**Objectives**
- Discuss issues associated with software development process
  - Organizational
  - Procedural
- Identify best practices to increase your success rate

**Part I: Organizational Issues**
- Before taking any project and writing any code, ask yourself:
  - Is my organization ready to develop software?
  - Some people believe good developers is all you need
    - Reality: Talent is over rated.
    - Discipline is the key to success
    - Joel Spolsky – former Microsoft Excel PM
    - Internet blog with many rule of thumbs and ideas
      - Some are not right IMO

**Joel Test: 12 Steps to better code**
- **Test 1:** Do you use source control?
  - SVN, CVS
  - Manage code and integrate with the rest
  - Keep backups for free ...
- **Test 2:** Can you make a build in one step?
  - Start you application top down
    - Phase 1 of DB Project
  - No mystery to compile, deploy and run application
  - Most IDE create a project that runs!
- CMSC 435 @ UMD – Software Engineering course
  - Deliverable: software application with one click installer

**Joel Test: 12 Steps to better code**
- **Test 3:** Do you make daily builds?
  - Make sure you new code
  - Work and does not breaks someone else code
  - ICOM 5016 last day integration syndrome
  - Do it when people are around to fix it
  - Rotate who is responsible for the build
  - But if someone breaks it that person should fix it

**Joel Test: 12 Steps to better code**
- **Test 4:** Do you have a bug database?
  - Track know bugs
    - Pick the ones to fix now and the ones to be left for future
    - Track cause, buggy behavior, expected behavior, owner

**Joel Test: 12 Steps to better code**
- **Test 5:** Do you fix bugs before writing new code?
  - Critical bugs must be fixed ASAP
    - Ex. NULL pointers, number overflows, etc.
  - You know what are doing and is easier to track what happened
    - In one week you will forget what the code was doing ...
  - Lots of unfixed bugs == unreliable schedule to finish
  - ICOM Software Gurus 😊
    - Write 5000 lines of undebugged and untested code
    - Expect to be able to fix them a week before deadline
    - Often they get bored and quit the project (go to play games)
<table>
<thead>
<tr>
<th>Joel Test: 12 Steps to better code</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Test 6:</strong> Do you have an up-to-date schedule?</td>
</tr>
<tr>
<td>- Schedule is not carved in stone</td>
</tr>
<tr>
<td>- Each developer must update time to end task</td>
</tr>
<tr>
<td>- Make sure debugging and testing is included</td>
</tr>
<tr>
<td>- Do not let manager change time</td>
</tr>
<tr>
<td>- Project will fail</td>
</tr>
<tr>
<td>- Cut luxury features in order to meet deadline</td>
</tr>
<tr>
<td><strong>Test 7:</strong> Do you have a spec?</td>
</tr>
<tr>
<td>- Functional specification – what the software will do?</td>
</tr>
<tr>
<td>- Not UML, not layer diagram</td>
</tr>
<tr>
<td>- Test and possible GUI sketch</td>
</tr>
<tr>
<td>- What will happen when people use the code</td>
</tr>
<tr>
<td>- No spec == guessing</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Joel Test: 12 Steps to better code</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Test 7:</strong> Do you have a spec?</td>
</tr>
<tr>
<td>- Spec helps you &quot;debug application&quot;</td>
</tr>
<tr>
<td>- What is needed and what is not needed</td>
</tr>
<tr>
<td>- Right vs. wrong behavior</td>
</tr>
<tr>
<td>- Spec helps you control schedule</td>
</tr>
<tr>
<td>- Identify required vs. nice to have (luxury) features</td>
</tr>
<tr>
<td><strong>Test 8:</strong> Do programmers have quiet working conditions?</td>
</tr>
<tr>
<td>- People like to concentrate and write code (inspiration)</td>
</tr>
<tr>
<td>- Distractions</td>
</tr>
<tr>
<td>- Phone</td>
</tr>
<tr>
<td>- Constant questions about schedule or windows crash</td>
</tr>
<tr>
<td>- Far away bath rooms / food / coffee</td>
</tr>
<tr>
<td>- Co-worker interruptions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Joel Test: 12 Steps to better code</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Test 8:</strong> Do programmers have quiet working conditions?</td>
</tr>
<tr>
<td>- One minute interruption ≈ 15 minutes of lost work</td>
</tr>
<tr>
<td>- Give people their own desk with their machine</td>
</tr>
<tr>
<td><strong>Test 9:</strong> Do you use the best tools money can buy?</td>
</tr>
<tr>
<td>- Do not torture your developers with</td>
</tr>
<tr>
<td>- Old machines with small monitors</td>
</tr>
<tr>
<td>- Disk space quotas</td>
</tr>
<tr>
<td>- Outdated OS release</td>
</tr>
<tr>
<td>- Bad software tools</td>
</tr>
<tr>
<td>- Microsoft Paint vs. Photoshop for Web imaging</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Joel Test: 12 Steps to better code</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Test 10:</strong> Do you have testers?</td>
</tr>
<tr>
<td>- UML bug free mythology</td>
</tr>
<tr>
<td>- Reality: Every software coding effort is full of bugs</td>
</tr>
<tr>
<td>- Bad design or bad implementation</td>
</tr>
<tr>
<td>- Programmer does first test</td>
</tr>
<tr>
<td>- Unit</td>
</tr>
<tr>
<td>- Dedicated tester check whole system or subsystem</td>
</tr>
<tr>
<td>- Unbiased</td>
</tr>
<tr>
<td>- Tests several scenarios and documents anomalies</td>
</tr>
<tr>
<td>- Testing and coding should be interleaved</td>
</tr>
<tr>
<td>- Write code, debug, test, write code, debug, test, ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Joel Test: 12 Steps to better code</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Test 11:</strong> Do new candidates write code during their interview?</td>
</tr>
<tr>
<td>- No writing code == uncertain skills == uncertain project member == uncertain project outcome</td>
</tr>
<tr>
<td>- Resume is paper – you can put whatever you want</td>
</tr>
<tr>
<td>- Need to make candidates write code</td>
</tr>
<tr>
<td>- Remove duplicates from a linked list</td>
</tr>
<tr>
<td>- Sort data on an array</td>
</tr>
<tr>
<td>- ICOM 4.0 GPA Students</td>
</tr>
<tr>
<td>- Some of them cannot write code</td>
</tr>
<tr>
<td>- They even evade ICOM 5015</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Joel Test: 12 Steps to better code</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Test 12:</strong> Do you do hallway usability testing?</td>
</tr>
<tr>
<td>- If your co-workers have a hard time with your GUI the user has no chance</td>
</tr>
<tr>
<td>- Show people you UI and collect data on</td>
</tr>
<tr>
<td>- Intuitiveness of UI</td>
</tr>
<tr>
<td>- Problems with locations of buttons, menus, etc.</td>
</tr>
<tr>
<td>- Issues with ease to find desired information</td>
</tr>
<tr>
<td>- You can go to a more complex usability testing later on</td>
</tr>
<tr>
<td>- If you cannot convince your coworker you are in trouble</td>
</tr>
<tr>
<td>- Redesigning the UI can be quite expensive</td>
</tr>
</tbody>
</table>
Software Products classification

- Products can be classified as
  - Shrink wrap
  - Customized
  - Throwaway

Shrink wrap
- Targeted to a general audience
- Ex. MS Office, Photoshop, iTunes
- Customized
  - Specific to a given user or industry
  - Ex. CESCO David, UPR PATSI, Universal Insurance Claims Management
- Throwaway
  - Internal code used to experiment with a given technology
  - Ex. Phase 1 and Phase 2 of ICOM 5016 Project

Throwaway
- Internal code used to experiment with a given technology
- Ex. Phase 1 and Phase 2 of ICOM 5016 Project

Software Products classification

- Used by a large number of people
- Little control on how it is used
- Sell at retail stored or over the Web
- Develop and release it to the public
  - Bug fixed must be provided over Web
  - Scales well in terms of money
    - License issued to individual users
    - Should be able to recover cost with first N licenses
    - After that is all profit
  - Need to test and maintain aggressively
    - To continue selling it and making profit
    - Create loyal customer base

Customized software

- Also called internal software
- Used by people at a company or community
- Smaller audience
- More control on how it used
  - You can actually dictate requirements for usage
- Develop and deploy to the company/community
  - Need to give them training
  - Often system is buggy and you need to keep fixing it
- Less scale in term of profit
  - Contract-based: Once contract is over you get no money
  - Contracts then to be expensive (to account for profits vs losses)
  - Contract expires and no more maintenance is given
    - Unless a maintenance contract gets setup

Part II: Procedural Issues

- Software development is cyclic!
  - Old school water fall software development process assures failure
  - You need to have constant testing and feedback from the user
  - UML will not produce code for you!
  - How do I specify a multi-threaded system with a shared queue that controls access to a pool of disks?
  - UML is good to talk with others about your code
    - Like ER diagrams
  - Source code == real software specification

Making money on software

- Shrink wrap
  - Make a product that many people will use
    - Ex. Office, Photoshop, MS.Net, iWeb, MacOS
  - Companies: Microsoft, Apple, IBM, Adobe, Skype
  - Customized
    - Make a product that a big agency will use
    - Companies: Rock Solid, EDS, IBM, HP
  - You should try to make shrinkwrap whenever possible
    - Only do customize to help you get cash to make another product
    - Shrink wrap is where you want to be
Cowboy Coding Model

- You start writing code without an actual plan
- Hacker’s way of doing things
  - I will start writing code and I will figure out things along the way
  - Many ICOM Software Gurus work like this
- You guarantee that the project will be
  - Late
  - Full of hard to understand code
  - Full of incompatibilities
  - Full of unusable features
  - Featuring a hard to use UI

Waterfall Model

- Software is built in steps
  - One phase leads to the next
  - If this phase is right the next will likely be right 😊

Waterfall Model: Problems

- In each phase you deal with a bunch of uncertainties
  - Customer changes her mind about UI
  - You drop the ball with the design
  - Mixed data model with storage logic
  - Use multi-threaded when multi-process was better
  - You realize your platform has buggy support for networking
  - Ex. PDF!
- Change is assured when building software
  - You need a way to make mid-flight course corrections

Reality in Software Development

- At each step you might need to revisit decisions from previous phase

Rapid Application Development (RAD)

- Build incomplete but functional prototype (like a demo!)
- Debug and test major components
- Involve customer by showing prototype
  - Nail down UI
  - Prevent change of accepted features ...
  - Add features/fixes into prototype until you reach release status
  - Hey, but finish the product!!!
- Examples:
  - Agile Programming
  - Extreme Programming
  - SCRUM

Agile Programming

- Family of techniques based upon
  - Inclusion of customer into design/development
  - Short cycle to produce working code (not all features)
    - Every few weeks a new version with a set of new features is delivered
  - Test-Driven software development
    - First make the tests, then you write code that can pass them
  - Refactor code
    - Change code based on results of debugging, testing, and user feedback
  - Produce stable release as results of continuous improvement process
Extreme Programming

- Based on daily practices and team values
- Customer and business people are part of the team
- Always deliver a new working version ASAP
- Communicate effectively with all team members

XP Values

- Simplicity
  - Write code that is simple, clean and straightforward
- Communication
  - Keep direct communication between customers, developers, business people and managers
- Feedback
  - Always comment on out other code, features, and issues
    - E.g., code reviews
- Courage
  - Write the code! If you mess up just refactor
  - Avoid getting stuck in perfect implementation issues

XP Activities

- Simple Design
  - Start with a simple system that works
  - Add new working features
- Pair Programming
  - 2 programmers work side by side on the same machine (like Spartan kings)
  - Faster, better code plus you have redundancy
- Test-Driven Development
  - Unit test and full system tests as new features are added
- Design Improvement
  - Refactoring – fix the design as you write code
  - You only know you are wrong when you see it

SCRUM

- XP can be chaotic
- Scrum is controlled chaos
- The Team:
  - Scrum master
    - PM
  - Product Owner
  - Customer and business people
  - Developers
- Team works in sprints or burst of one month
  - Design, code, test and demo software
  - Next sprint adds features to previous release
  - Backlog of the spring list the features to do in each sprint

SCRUM Process

- Start out by giving high level system organization
  - Boxes and arrows

Software System Architecture
Layered Software Design

- Break down software model into layer
- Each layer is one or more libraries with specific role

Design Patterns

- Well understood and documented recipes to build software
  - Reusable code
  - Idea borrowed from architecture
  - Archetypes
  - Columns, arcs, etc.
- Smalltalk had them for GUI
- Gang of Four Book (GoF) popularized design patterns for CS
- You should build your libraries around them

Example: Abstract Factory

- You need to write an email client
- Must run in
  - Windows XP and Vista
  - MacOS X
  - Ubuntu
- Each one has a different look and feel
- You do not want to write the different programs
- Instead you want to share as much code as possible
  - Only differentiate in how UI elements are created

Questions?