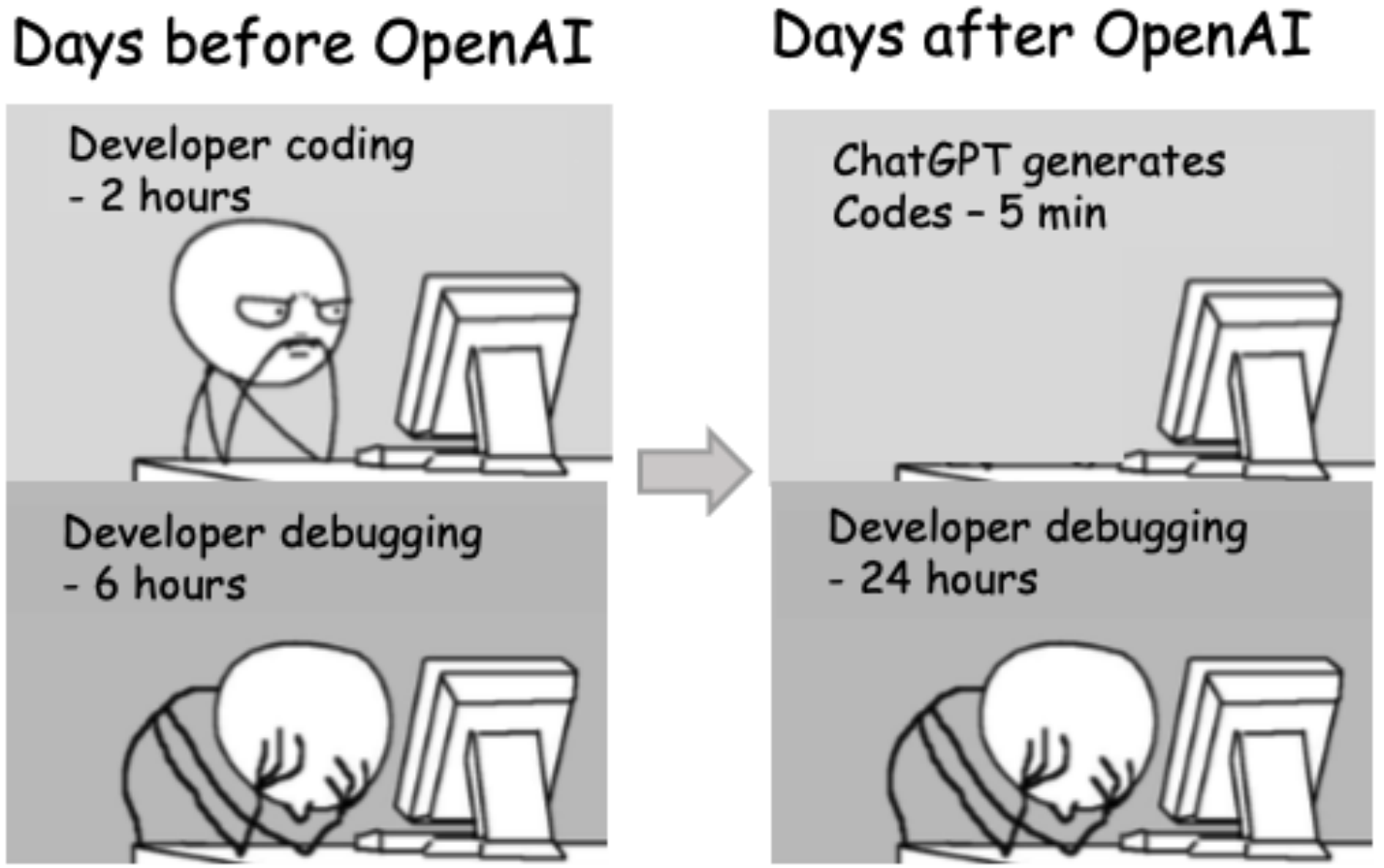
3. Software Development

Software Development



Software Development Workflow

- Requirements 
- Analysis and Design 
- Implementation 
- Testing 
- Deployment and Maintenance 

We will briefly introduce each topic (more in later lectures)

Requirements engineering

- Three components:
  1. Requirements elicitation: *What do the stakeholders/users want?*
     - Communicate with stakeholders, clarify and reach consensus
  2. Requirements specification: *What exactly are we doing?*
     - User requirements, system requirements
  3. Requirements validation: *Can we do these things?*
     - Feasibility studies, prototyping

Analysis and Design

- Architectural Design
  - What classes? What methods? How do they interact?
- Security by Design
  - What are the threats? Do we cause any? How do we prevent them?
- User Interface and User Interaction
  - UI mockups, diversity of needs

Testing

- Can be divided by *who* runs the tests:
  - **Development testing**, by developers
    - Unit testing, integration testing, structural (white-box) testing…
  - **Release testing**, by testers before release
    - Functional (black-box) testing, acceptance testing, …
  - **User testing**, by users
    - Alpha testing, beta testing…
- Testing can and should be integrated into development (how?)

Maintenance

- Software should be seen as constantly evolving
  - Dependencies are being updated
  - User requirements and environments are changing
  - Business reality is shifting
- Good practices minimize maintenance costs
- What is maintenance? (If you updated a piece of deprecated software, did you create new software?)
- New bugs, new requirements and new problems will appear
  - Debugging is twice as hard as coding

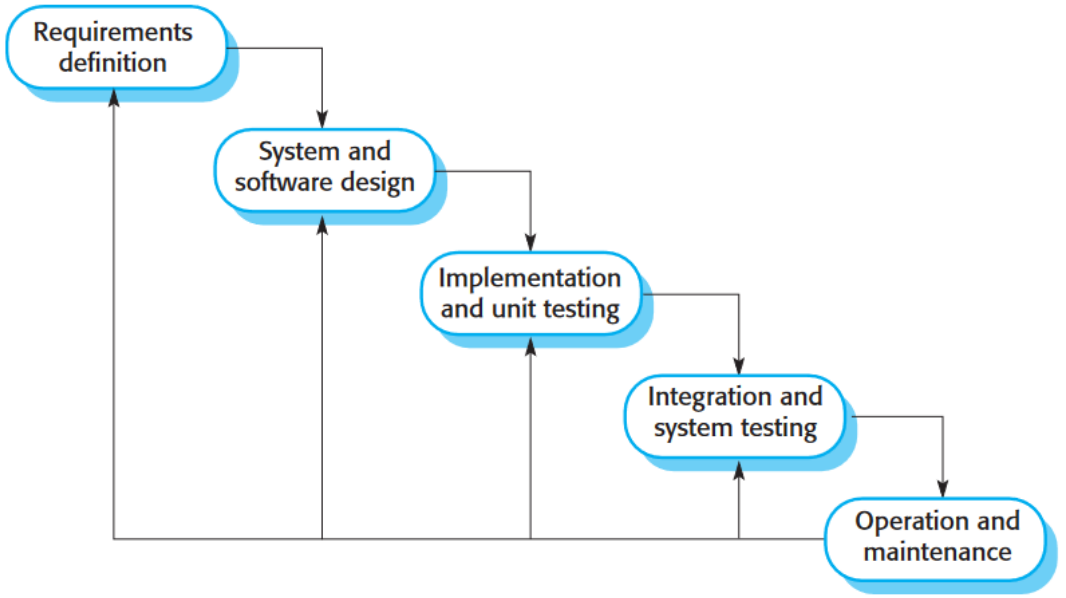
Software Process Models

- How do we *schedule* the software development workflow?

1. Waterfall model
2. Incremental development model
3. Agile

Waterfall Model

- The strawman of models
- Each step must be completed before the next (what are the problems?)



Sommerville

Incremental Development Model

- Produce an *initial version* as early and as soon as possible
  - Start with minimal specifications and aim for release
- Repeat the software development workflow for each version
  - Rethink specifications given development reality/customer feedback
  - Re-design architecture with evolving specifications
  - Refactor and expand codebase
  - Repeated releases to track changes in customer satisfaction

Agile

- What really is Agile?
- Three perspectives
  - Agile in the Textbook
  - Agile as Ceremony
  - Agile as Practice

Agile in the Textbook

- Agile values:
  - Individuals and interactions over processes and tools
  - Working software over comprehensive documentation
  - Customer collaboration over contract negotiation
  - Responding to change over following a plan

Agile in the Textbook

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Embrace change	Expect the system requirements to change, and so design the system to accommodate these changes.
Incremental delivery	The software is developed in increments, with the customer specifying the requirements to be included in each increment.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.
People, not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.

Agile as Ceremony

- Scrum
  - Sprints: a planned 2-3 week block of work
  - Daily stand-ups
  - Sprint Retrospectives
  - Planning for the next Spring
  - Guided by an Agile Coach (product manager)
- Important to measure “Velocity” (how much work can be done)

Agile as Ceremony

Agile in Practice

- Things to consider:
  - How frequent should iteration be for our product? (Is it always 2 weeks?)
  - How rigid is our planning? What happens if devs need more or less time?
  - Where are we getting customers involved?
  - How should devs interact with each other?
  - What is the potential for innovation?
- Good implementations of Agile should…
  - Not be meaningless burden on developers
  - Not be rigid and unresponsive to change
  - Not ignore the needs of people
  - Not be wholly dependent on the quality of the coach

Extreme Programming (XP)

- Idea: minimize design and maximize agile
  - “Ten minutes of design is ideal”
  - YAGNI - “You Ain’t Gonna Need It”
  - Frequently involve customer in process through user stories
  - Test everything: unit test all functions, integrate test at end of day
  - Do not plan for change
- Criticisms:
  - “Big Design Up Front” saves time later
  - Refactoring nightmare
  - Unit/integration tests are limited

Pair Programming

- A core XP practice: two developers code together
  - Share experience and knowledge – addresses critical issue of skill sets becoming too specialized
  - Limits the blame game
  - Constant reviews on each other's code
- Criticisms?

Scaling

- Common criticism of Agile: it does not scale
- “SAFe” (Scaled Agile Framework):
  - Synchronize people, provide transparency to development process
  - Inspect and adapt
  - Avoid work in progress, define deliverables in increments
  - Lean product management