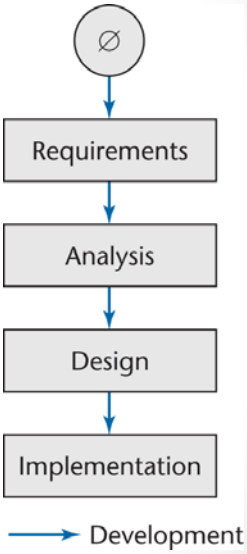
Software Engineering

CSC440/640
Prof. Schweitzer
Week 2

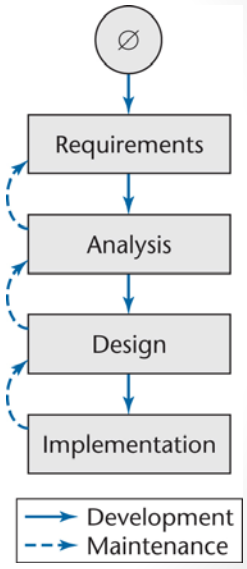
Software Development in Theory

- Ideally, software is developed as described in Chapter 1
  - Linear
  - Starting from scratch
- In the real world, software development is totally different
  - We make mistakes
  - The client's requirements change while the software product is being developed



Waterfall Model

- The linear life cycle model with feedback loops
  - The waterfall model cannot show the order of events



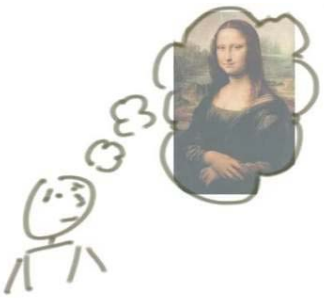
Lessons of the Winburg Mini Case Study

- In the real world, software development is more chaotic than the Winburg mini case study
- Changes are always needed
  - A software product is a model of the real world, which is continually changing
  - Software professionals are human, and therefore make mistakes

Moving Target Problem

- A change in the requirements while the software product is being developed
- Even if the reasons for the change are good, the software product can be adversely impacted
  - Dependencies will be induced
- Any change made to a software product can potentially cause a *regression fault*
  - A fault in an apparently unrelated part of the software
- If there are too many changes
  - The entire product may have to be redesigned and reimplemented
- Change is inevitable
  - Growing companies are always going to change
  - If the individual calling for changes has sufficient clout, nothing can be done about it
- There is no solution to the moving target problem

Incremental Development



- Builds a bit at a time
- Incrementing often calls for a fully formed idea
- Changing while incrementing slows us down

1 

2 

3 

Iterative Development

- Builds a rough version, then slowly adds detail and additional quality
- Iterating allows you to move from a vague idea to realization
- While iterating we expect change

1 

2 

3 

Miller's Law

- At any one time, we can concentrate on only approximately seven chunks (units of information)
- To handle larger amounts of information, use stepwise refinement
  - Concentrate on the aspects that are currently the most important
  - Postpone aspects that are currently less critical
  - Every aspect is eventually handled, but in order of current importance
- This is an incremental process

Classical Phases vs. Workflows

- Sequential phases do not exist in the real world
- Instead, the five core workflows (activities) are performed over the entire life cycle
  - Requirements workflow
  - Analysis workflow
  - Design workflow
  - Implementation workflow
  - Test workflow

Workflows

- All five core workflows are performed over the entire life cycle
- However, at most times one workflow predominates
- Examples:
  - At the beginning of the life cycle
    - The requirements workflow predominates
  - At the end of the life cycle
    - The implementation and test workflows predominate
- Planning and documentation activities are performed throughout the life cycle

Strengths of the Iterative-and-Incremental Model

- There are multiple opportunities for checking that the software product is correct
  - Every iteration incorporates the test workflow
  - Faults can be detected and corrected early
- The robustness of the architecture can be determined early in the life cycle
  - Architecture — the various component modules and how they fit together
  - Robustness — the property of being able to handle extensions and changes without falling apart

Strengths of the Iterative-and-Incremental Model

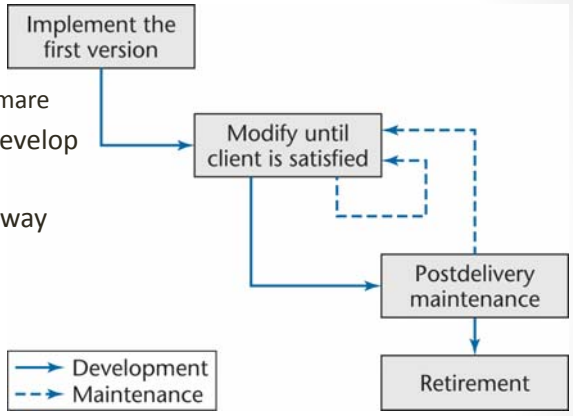
- We can mitigate (resolve) risks early
  - Risks are invariably involved in software development and maintenance
- We have a working version of the software product from the start
  - The client and users can experiment with this version to determine what changes are needed
- Variation: Deliver partial versions to smooth the introduction of the new product in the client organization

Other Life-Cycle Models

- Featured Here:
  - Code-and-fix life-cycle model
  - Rapid prototyping life-cycle model
  - Open-source life-cycle model
  - Agile processes
- Not Featured
  - Synchronize-and-stabilize life-cycle model (Microsoft)
  - Spiral life-cycle model
  - Kan Ban

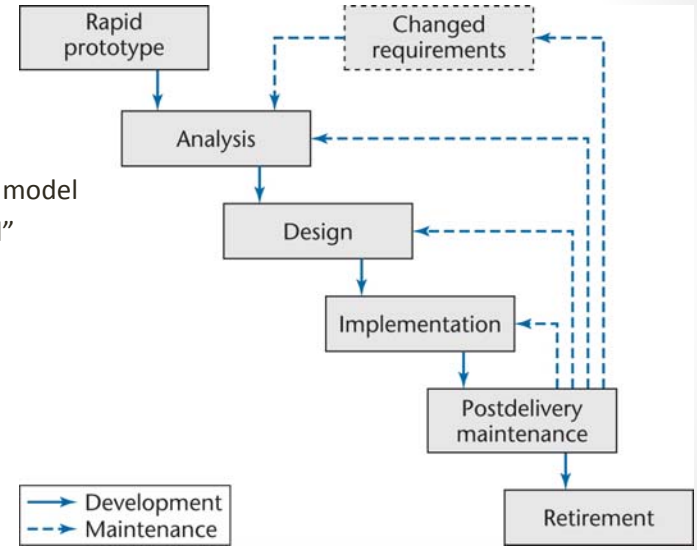
Code-and-Fix Model

- No design
- No specifications
  - Maintenance nightmare
- The easiest way to develop software
- The most expensive way



Rapid Prototyping Model

- Linear model
- "Rapid"



Open-Source Life-Cycle Model

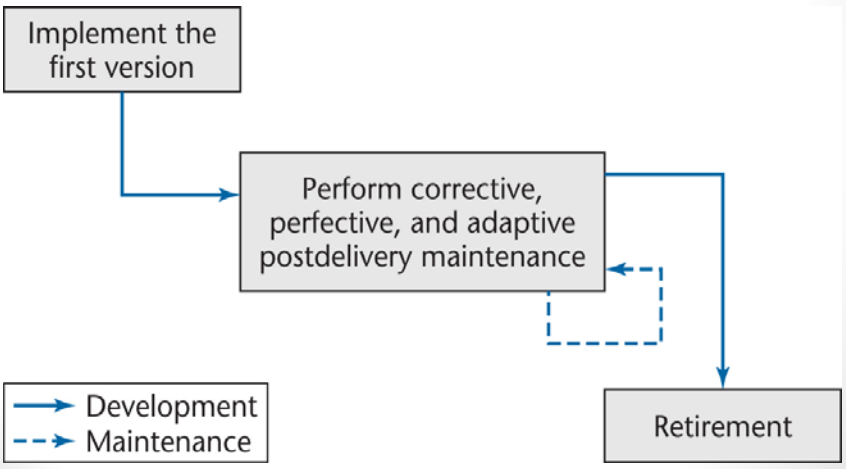
- Two informal phases
- First, one individual builds an initial version
  - Made available via the Internet (e.g., SourceForge.net)
- Then, if there is sufficient interest in the project
  - The initial version is widely downloaded
  - Users become co-developers
  - The product is extended
- Key point: Individuals generally work voluntarily on an open-source project in their spare time

The Activities of the Second Informal Phase

- Reporting and correcting defects
  - Corrective maintenance
- Adding additional functionality
  - Perfective maintenance
- Porting the program to a new environment
  - Adaptive maintenance
- The second informal phase consists solely of post-delivery maintenance
  - The word “co-developers” on the previous slide should rather be “co-maintainers”

Open-Source Life-Cycle Model

- Post-delivery maintenance life-cycle model



Open-Source Life-Cycle Model

- Closed-source software is maintained and tested by employees
  - Users can submit failure reports but never fault reports (the source code is not available)
- Open-source software is generally maintained by unpaid volunteers
  - Users are strongly encouraged to submit defect reports, both failure reports and fault reports
- Core group
  - Small number of dedicated maintainers with the inclination, the time, and the necessary skills to submit fault reports ("fixes")
  - They take responsibility for managing the project
  - They have the authority to install fixes
- Peripheral group
  - Users who choose to submit defect reports from time to time

Open-Source Life-Cycle Model

- An initial working version is produced when using
  - The rapid-prototyping model;
  - The code-and-fix model; and
  - The open-source life-cycle model
- Then:
  - Rapid-prototyping model
    - The initial version is discarded
  - Code-and-fix model and open-source life-cycle model
    - The initial version becomes the target product
- Consequently, in an open-source project, there are generally no specifications and no design
- How have some open-source projects been so successful without specifications or designs?

Open-Source Life-Cycle Model

- The open-source life-cycle model is restricted in its applicability
- It can be extremely successful for infrastructure projects, such as
  - Operating systems (Linux, OpenBSD, Mach, Darwin)
  - Web browsers (Firefox, Netscape)
  - Compilers (gcc)
  - Web servers (Apache)
  - Database management systems (MySQL)

Agile Processes

- Somewhat controversial new approach
- Stories (features client wants)
  - Estimate duration and cost of each story
  - Select stories for next build
  - Each build is divided into tasks
  - Test cases for a task are drawn up first
- Pair programming
- Continuous integration of tasks
- Different Styles/Variations
  - XP
  - Scrum
  - Kanban

Unusual Features of XP

- Colocation of team members
- A client representative is always present
- Software professionals cannot work overtime for 2 successive weeks
- No specialization
- Refactoring (design modification)

Acronyms of Extreme Programming

- YAGNI (you aren't gonna need it)
- DTSTTCPW (do the simplest thing that could possibly work)
- A principle of XP is to minimize the number of features
  - There is no need to build a product that does any more than what the client actually needs

Agile Processes

- XP is one of a number of new paradigms collectively referred to as agile processes
- Seventeen software developers (later dubbed the "Agile Alliance") met at a Utah ski resort for two days in February 2001 and produced the Manifesto for Agile Software Development
- The Agile Alliance did not prescribe a specific life-cycle model
  - Instead, they laid out a group of underlying principles
- Agile processes are a collection of new paradigms characterized by
  - Less emphasis on analysis and design
  - Earlier implementation (working software is considered more important than documentation)
  - Responsiveness to change
  - Close collaboration with the client

The Agile Manifesto

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

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

That is, while there is value in the items on the right, we value the items on the left more.

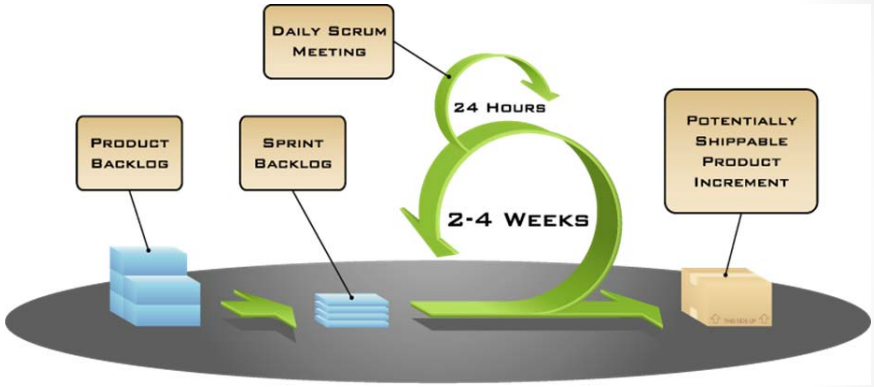
Agile Processes

- It gives the client confidence to know that a new version with additional functionality will arrive every 3 weeks
- The developers know that they will have 3 weeks (but no more) to deliver a new iteration
  - Without client interference of any kind
- If it is impossible to complete the entire task in the time box, the work may be reduced ("descoped")
  - Agile processes demand fixed time, not fixed features
- Another common feature of agile processes is *stand-up meetings*
  - Short meetings held at a regular time each day
  - Attendance is required
- Participants stand in a circle
  - They do not sit around a table
  - To ensure the meeting lasts no more than 15 minutes

Agile Processes (contd)

- At a stand-up meeting, each team member in turn answers five questions:
  - What have I done since yesterday's meeting?
  - What am I working on today?
  - What problems are preventing me from achieving this?
  - What have we forgotten?
  - What did I learn that I would like to share with the team?
- The aim of a stand-up meeting is
  - To raise problems
  - Not solve them
- Solutions are found at follow-up meetings, preferably held directly after the stand-up meeting

Scrum Release Cycle



The Unified Process

- Until recently, three of the most successful object-oriented methodologies were
  - Booch's method
  - Jacobson's Objectory
  - Rumbaugh's OMT
- In 1999, Booch, Jacobson, and Rumbaugh published a complete object-oriented analysis and design methodology that unified their three separate methodologies
  - Original name: *Rational Unified Process* (RUP)
  - Next name: *Unified Software Development Process* (USDP)
  - Name used today: *Unified Process* (for brevity)

The Unified Process

- The Unified Process is not a series of steps for constructing a software product
  - No such single “one size fits all” methodology could exist
  - There is a wide variety of different types of software
- The Unified Process is an adaptable methodology
  - It has to be modified for the specific software product to be developed
- UML is graphical
  - A picture is worth a thousand words
- UML diagrams enable software engineers to communicate quickly and accurately

Iteration and Incrementation within the Object-Oriented Paradigm

- The Unified Process is a modeling technique
  - A model is a set of UML diagrams that represent various aspects of the software product we want to develop
- UML stands for unified modeling language
  - UML is the tool that we use to represent (model) the target software product
- The object-oriented paradigm is iterative and incremental in nature
  - There is no alternative to repeated iteration and incrementation until the UML diagrams are satisfactory

Overview of the Requirements Workflow

- First, gain an understanding of the application domain (or domain, for short)
  - That is, the specific business environment in which the software product is to operate
- Second, build a business model
  - Use UML to describe the client's business processes
  - If at any time the client does not feel that the cost is justified, development terminates immediately

Overview of the Requirements Workflow

- It is vital to determine the client's constraints
  - Deadline
    - Nowadays, software products are often mission critical
  - Parallel running
  - Portability
  - Reliability
  - Rapid response time
  - Cost
    - The client will rarely inform the developer how much money is available
    - A bidding procedure is used instead
- The aim of this *concept exploration* is to determine
  - What the client needs
  - *Not* what the client wants

The Analysis Workflow

- The aim of the analysis workflow
  - To analyze and refine the requirements
- Why not do this during the requirements workflow?
  - The requirements artifacts must be totally comprehensible by the client
- The artifacts of the requirements workflow must therefore be expressed in a natural (human) language
  - All natural languages are imprecise

The Analysis Workflow

- Example from a manufacturing information system:
  - "A part record and a plant record are read from the database. If it contains the letter A directly followed by the letter Q, then calculate the cost of transporting that part to that plant"
- To what does it refer?
  - The part record?
  - The plant record?
  - Or the database?
- Two separate workflows are needed
  - The requirements artifacts must be expressed in the language of the client
  - The analysis artifacts must be precise, and complete enough for the designers

The Specification Document

- Specification document ("specifications")
  - It constitutes a contract
  - It must not have imprecise phrases like "optimal," or "98% complete"
- Having complete and correct specifications is essential for
  - Testing and
  - Maintenance
- The specification document must not have
  - Contradictions
  - Omissions
  - Incompleteness
- RFC 2119 - http://tools.ietf.org/html/rfc2119

Software Project Management Plan

- Once the client has signed off the specifications, detailed planning and estimating begins
- We draw up the software project management plan, including
  - Cost estimate
  - Duration estimate
  - Deliverables
  - Milestones
  - Budget
- This is the earliest possible time for the SPMP

The Design Workflow

- The aim of the design workflow is to refine the analysis workflow until the material is in a form that can be implemented by the programmers
  - Many nonfunctional requirements need to be finalized at this time, including
    - Choice of programming language
    - Reuse issues
    - Portability issues

Classical Design

- Architectural design
  - Decompose the product into modules
- Detailed design
  - Design each module:
    - Data structures
    - Algorithms

Object-Oriented Design

- Classes are extracted during the object-oriented analysis workflow and
  - Designed during the design workflow
- Accordingly
  - Classical architectural design corresponds to part of the object-oriented analysis workflow
  - Classical detailed design corresponds to part of the object-oriented design workflow

Implementation Artifacts

- Each component is tested as soon as it has been implemented
  - *Unit testing*
- At the end of each iteration, the completed components are combined and tested
  - *Integration testing*
- When the product appears to be complete, it is tested as a whole
  - *Product testing*
- Once the completed product has been installed on the client's computer, the client tests it
  - *Acceptance testing*

Implementation Artifacts

- COTS software (Packaged Software) is released for testing by prospective clients
  - Alpha release
  - Beta release
  - Release Candidate?
- There are advantages and disadvantages to being an alpha or beta release site

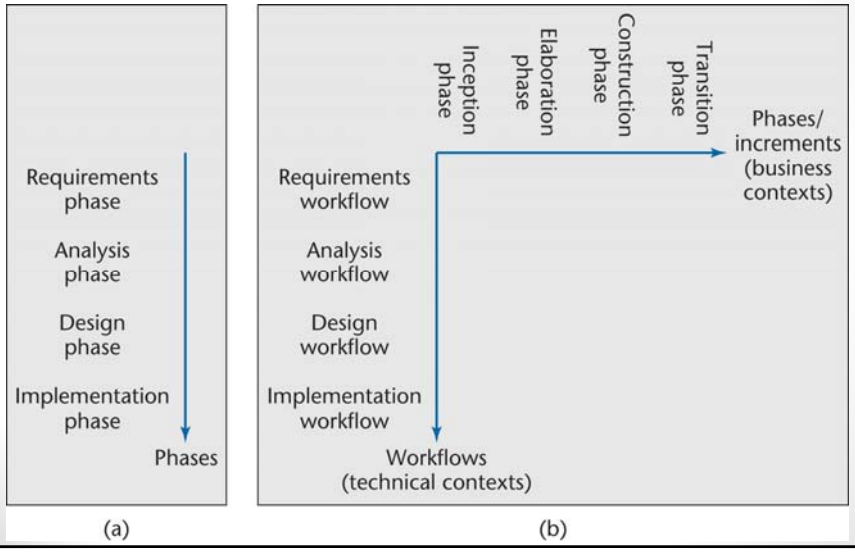
Postdelivery Maintenance

- Postdelivery maintenance is an essential component of software development
  - More money is spent on postdelivery maintenance than on all other activities combined
- Problems can be caused by
  - Lack of documentation of all kinds
- Two types of testing are needed
  - Testing the changes made during postdelivery maintenance
  - Regression testing
- All previous test cases (and their expected outcomes) need to be retained

The Phases of the Unified Process

- The four increments are labeled
  - Inception phase
  - Elaboration phase
  - Construction phase
  - Transition phase
- The phases of the Unified Process are the increments

One- and Two-Dimensional Life-Cycle Models



The Inception Phase

- The aim of the inception phase is to determine whether the proposed software product is economically viable
- Goals:
  - Gain an understanding of the domain
  - Build the business model
  - Delimit the scope of the proposed project
    - Focus on the subset of the business model that is covered by the proposed software product
  - Begin to make the initial business case

The Initial Business Case

- Questions that need to be answered include:
  - Is the proposed software product cost effective?
  - How long will it take to obtain a return on investment?
  - Alternatively, what will be the cost if the company decides not to develop the proposed software product?
  - If the software product is to be sold in the marketplace, have the necessary marketing studies been performed?
  - Can the proposed software product be delivered in time?
  - If the software product is to be developed to support the client organization's own activities, what will be the impact if the proposed software product is delivered late?

The Initial Business Case

- What are the risks involved in developing the software product
- How can these risks be mitigated?
  - Does the team who will develop the proposed software product have the necessary experience?
  - Is new hardware needed for this software product?
  - If so, is there a risk that it will not be delivered in time?
  - If so, is there a way to mitigate that risk, perhaps by ordering back-up hardware from another supplier?
  - Are software tools (Chapter 5) needed?
  - Are they currently available?
  - Do they have all the necessary functionality?

The Inception Phase: Risks

- There are three major risk categories:
  - Technical risks
    - See earlier slide
  - The risk of not getting the requirements right
    - Mitigated by performing the requirements workflow correctly
  - The risk of not getting the architecture right
    - The architecture may not be sufficiently robust
- To mitigate all three classes of risks
  - The risks need to be ranked so that the critical risks are mitigated first

The Inception Phase: Workflows

- A small amount of the analysis workflow may be performed during the inception phase
  - Information needed for the design of the architecture is extracted
- Accordingly, a small amount of the design workflow may be performed, too
- Coding is generally not performed during the inception phase
  - However, a *proof-of-concept prototype* is sometimes build to test the feasibility of constructing part of the software product
- The test workflow commences almost at the start of the inception phase
  - The aim is to ensure that the requirements have been accurately determined

Inception Phase: Documentation

- The deliverables of the inception phase include:
  - The initial version of the domain model
  - The initial version of the business model
  - The initial version of the requirements artifacts
  - A preliminary version of the analysis artifacts
  - A preliminary version of the architecture
  - The initial list of risks
  - The initial ordering of the use cases (Chapter 10)
  - The plan for the elaboration phase
  - The initial version of the business case

Elaboration Phase

- The aim of the elaboration phase is to refine the initial requirements
  - Refine the architecture
  - Monitor the risks and refine their priorities
  - Refine the business case
  - Produce the project management plan
- The major activities of the elaboration phase are refinements or elaborations of the previous phase

Tasks of the Elaboration Phase

- The tasks of the elaboration phase correspond to:
  - All but completing the requirements workflow
  - Performing virtually the entire analysis workflow
  - Starting the design of the architecture

Elaboration Phase: Documentation

- The deliverables of the elaboration phase include:
  - The completed domain model
  - The completed business model
  - The completed requirements artifacts
  - The completed analysis artifacts
  - An updated version of the architecture
  - An updated list of risks
  - The project management plan (for the rest of the project)
  - The completed business case

Construction Phase

- The aim of the construction phase is to produce the first operational-quality version of the software product
  - This is sometimes called the beta release
- The emphasis in this phase is on
  - Implementation and
  - Testing
    - Unit testing of modules
    - Integration testing of subsystems
    - Product testing of the overall system

Construction Phase: Documentation

- The deliverables of the construction phase include:
  - The initial user manual and other manuals, as appropriate
  - All the artifacts (beta release versions)
  - The completed architecture
  - The updated risk list
  - The project management plan (for the remainder of the project)
  - If necessary, the updated business case

The Transition Phase

- The aim of the transition phase is to ensure that the client's requirements have indeed been met
  - Faults in the software product are corrected
  - All the manuals are completed
  - Attempts are made to discover any previously unidentified risks
- This phase is driven by feedback from the site(s) at which the beta release has been installed
- The deliverables of the transition phase include:
  - All the artifacts (final versions)
  - The completed manuals

Capability Maturity Models

- Not life-cycle models
- Rather, a set of strategies for improving the software process
  - SW–CMM for software
  - P–CMM for human resources ("people")
  - SE–CMM for systems engineering
  - IPD–CMM for integrated product development
  - SA–CMM for software acquisition
- These strategies are unified into CMMI (capability maturity model integration)

SW-CMM

- 5 Levels of Maturity
  1. **Initial** – Chaotic/Ad Hoc/Heroics. Process is new or *undocumented* and *unrepeatable*
  2. **Repeatable** – Process is *documented* and attempts are made to repeat across projects
  3. **Defined** – Process is defined and *accepted* within the business and *decomposed* into work instructions
  4. **Managed** – Process is managed and there are *metrics* for performance
  5. **Optimizing** – Management includes a process to *improve* the software process
- Similar in concept to ISO 9000 Series of Standards

Next Week

- Reading for Next Week
  - Chapters 4 & 7
  - Quiz Will Come from These Two Chapters
  - Skipping Chapters 5 & 6 (for now)

Software Project

- Problem Statement In Appendix A of the Book
- We'll Divide into Teams Tonight
- Read Through the Problem Statement for Next Week
- Deliverable 1 – Next Week
  - Pick a "Project Manager" – Ultimately this person will be responsible for official deliverables… i.e. the stuff that person turns into me is what counts for your team.
  - Team Communication Plan – How will you communicate with each other? How will you assign tasks? Will you have team meetings at regular intervals?
    - Excel/Word
    - Trello
    - Google Docs
    - Something else?

Software Project Deliverables

- Required Deliverables at End of Project
  - Communication Plan
  - Requirements Document
  - Design Document
  - Test Plan
  - Final Code