User-Centered Development

- User-centered development refers to a design process for creating a system that meets the needs of the user
- Users should be included in the design process
  - Textbook talks about design in the context of web sites, but here we will apply the same concepts to an arbitrary software system
  - Corporate examples
    - Redesign of IBM site increased sales 400%, use of “help” button decreased 84%
  - Basic idea is to get user input at all stages of the development process, avoid developer’s preference for how to do something
Cognitive Foundations

• Users generally won’t have training for your application, so it must be easy to use
  – Particularly true for web-based applications
• But users probably have prior context of other web-based applications or Office apps
  – Predictability helps ease of use
    • Expect blue text to be a hyperlink
    • Expect “shopping cart” not “item cache”
    • What if your word processor behaved differently every time you used it?

Software Development Process - Metaphors

• Software Farming
  – Growing a system like growing crops?
• Software Oyster Farming
  – System Accretion
    • How an oyster makes a pearl by gradually adding small amounts of calcium carbonate
  – Incremental development
• Building Construction
  – If frame not designed to support the weight…
Development Methodologies

• Classic: “Waterfall Model”

<table>
<thead>
<tr>
<th>Activity</th>
<th>Document</th>
</tr>
</thead>
<tbody>
<tr>
<td>Planning</td>
<td>Statement of Requirements, Functional Specs</td>
</tr>
<tr>
<td>Requirements Analysis</td>
<td>Requirement Specifications</td>
</tr>
<tr>
<td>Design</td>
<td>Design, Architectural Specifications</td>
</tr>
<tr>
<td>Implementation</td>
<td>Implementation Specifications</td>
</tr>
<tr>
<td>Testing</td>
<td>Test Specifications and Reports</td>
</tr>
<tr>
<td>Maintenance</td>
<td>User Manual</td>
</tr>
</tbody>
</table>

Waterfall Model

• Why develop this way?
  – Each activity builds on the previous activity
    • Sequential nature critical before proceeding
  – Formal development process ensures a well-thought out and well-crafted project rather than slinging code willy-nilly
  – Many entry points for formal review and verification
  – Used successfully for many years
Development Methodologies

• Other end of the spectrum: Prototyping

• Prototyping is
  – A controlled experiment with specific questions to answer
  – More than a demo, less than an alpha
  – Functional for limited deployment
    • Iterative process, e.g. making stone soup
    • Good way to start, not always a good way to end
  – Goal-oriented, way to fail early rather than fail later

Prototyping

• Why prototype?
  – Quickly answer critical questions up front
  – Allows feedback in all areas without an existing product
  – A picture is worth 1000 words, a prototype is worth 1000 pictures
Prototyping Steps

1. Start with reasonable concept of end product
2. Ask what the key technologies, risks, issues, and questions are along with expected outcomes
3. Determine best approach to answer the questions
4. Acquire the resources to implement the prototype
5. Implement prototype, using non-critical “filler” if needed for completeness
6. Run experiment or system, examine results, iterate as necessary

We still need specification documents in prototyping!

Sequential or Iterative approach?

- You might choose a sequential approach:
  - Requirements are stable
  - Design is well understood
  - Development team familiar with the application
  - Low risk
  - Long term predictability
  - Cost of changing requirements, design and code downstream is high
Sequential or Iterative?

- You might choose an iterative approach:
  - The requirements are not well understood or you expect them to be unstable
  - The design is complex or unknown
  - The development team is unfamiliar with the application area
  - The project contains a lot of risk
  - Long term predictability is not important
  - The cost of changing requirements, design, and code downstream is low
- eXtreme Programming practice is an iterative approach

Requirements & Documentation

- Benefits of Requirements Documentation:
  - Forces you to think the problem all the way through
    - If you can’t write it down, you probably don’t understand it
    - Defines why you are doing it, what you will be doing, and how it will be done
  - Vehicle for communications
  - Defines commitments and reduces misunderstandings and “feature creep”
  - Bugs found/ fixed in requirements are much less expensive than if found after construction
Myth of Stable Requirements

• For a real project, expect requirements to change
  – Average project experiences 25% change in requirements during development which accounts for 70-85% of rework

Options to handle changing requirements

• Use checklist from SWE to assess the quality of your requirements
• Make sure everyone knows the cost of requirements change
• Set up a change-control procedure
• Use evolutionary or prototyping approaches
• Dump the project
• Examining business case for changes
Software architecture and design

- Architecture is the high-level part of software design; overall framework
- Good architecture makes construction easy
- Typical components
  - Major classes
  - Data design
  - Business rules
  - User interface design
  - Security
  - Performance
  - Scalability
  - Error Processing

How much time?

- A well-run project devotes 10-20% of its effort and 20-30% of its schedule to requirements, architecture, and up-front planning
- If requirements are unstable you’ll probably need to resolve requirements issues yourself
  - Allow time for defining requirements well enough that their volatility will have a minimal impact on construction
  - Treat requirements or architecture work as its own project if unknown; estimate remaining project
User Centered Lifecycle

Variation of the traditional model with more emphasis on the user

1. Define the mission of the system and the user population
   • Clearly decide the goal and the targeted users, establish boundaries for collecting user requirements
2. Collect the user requirements for the system
   • Requirements come from the users, we’ll see later several ways to collect the requirements
3. Create the conceptual design of the system
   • No coding yet! Abstract design, layout, data structures, objects, etc.
4. Create the physical design of the system
   • The long nights of coding
5. Perform usability testing of the system
   • Developer, alpha, beta, other test methodologies
6. Implement and market the system
   • System goes “live”
7. Evaluate and improve the system

Planning – Estimating Effort

• Planning is estimating the effort for each activity
• Example for Waterfall Model:
Plug in Actual Hours

- E.g. if have 150 hours over the entire semester to work on the project:
  - $150 \times 0.05 = 7.5$ hours for presentation
  - $150 \times 0.35 = 52.5$ hours for development
- Etc.
- Can begin to see how realistic a schedule may be and what work is actually involved
- Hard part is coming up with good estimates – programmers are notorious for underestimating!

Project Scheduling Guidelines

- Use previous experience in estimate
- Build in contingency time
- Software engineers are notoriously optimistic
- Use bottom-up scheduling for better accuracy
- Take deadlines and other commitments into account and limit project activities accordingly or make tradeoffs
- Set clear milestones with specific timelines
Gantt Chart

- Can create on paper, in Excel, Project, Web applets
  - Excel probably the easiest
- Graphical view of schedule with dependencies
Project Skills

• Success of a project is not just good project management, but effective documentation, communications, publishing, writing, and presentation skills

• Simplest Method – Project Notebook
  – Can be hardback or electronic
  – Keep references, new ideas, software designs, reports, timelines, etc.
  – Where project management software is useful: these things are easily categorized, dates, etc.

Project Communications

• Regular progress reports
  – Keep a weekly status report of what issues arose, what was accomplished, etc.
  – Makes it much easier to write the final report and maintain a history of the project
  – Useful tool for communications w/supervisor

• Organize project meetings w/supervisor
  – Develop agenda for meetings
    • Report of prior material
    • Current progress
    • Plans
    • Issues
    • Other business
Project Status Reporting

• Report in terms of milestones and deliverables and be specific
  – Yes: “Design approved”, “Interface finalized”
  – No: “Code 80% done” (In terms of what? This the easy 80%?)

• Report problems early
  – Flag problems or potential slips early, don’t try to hide them and hope it will work out later

• Articulate unknowns
  – Margin of error and major unknowns should be declared

• Ready, Fire, Aim
  – Everyone is always behind, don’t try to make up by coding before defining the work, deliverables, etc.

This Class

• In this class we’ll focus on coding and debugging techniques that are not normally covered in other classes

• Practice with technical writing, giving presentations, evaluating code
  – Exercises built around your project so there is not too much “extra” work
Which language?

- Many students often use a new programming language for their project
- Many languages are better suited to particular projects
  - E.g. PHP for Web; better than C++

- Use an appropriate language but it can be a big risk to learn a new programming language during the short span of the semester
  - Usually better to stick with the language you know best, if it applies. If it doesn’t consider changing requirements if possible