Software Process

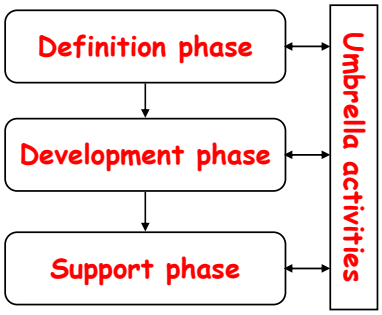
Software Process

- For a single project
  - planning (time, resources, assignments)
  - tracking and measuring
- Across multiple projects
  - organizational planning (time, resources, etc.)
  - hiring, training, tool acquisition, etc.
  - process assessment and improvement
- For software engineering in general
  - helps organize the field
  - it is a topic of its own right (process models)

Elements of the Process

- **Tasks** *analyze requirements*
- **Work products** *requirements specification*
- **Relationships** *req. specs feed design*
- **Milestones** *reviewed requirements*
- **People** *Alice and Bob*
- **Methods** *use OO analysis with UML*
- **Tools** *Rational Rose*

Generic View of Software Process



Definition Phase

- Tasks related to problem definition
  - What? - requirements, constraints, external environment, validation criteria, etc.
- Step 1: system engineering
  - Hardware, software, people, databases, operational procedures, etc.
- Step 2: analysis of the problem
  - requirements analysis and domain analysis
- Step 3: project planning
  - Resources (e.g. people), cost, schedule

Development Phase

- Tasks related to problem solution
  - How? - architecture, components, data, algorithms, programming, testing, etc.
- Step 1: software design (the blueprint)
  - Design models that describe structure, interactions, etc.
- Step 2: code generation
- Step 3: software testing
  - Goal: uncover as many errors as possible

Support Phase

- Tasks related to software change
  - Definition and development in the context of existing software
- Correction of defects
  - e.g. Y2K problem - many billions of dollars
- Adaptation to changes in the environment
  - New hardware, OS, business rules, etc.
- Enhancements (new features, etc.)
- Reengineering: for easier future changes

Real-World Example: Cobol

- Business programming language
  - Initial spec: 1960, last spec: 2002
  - Millions lines of code in legacy applications
  - Dozens of books and training courses
    - "Cobol for the 21 century", 10th ed

```
GetBookRankings.
  MOVE W-BookNum TO PrevBookNum
  MOVE ZEROS TO BookSalesTotal
  PERFORM UNTIL W-BookNum NOT EQUAL TO PrevBookNum
              OR EndOfWorkfile
    ADD W-Copies TO BookSalesTotal
    RETURN WorkFile
      AT END SET EndOfWorkfile TO TRUE
      END-RETURN
  END-PERFORM
```


Some Umbrella Activities

- Project management
  - Tracking and control of people, process, work products, schedule, cost, quality, risk, etc.
- Quality assurance: activities that ensure high quality of all work products
  - Formal technical reviews of requirement specifications, designs, source code
  - Software testing
- Configuration management
  - Controls the changes in work products: e.g., version control for source code (RCS,CVS)

Process Model

- A general pattern for a software process
  - Instantiated for each specific project
  - May have to be modified for the particular circumstances
    - Not a dogma – adjust as necessary
- The model defines tasks, work products, relationships, milestones, etc.
  - e.g. "the output of task X is input for task Y"

Observations

- Process models are idealizations
  - The real world is a very complex place
- They can be difficult to execute
- They can be viewed as interfering with "real work"
- Conformance can be faked
- Still, they are essential to avoid chaos

Code-and-Fix Process

- The first thing people tried in the 1950s
  - Beginning programmers often use it

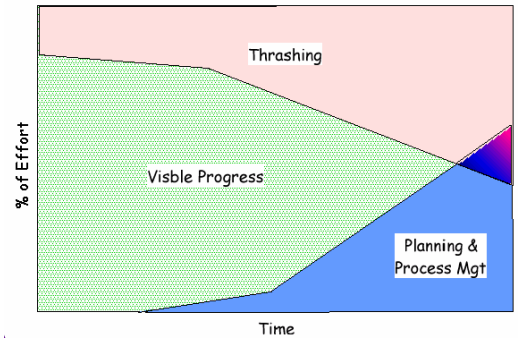
1. Write program
2. Improve it (debug, add functionality, improve efficiency, …)
3. GOTO 1

- Works for small 1-person projects and for some CS course assignments

Problems with Code-and-Fix

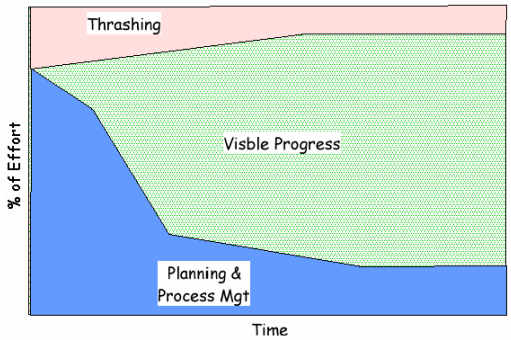
- Poor match with user needs
- Bad overall structure
  - No blueprint
- Poor reliability - no systematic testing
- Maintainability? What's that?
- What happens when the programmer quits?

Code-and-Fix



from McConnell, *After the Goldrush*

More Advanced Process



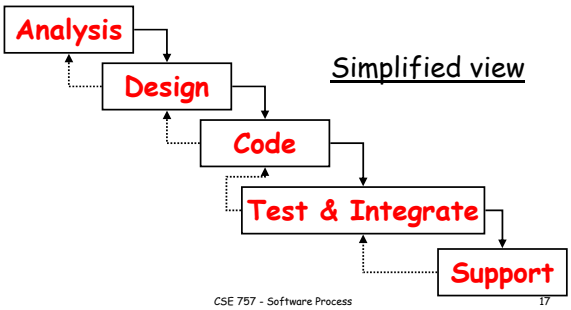
from McConnell, *After the Goldrush*

Examples of Process Models

- Waterfall model
- Prototyping model
- Iterative & incremental model
- Dozens of others
  - Spiral model
  - Rapid Application Development (RAD) model
  - Formal methods model
  - ...

Waterfall Model

- The "classic" process model since 1970s
  - Also called "software life cycle"



Waterfall Phases

- Analysis: requirements (features, etc.) and relevant domain concepts
- Design: solution blueprint for the systems and for individual components
- Individual components: implementation and testing (unit testing)
- System integration
  - with integration/system testing
- Software support and maintenance

Key Points of the Model

- The project goes through the phases sequentially
- Possible feedback and iteration across phases
  - e.g. during coding, a design problem is identified
- Typically, few/no iterations are used
  - e.g., after a certain point of time, the design is "frozen"

Waterfall Assumptions

- Requirements are known from the start, before design
- Requirements are stable
- The design can be done abstractly and speculatively
  - i.e. it is possible to correctly guess in advance how to make it work
- Everything will fit together when we start the integration

Pros and Cons

- Inflexible: the actual process is not so sequential
  - Limited use of iteration, problems from earlier phases are hard to fix
- Expects full requirements early
- Working program is not available early
  - High risk issues are not tackled early enough
- Pros: widely used, systematic, good for projects with well-defined requirements
  - Makes managers happy

Prototyping Model

- Problem: customers have general objectives but no detailed requirements
- Solution: build a prototype
  - Analysis of known requirements, plus quick-and-dirty design and implementation
- Iterations: customer evaluation followed by prototype refinements
- Goal: understand the requirements
  - The prototype is thrown away

Pros and Cons

- Better understanding of requirements
  - Good starting point for other process models (e.g. waterfall)
- Problem: the prototype may be used as a starting point rather than thrown away
  - Bad idea: prototypes typically have poor design and quality
- Bad decisions during prototyping may propagate to the real product

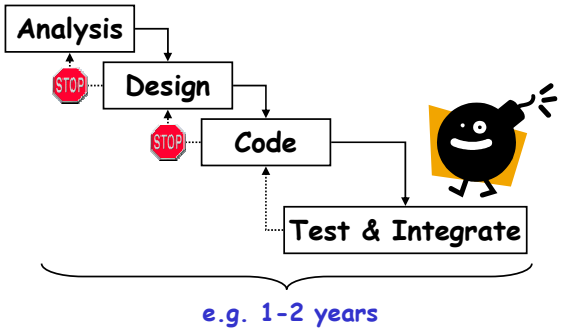
Iterative & Incremental Model

- Waterfall: single release
- Iterative: many releases (increments)
  - First increment: core functionality
  - Successive increments: add/fix functionality
  - Final increment: the complete product
- Each iteration: a short mini-project with a separate lifecycle
  - e.g., waterfall
- Increments may be built sequentially or in parallel

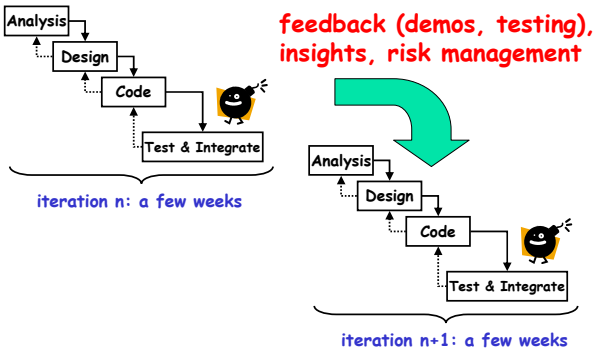
Iterative & Incremental Model

- Outcome of each iteration: tested, integrated, executable system
- Iterations length is short and fixed
  - e.g. 2 weeks, 4 weeks, 6 weeks
- Takes many iterations (e.g. 10-15)
- Does not try to "freeze" the requirements and design speculatively
  - Rapid feedback, early insight, opportunity to modify requirements and design
  - Later iterations: reqs/design become stable

Waterfall (Sequential) Model



Iterative Model



Iterative & Incremental Model

- Risks are addressed in early iterations
  - e.g. server application with 10,000 concurrent users and <1s response: quickly build and evaluate components/architecture
- Continuous feedback from users
  - To build what the users want

- Early quality control: testing and reviews

Iterative vs. Waterfall

- Waterfall: tries to fully specify and freeze requirements and design
  - Problem 1: the requirements may change in the future
  - Problem 2: early design decisions are speculative
- Iterative: accepts requirements change as inevitable, and allows the design to evolve

Iterative vs. Waterfall

- Waterfall: inadequate risk management
  - Testing in the end, when problems are hard and expensive to fix
- Iterative: identifies high-risk issues through rapid feedback
  - At the end of each iteration, learn from users, developers, testers
    - Users try the partial system and say "Well, the feature I wanted is actually ..."
    - Usability testing: e.g. the user interface
    - Load testing

Pros and Cons

- Early discovery of high risks
  - Technical, requirements, usability, etc.
- Early visible progress
  - Customers and managers are happy
- Managed complexity: avoids long and complex analysis/design steps
  - No "analysis paralysis"
  - Smaller increments are easier to manage

Pros and Cons

- Operational product is available early
  - Allows early feedback from users
    - Result: a system the users actually want
  - The market may force a limited version
- Accommodates changes
  - e.g. experience with earlier increments may be used to define/refine the requirements
- Constant changes ("feature creep") may erode system architecture

Final Notes

- Currently, iterative & incremental models are very popular: best practices
  - Solve many problems associated with the waterfall model
- More about one such model: Unified Process, defined by the same folks that created UML
  - Popular example of an iterative model

Overview

- The Unified Process (UP): an example of an iterative and incremental process
  - Very popular in the last few years
  - Incorporates modern principles for software development
- Will provide a context for our discussion of analysis and design
- There are many other process models
  - The UP is a good sample process

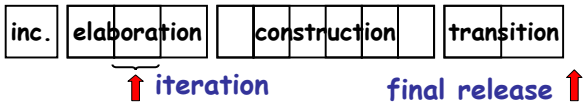
Objectives

- Define the organization of the UP
- Discuss key practices of the UP

A Little History

- "The three amigos": Grady Booch, Ivar Jacobson, James Rumbaugh
  - Separate methodologies for object-oriented analysis and design (OOAD) in the early 90s
  - Created the Unified Modeling Language (UML) in 1996
- 1999: defined the Unified Process (UP) in Rational Software Inc. Rational. software IBM.
  - Refinement: Rational Unified Process (RUP)
  - Rational makes lots of money from the RUP

Organization of the UP



- Inception: preliminary investigation
- Elaboration: analysis, design, some coding
- Construction: more coding and testing
- Transition: testing and deployment

Inception

- Short initial phase (e.g. about a week)
  - Investigation of purpose, scope, and feasibility: should we even bother?
- Order-of-magnitude unreliable estimates of cost and time
- Definition of some requirements (10%)
- Plan for the first iteration of elaboration
- Possibly a proof-of-concept prototype

Elaboration

- Analysis
  - Most of requirements analysis
  - Most of domain analysis (domain modeling)
- Most of the design
- Some coding
  - Iterative implementation of the core architecture and high-risk requirements
- Testing of all implemented code
- More precise estimates of time/cost

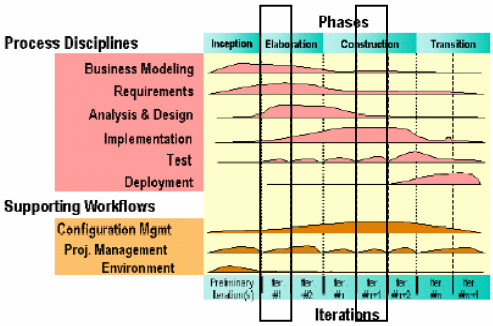
Remaining Phases

- **Construction phase**
  - Iterative implementation and testing of the remaining lower risk and easier aspects
  - Very little analysis or design
  - Preparation for deployment
- **Transition**
  - Beta testing
  - Deployment

UP Disciplines

- Discipline: a set of activities and related artifacts in one subject area
- Artifact - any kind of work product: e.g. code, diagrams, models, documents, ...
- We will consider three disciplines
  - Business modeling – e.g. domain analysis
  - Requirements – requirements analysis
  - Design

Effort Distribution



source: Rational. software *white paper, May 2002*

Iteration Length

- Iterations should be short
  - Typically 2-6 weeks
  - Goal: small steps, rapid feedback, adaptation
  - Exception: massive teams with lots of communication – but no more than 3-6 mo.
- Iterations should be timeboxed (fixed in length)
  - The system should be integrated, tested, and stabilized by the schedule date
  - If not possible: move tasks to the next iter.

Reasons for Timeboxing

- Work expands to fill the available time
  - With a deadline two weeks away, people focus and get moving
- Encourages prioritization and decisiveness
- Team satisfaction and confidence
  - Quick and repeating sense of completion, competency, and closure
  - Also, increased confidence for customers and managers

High-risk and High-value Issues

- In the early iterations, focus on:
  - Overall architecture
  - Components for risky requirements
  - Components with high value to the customer
- Better to fail early than late
  - The process is risk-driven

Other UP Practices

- Continuously engage the customer
- Early focus on the core architecture
  - Shallow implementation: overall structure, component interfaces and responsibilities, without "in-depth" implementation
- Verify quality often and early
  - Testing and reviews: critical for early feedback and to avoid expensive late defects
  - Unlike the waterfall model

Some Work Products

Artifact	Incep	Elab	Const	Trans
Use-Case Model	X	X		
Supplem. Spec	X	X		
Domain Model		X		
Design Model		X	X	
Implem. Model (code)		X	X	X

Requirements analysis: **Use-Case Model + Supplementary Specification**
Domain analysis: **Domain Model**
Design: **Design Model**
Coding: **Implementation Model**

Relationships Among Models