The Software Process

- **Chapters 3 & 4**

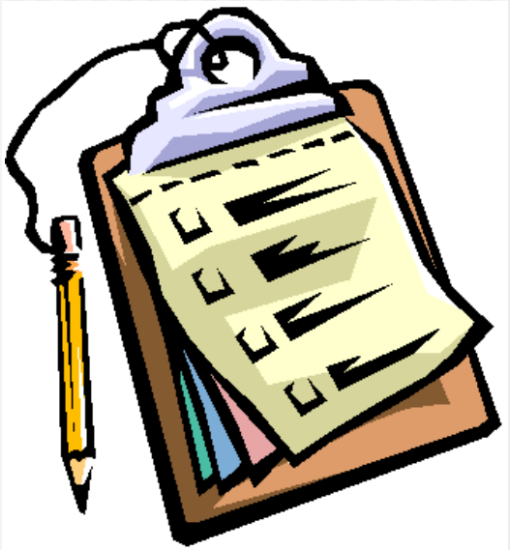
Slide Set to accompany

Software Engineering: A Practitioner's Approach, 8/e

by Roger S. Pressman and Bruce R. Maxim

Agenda

- Software Process Structure
- Process Models



Software Process Structure

Software Process

- "A process defines ***who*** is doing ***what***, ***when*** and ***how to*** reach a certain goal."

  Ivar Jacobson, Grady Booch, and James Rumbaugh

- Process defines the approach that is taken as software is engineered
- Framework for **activities**, **actions**, and **tasks**
  - required to build high-quality software

Activity, Action and Task

- Activity strives to achieve a broad objective
  - Applied **regardless of** the application domain, size of the project, complexity of the effort,...
  - e.g., modeling, communication, …
- Action encompasses a set of tasks that produce a major work product
  - e.g., architectural design (major product: an architectural design model).
- Task focuses on a small, but well-defined objective that produces a tangible outcome
  - e.g., conducting a unit test

A Generic Process Framework

Process framework

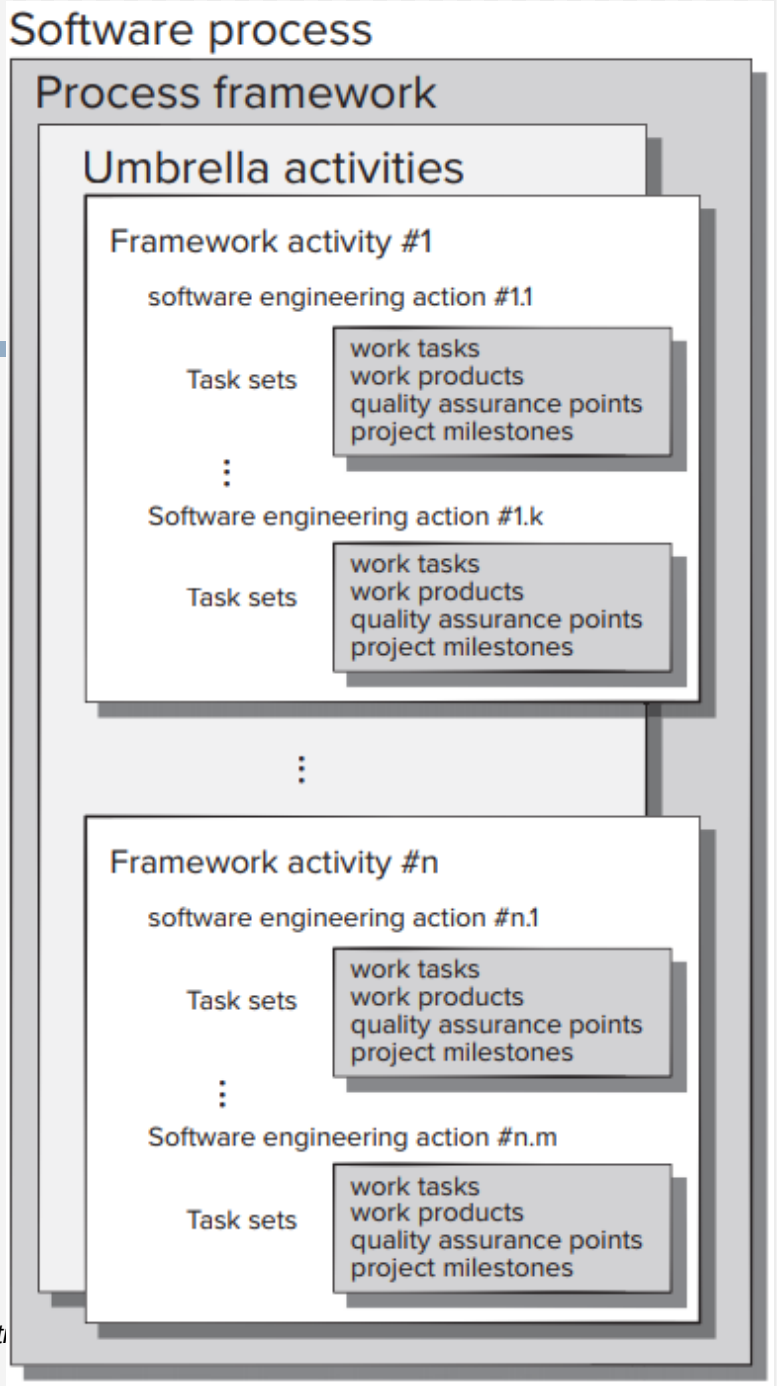
Framework activities

- work tasks
- work products
- milestones & deliverables
- QA checkpoints

Umbrella Activities

Framework Activities VS Umbrella Activities

- Framework activities are applicable to all software projects, regardless of their size or complexity
- Umbrella activities are applicable across the entire software process
  - Complement framework activities
  - Applied throughout a software project



Framework Activities

- **Communication**
  - Collaborate with the customer and stakeholders
  - Intent: to understand objectives and to gather requirements
- **Planning**
  - Software project plan (risks, resources, products, schedules,..)
- **Modeling**
  - Analysis of requirements
  - Design
- **Construction**
  - Code generation and Testing
- **Deployment**
  - The software is delivered to the customer
  - The customer evaluates the product and provides feedback

Notes on Framework Activities

- These five **generic** framework activities can be used for the development of different systems (from small and simple to large and complex)
  - The details of the process is quite different in each case, but the framework activities remain the same
- For many software projects, framework activities are applied iteratively
  - Applied repeatedly through a number of iterations
  - Each iteration produces a software increment
  - An increment provides a subset of overall features
  - Software is gradually completed

Typical Umbrella Activities

- Software project tracking and control
- Risk management
- Software quality assurance
- Technical reviews
- Measurement
  - e.g., lines of code, code complexity, customer satisfaction score, cycle time, cost variance, schedule variance,…
- Software configuration management
- Reusability management
- Work product preparation and production

Process Adaptation

- Software process is **not a rigid prescription**
- It should be **adaptable**
  - to the problem,
  - to the project,
  - to the team.
- Conclusion:
  - A process adopted for one project might be significantly different than a process adopted for another project

Adapting a Process Model

- The overall flow of activities, actions, and tasks and the interdependencies among them
- The degree to which actions and tasks are defined within each framework activity
- The degree to which work products are identified and required
- The manner which quality assurance activities are applied
- The manner in which project tracking and control activities are applied
- The degree to which the customer and other stakeholders are involved with the project
- The level of autonomy given to the software team
- The degree to which team organization and roles are prescribed

Example: Adapting a Task Set

- *Action:* Requirements gathering

(an important action during the communication activity)

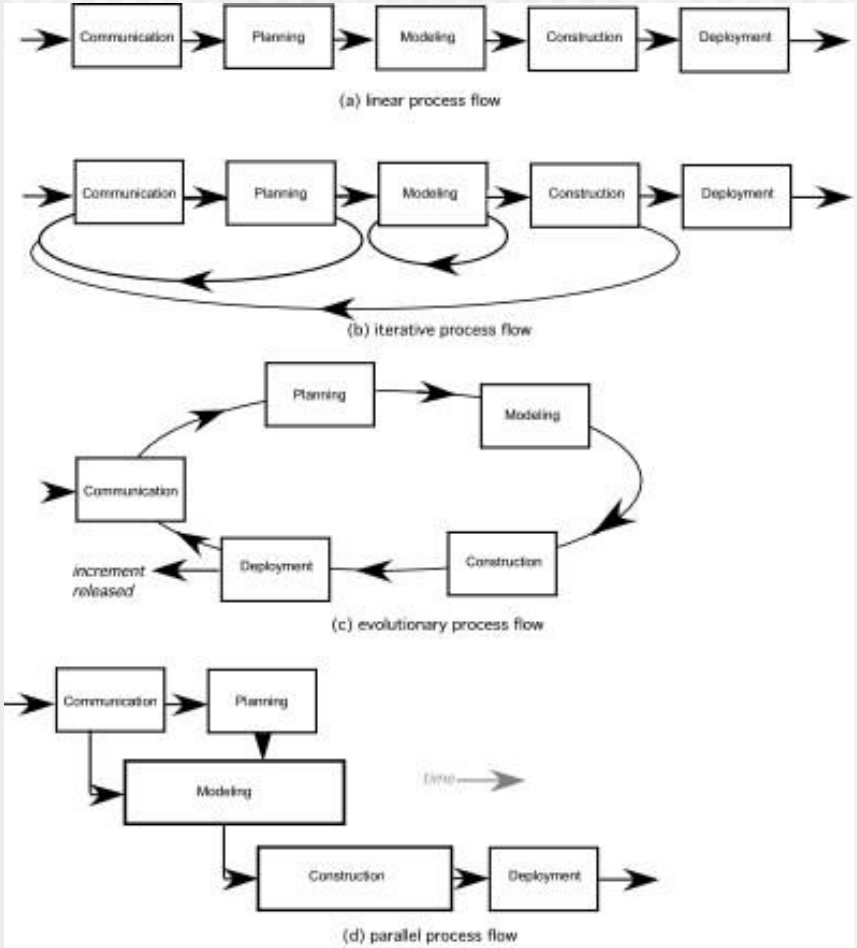
- **Task set for a small and simple project**

1. Make a list of stakeholders for the project.
2. Invite all stakeholders to an informal meeting.
3. Ask each stakeholder to make a list of features and functions required.
4. Discuss requirements and build a final list.
5. Prioritize requirements.
6. Note areas of uncertainty.

- **Task set for a large and complex project**

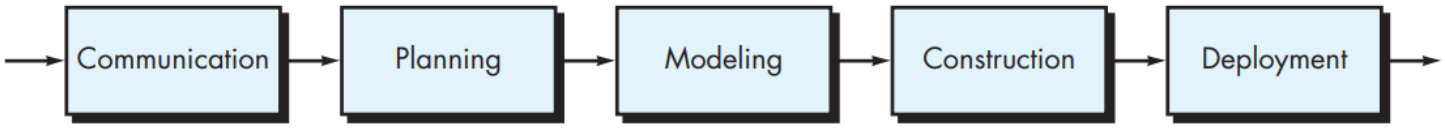
1. Make a list of stakeholders for the project
2. Interview each stakeholder separately to determine overall needs
3. Build a preliminary list of functions and features based on stakeholder input
4. Schedule a series of facilitated application specification meetings
5. Conduct meetings
6. Produce informal user scenarios as part of each meeting
7. Refine user scenarios based on stakeholder feedback
8. Build a revised list of stakeholder requirements
9. Prioritize requirements
10. ….

Process Flow (also called *work flow*)



Linear Flow

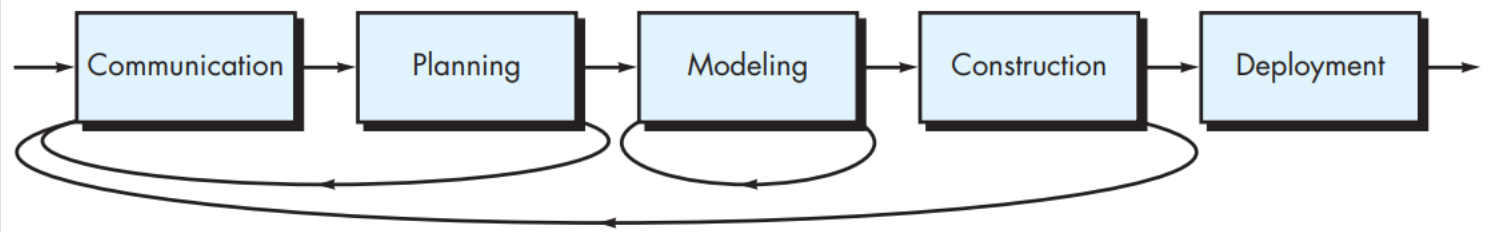
- **Linear process** flow executes each of the five framework activities in sequence, beginning with communication and culminating with deployment



(a) Linear process flow

Iterative Flow

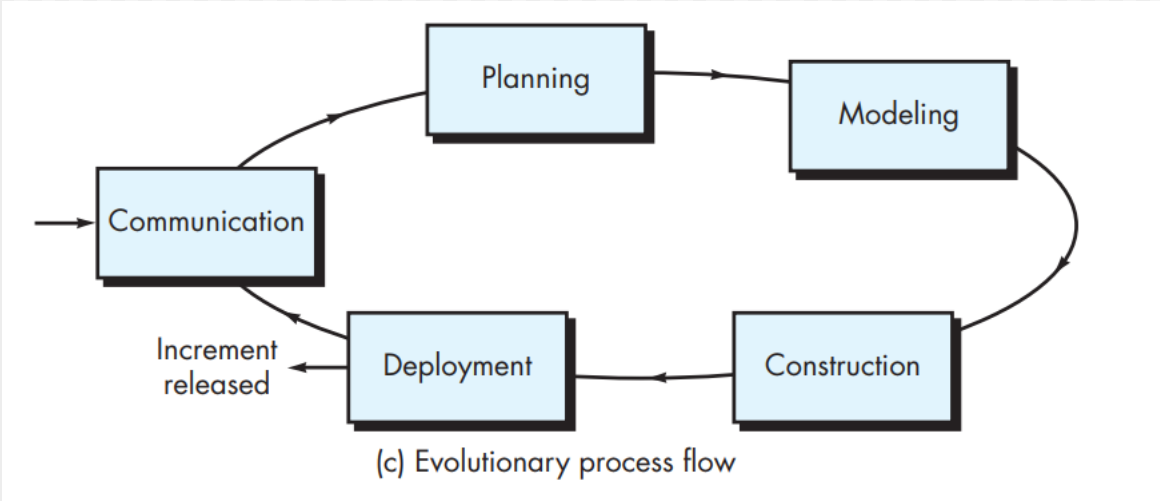
- **Iterative process** flow repeats one or more of the activities before proceeding to the next



(b) Iterative process flow

Evolutionary Flow

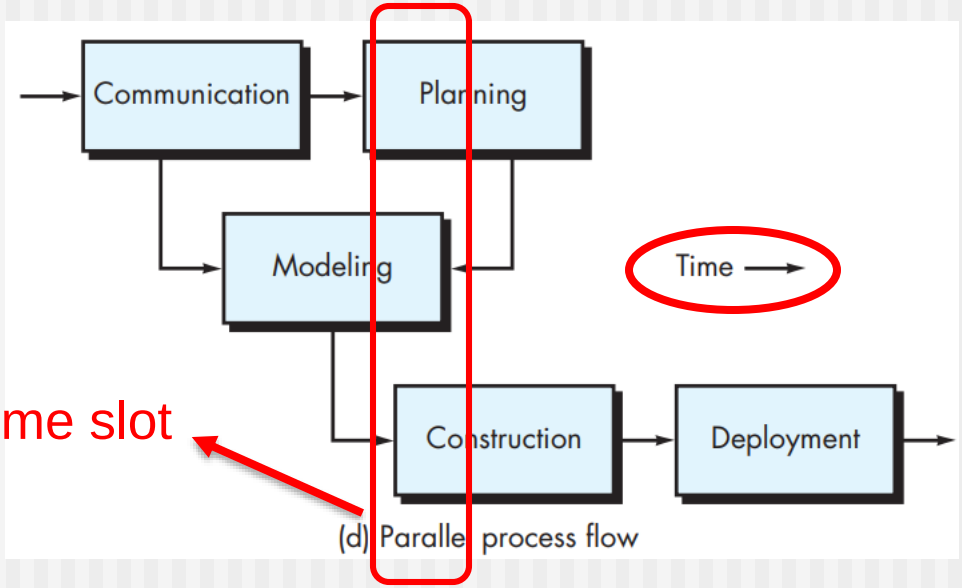
- **Evolutionary process** flow iterates the activities in a “circular” manner
- Each circuit through the five activities leads to a more complete version of the software



(c) Evolutionary process flow

Parallel Flow

- **Parallel process** flow executes one or more activities in parallel with other activities
- For example, modeling for one aspect of the software might be executed in parallel with construction of another aspect of the software



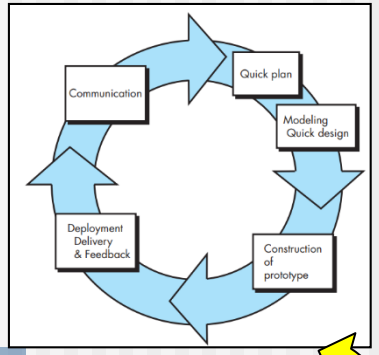
Process Patterns

- A *process pattern*
  - Describes a **repeatable process-related problem**,
  - Identifies the **environment** in which the problem has been encountered, and
  - Suggests one or more **proven solutions** to the problem.
- Each pattern must include:
  - Pattern name, intent, type, initial context, problem, solution, and resulting context
- Example: Pattern name=RequirementsUnclear
  - Problem: Requirements are hazy or nonexistent
    - stakeholders are unsure of what they want
  - Solution: A description of the prototyping process

Process Pattern Types

- *Stage patterns*—defines a problem associated with a framework **activity** for the process.
- *Task patterns*—defines a problem associated with a software engineering **action** or work task
- *Phase patterns*—define the **sequence of framework activities** that occur with the process

A Sample Process Pattern



INFO

An Example Process Pattern

The following abbreviated process pattern describes an approach that may be applicable when stakeholders have a general idea of what must be done but are unsure of specific software requirements.

Pattern Name. **RequirementsUnclear**

Intent. This pattern describes an approach for building a model (a prototype) that can be assessed iteratively by stakeholders in an effort to identify or solidify software requirements.

Type. Phase pattern.

Initial Context. The following conditions must be met prior to the initiation of this pattern: (1) stakeholders have been identified; (2) a mode of communication between stakeholders and the software team has been established; (3) the overriding software problem to be solved has been identified by stakeholders; (4) an initial understanding of project scope, basic business requirements, and project constraints has been developed.

Problem. Requirements are hazy or nonexistent, yet there is clear recognition that there is a problem to be solved, and the problem must be addressed with a software solution. Stakeholders are unsure of what they want; that is, they cannot describe software requirements in any detail.

Solution. A description of the prototyping process would be presented here and is described later in Section 4.1.3.

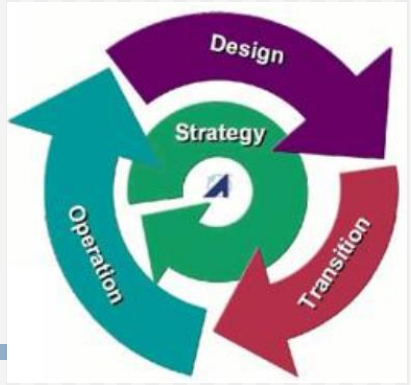
Resulting Context. A software prototype that identifies basic requirements (e.g., modes of interaction, computational features, processing functions) is approved by stakeholders. Following this, (1) the prototype may evolve through a series of increments to become the production software or (2) the prototype may be discarded and the production software built using some other process pattern.

Related Patterns. The following patterns are related to this pattern: **CustomerCommunication, IterativeDesign, IterativeDevelopment, CustomerAssessment, RequirementExtraction.**

Known Uses and Examples. Prototyping is recommended when requirements are uncertain.

Process Models

Process Model

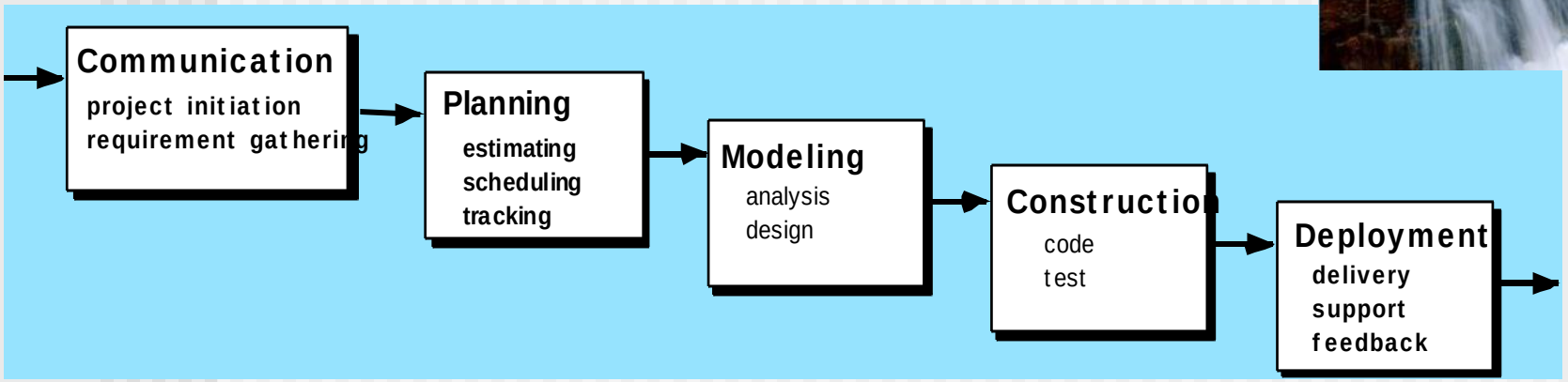


- A specific roadmap for software engineering
  - It defines the flow of all activities, actions and tasks,
  - the degree of iteration,
  - the work products,
  - and the organization of the work that must be done
- Different process models:
  - Traditional models, Evolutionary models, Agile models, …
- Different models are suitable for different projects
- Software process model ≈ Software process ≈ software development **methodology** ≈ software development life cycle ≈ software development process

Prescriptive Models

- Prescriptive process models advocate an orderly approach to software engineering
  - They strive for structure and order in software development
  - Sometimes referred to as “traditional” process models
  - Examples: Waterfall, Spiral,…
- Prescriptive process models define a prescribed set of **process elements** and a predictable **process flow**
  - Process elements: framework activities, actions, tasks, work products, quality assurance, and change control mechanisms
  - They prescribe a process flow (also called work flow)

The Waterfall Model



- The waterfalls model, sometimes called the **classic life cycle**
- A systematic, sequential approach to software development
- The oldest paradigm for software engineering
- A linear process model: progress is flowing steadily downwards (like a waterfall)

Problems with Waterfall

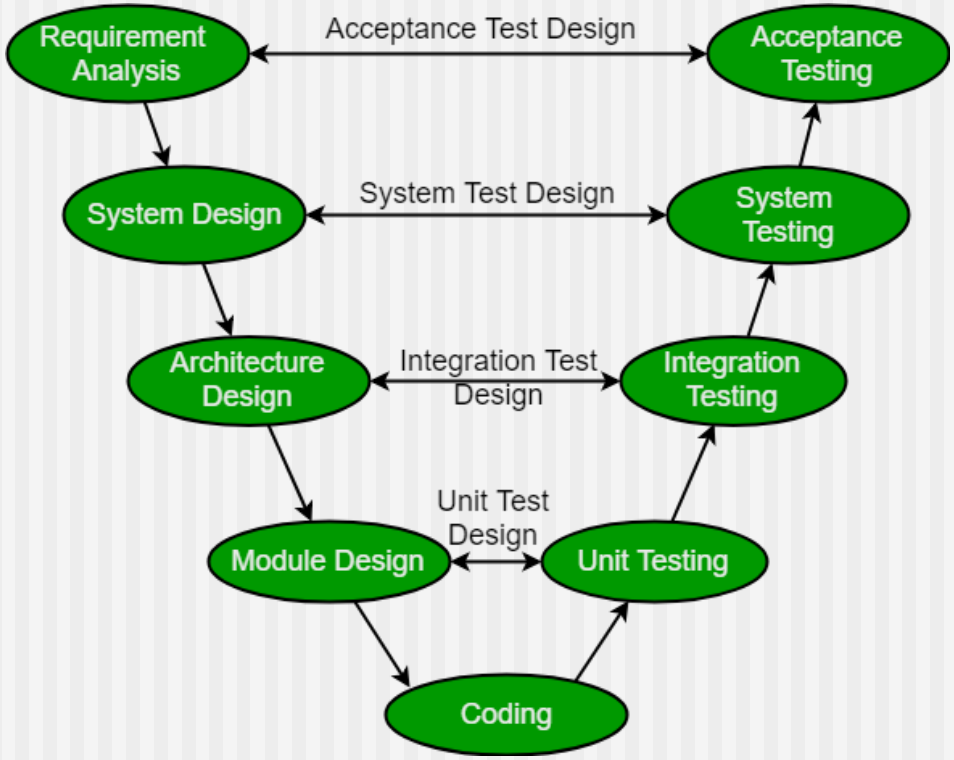
1. Real projects rarely follow the sequential flow that the model proposes
   - After-the-fact changes are prohibitively costly (if not impossible)
2. Difficult for the customer to state all requirements explicitly
3. The customer must have patience
   - A working version will not available late
   - A major blunder, if undetected until the working program is reviewed, can be disastrous
4. “Blocking states” problem

The Applicability of Waterfall

- When the requirements for a problem are well understood and reasonably stable
- This situation is often encountered when well-defined adaptations or enhancements to an existing system must be made
  - E.g., an adaptation to an accounting software because of changes to government regulations

The V-Model

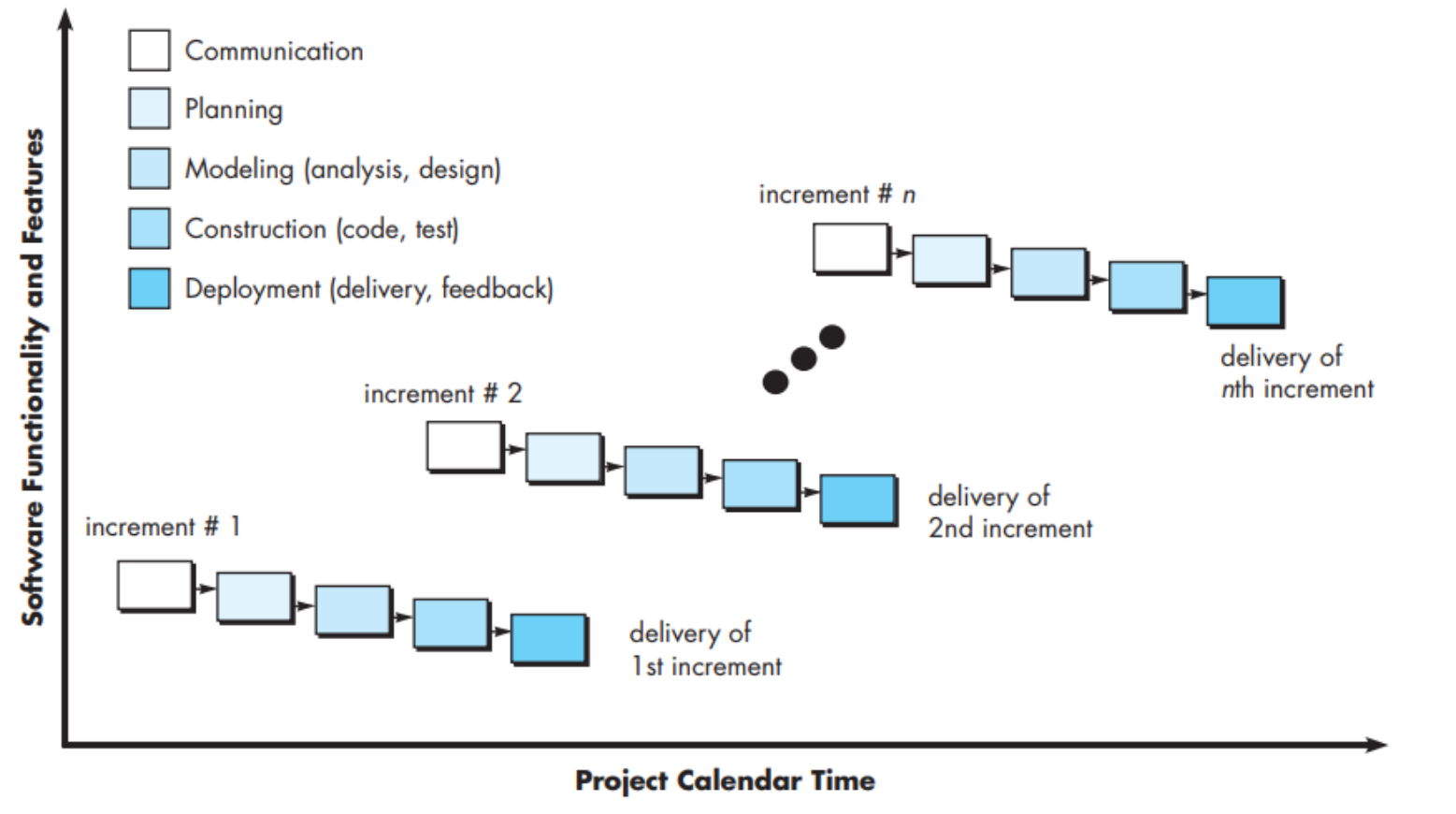
- A variation of the waterfall model
  - A variation in the **representation**
  - No fundamental difference
  - V-model is also linear

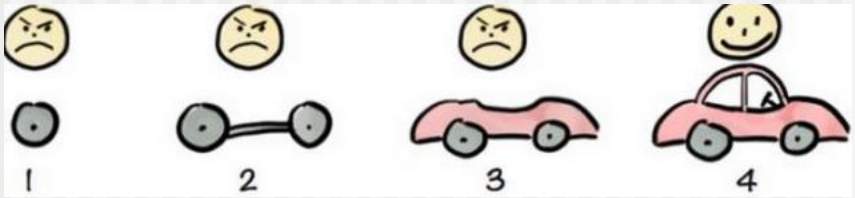


The Incremental Model

- There are many situations in which initial software requirements are reasonably well defined
- But we need to provide a limited set of software functionality to users quickly
  - Then refine and expand on that functionality in later releases
- In such cases, we choose a process model that is designed to produce the software in increments
  - Combines linear and parallel process flows

The Incremental Model



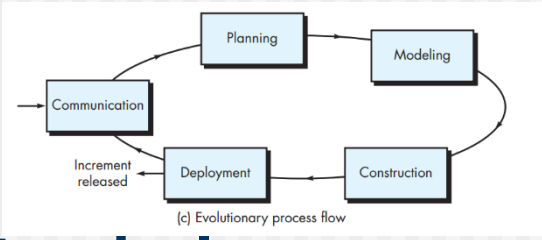


The Incremental Model

- Each linear sequence produces deliverable "increments" of the software
  - The first increment is often a **core product**
  - Basic requirements are addressed
  - But many supplementary features remain undelivered
- The core product is used by the customer
  - A working version will be available ~~late~~ **sooner**
- Based on evaluation results:
  - A plan is developed for the next increment
  - The core product is modified to better meet the customer needs and the delivery of additional features
- This process is repeated until the product is completed

Example

- Word-processing software developed using the incremental paradigm:
  - In the **first increment**: deliver basic file management, editing, and document production functions
  - In the **second increment**: more sophisticated editing and document production capabilities
  - In the **third increment**: spelling and grammar checking
  - In the **fourth increment**: advanced page layout capability

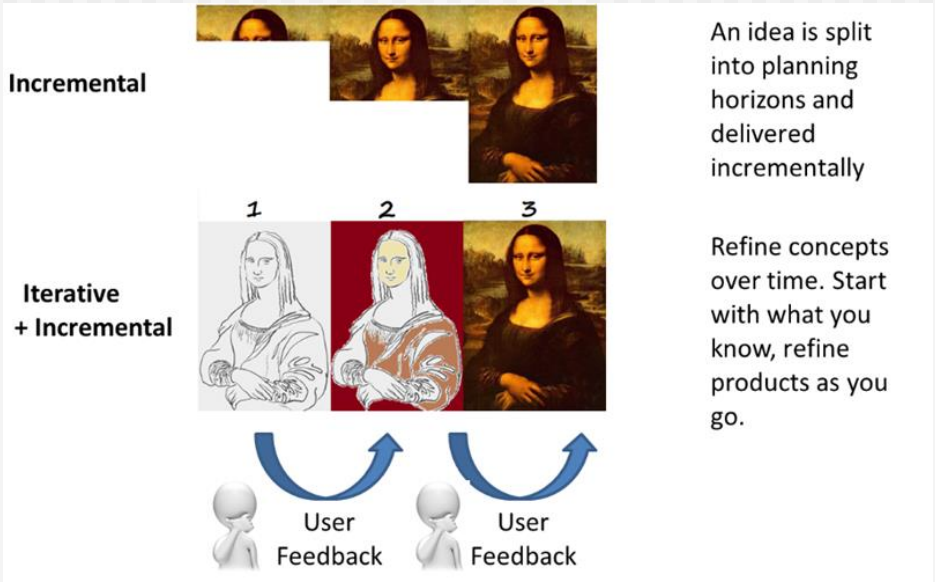


Evolutionary Process Models

- Software, like all complex systems, evolves over time
  - Grows and changes
- In these situations, you need an evolutionary model:
  1. Business and product requirements often change as development proceeds, making an end product unrealistic
  2. A set of basic requirements is well understood, but the details of product have yet to be defined (not soon)
- Evolutionary models are **iterative**
- Two common evolutionary models:
  - Prototyping
  - Spiral

Iterative vs. Incremental?

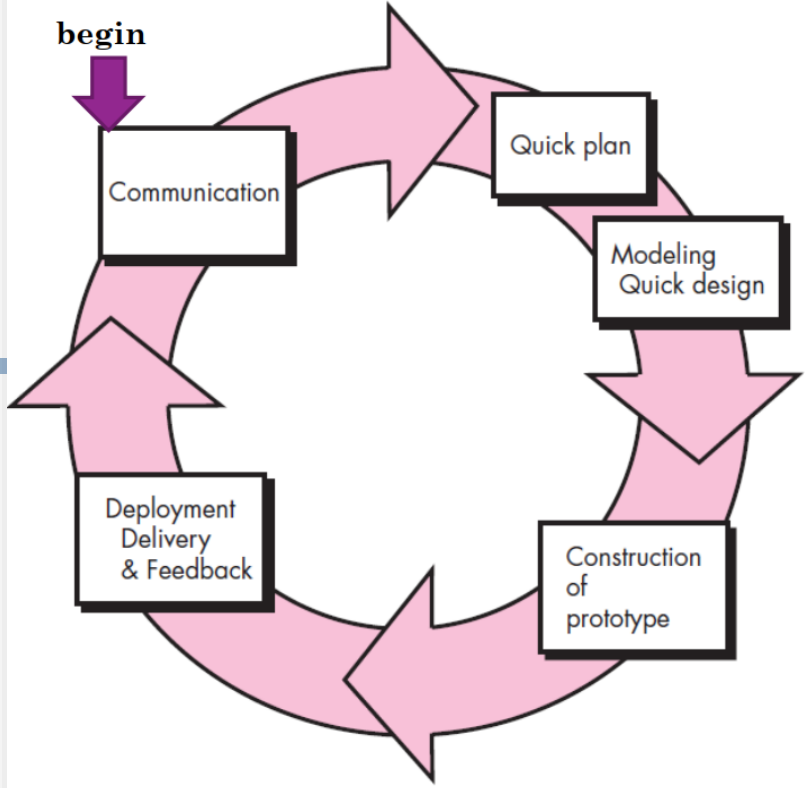
- An iterative process makes progress through continuous refinement
  - The final product may be quite different from the initial product
- An incremental process makes progress through small increments
  - Releasing small features at a time depending on their priorities



Evolutionary Models: Prototyping

- What is a prototype?
  - An early sample, model, or release of a product
- Benefit?
  - To get valuable **feedback** from the users early in the project
  - To be sure of the efficiency of an algorithm, the adaptability of an operating system, or …
- Better understand what is to be built when requirements are fuzzy
- It can be used within the context of any process model

Prototyping



- Two kinds of prototypes:

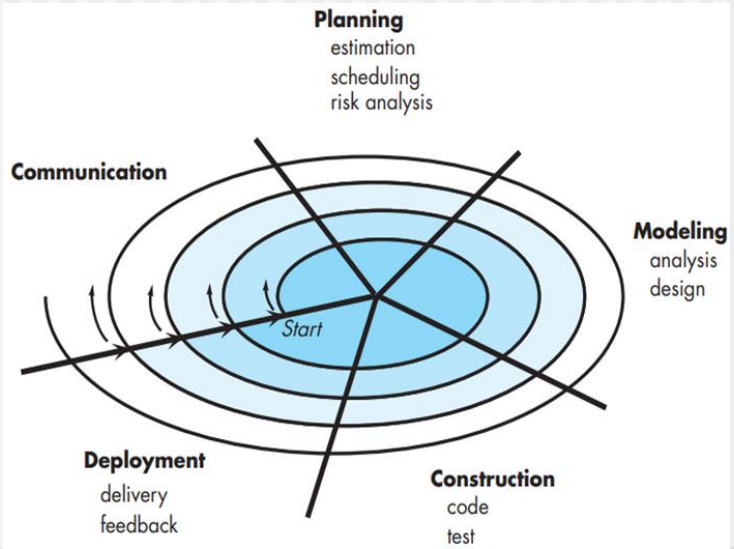
1. Throwaways
   - may be too slow, too big, awkward in use or all three
2. Evolutionary
   - slowly evolves into the actual system

- The problems of prototyping:

1. Stakeholders see what appears to be a working version of the software
2. As a software engineer, you often make implementation compromises in order to get a prototype working quickly

Evolutionary Models: The Spiral

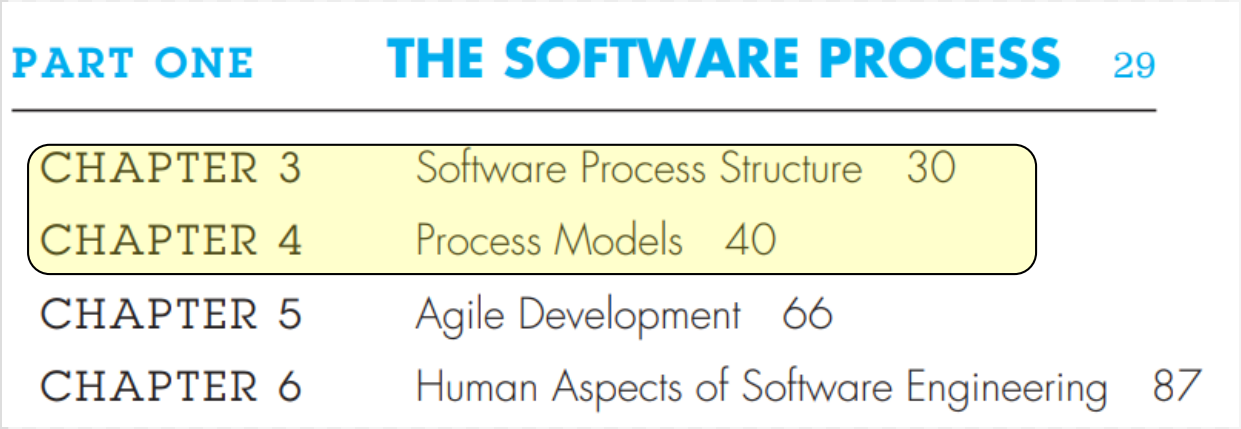
- Using the spiral model, software is developed in a series of iterations (i.e., evolutionary releases).
- During early iterations, the release might be a model or prototype.
- During later iterations, increasingly more complete versions of the engineered system are produced
- It is a **risk-driven** model
- Better understand and react to risks
- A realistic approach to the development of large-scale systems



Still Other Process Models

- Component based development—the process to apply when ***reuse*** is a development objective
- Formal methods—emphasizes the ***mathematical specification*** of requirements
- AOSD—provides a process and methodological approach for defining, specifying, designing, and constructing ***aspects***
- Unified Process—a "**use-case driven**, **architecture-centric**, **iterative** and **incremental**" software process closely aligned with the Unified Modeling Language (UML)

Further Reading



PART ONE THE SOFTWARE PROCESS 29

CHAPTER 3 Software Process Structure 30

CHAPTER 4 Process Models 40

CHAPTER 5 Agile Development 66

CHAPTER 6 Human Aspects of Software Engineering 87

The End

Further Study: Other Process Models

Component-Based Development

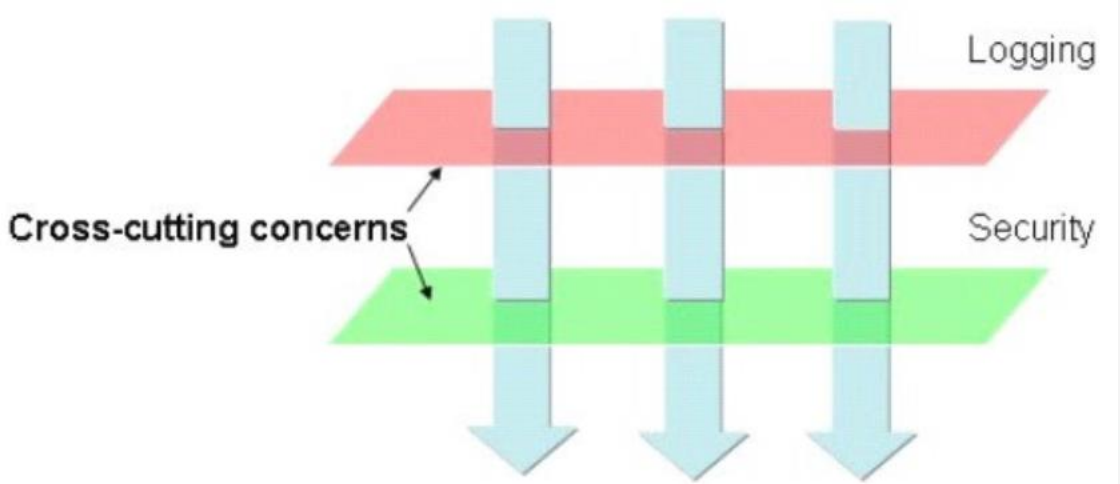
- The component-based development comprises applications from reusable components
- Commercial off-the-shelf (COTS) software components
  - Developed by vendors
  - Provides targeted functionality with well-defined interfaces that enable the component to be integrated into the software

The Formal Methods

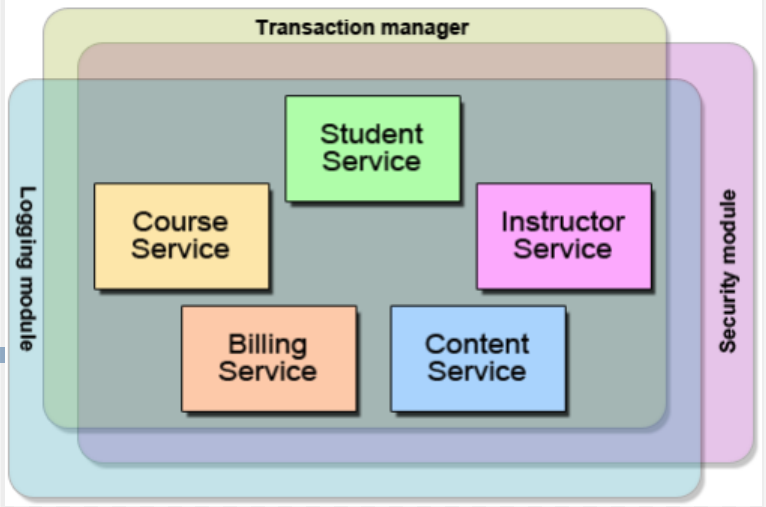
- Formal methods enable you to specify, develop, and verify a software by applying mathematical notation
- They provide a mechanism for eliminating many of the problems that are difficult to overcome using other software engineering paradigms.
  - Ambiguity, incompleteness, and inconsistency can be discovered and corrected more easily—not through ad hoc review, but through the application of mathematical analysis
- The formal methods approach has gained adherents for safety-critical software systems
  - e.g., aircraft avionics, medical devices, etc.

Aspect-Oriented Software Development (AOSD)

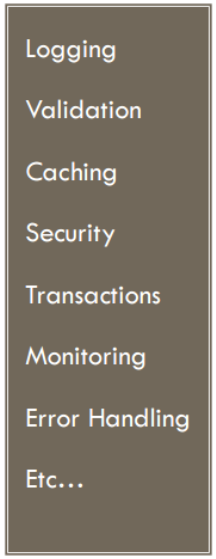
- Core concerns vs. Cross-cutting concerns
  - Core concerns: primary functionality of the system (business logic)
    - E.g., place a new order
  - Cross-cutting concerns: concerns that cut across multiple system functions, features, and information

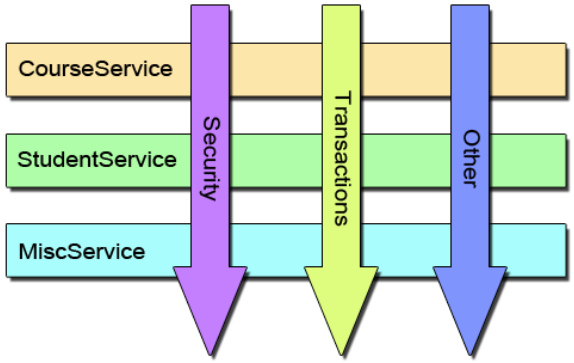


Aspects

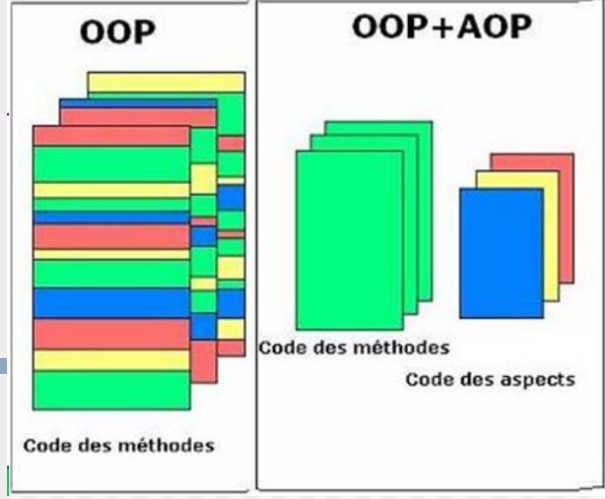


- An *aspect* is a representation of a cross-cutting concern.
- Example:
  - Authentication
  - Log
  - …
- Aspects modularize cross-cutting concerns that would otherwise end up scattered across several modules.





AOP vs. OOP



- AOP is **not a competitor** for OOP
  - it emerged out of OOP paradigm
  - AOP extends OOP by addressing few of its problems
  - AOP introduces neat ways to implement crosscutting concerns in a single place
    - which might have been scattered over several places in the corresponding OOP implementation
  - AOP makes the program cleaner and more loosely coupled
- So, it is **not** AOP **vs** OOP
  - It is AOP **with** OOP

AOP—An Example

- Note: Everything that AOP does could also be done without it by just adding more code
  - **AOP just saves you writing extra codes**
- Assume you have a graphical class with many "set...()" methods
- After each set method, the data of the graphics changed
  - thus the graphics need to be updated on screen
- Assume to repaint the graphics you must call "Display.update()“
- The classical approach is to solve this by adding ***more code***. At the end of each set method you write:

```
void set...(...) {
    :
    :
    Display.update();
}
```

AOP—An Example (cont.)

- If you have 3 set-methods, that is not a problem
- But if you have 200 (hypothetical), it's getting real painful to add this everywhere
- Also whenever you add a new set-method, you must be sure to not forget adding this to the end
  - otherwise you just created a bug
- AOP solves this without adding tons of code, instead you add an aspect:

```
after() : set() {
   Display.update();
}
```

- And that's it! Instead of writing the update code yourself, you just tell the system that after a set() pointcut has been reached, it must run this code.
  - No need to update 200 methods, no need to make sure you don't forget to add this code on a new set-method

AOP—An Example (cont.)

- Additionally you just need a pointcut:

```
pointcut set() : execution(* set*(*) ) && this(MyGraphicsClass) && within(com.company.*);
```

- That means:
  - if a method is named "set*" (* means any name might follow after set),
  - regardless of what the method returns or what parameters it takes
  - and it is a method of MyGraphicsClass
  - and this class is part of the package "com.company.*",
  - then this is a set() pointcut.
- And our first code (previous slide) says:
  - "after running any method that is a set() pointcut, run the following code.

AOP—An Example (cont.)

- Everything described in this example **can** be done at compile time.
- The pre-processor of AOP can just modify your source
  - e.g. adding Display.update() to the end of every set-pointcut method, before even compiling the class itself

Further Study: Risk Analysis

Assessing Project Risk

- Is project scope stable?
- Does the software engineering team have the right mix of skills?
- Are project requirements stable?
- Does the project team have experience with the technology to be implemented?
- Is the number of people on the project team adequate to do the job?
- Do all customer/user constituencies agree on the importance of the project and on the requirements for the system/product to be built?
- …

Risks Due to the Customer

Questions that must be answered:

- **Have you worked with the customer in the past?**
- **Has the customer agreed to spend time with you?**
- **Is the customer willing to participate in reviews?**
- **Is the customer technically sophisticated?**
- **Is the customer willing to let your people do their job—that is, will the customer resist looking over your shoulder during technically detailed work?**
- **Does the customer understand the software engineering process?**

Risks Due to Process Maturity

Questions that must be answered:

- **Have you established a common process framework?**
- **Is it followed by project teams?**
- **Do you have management support for software engineering ?**
- **Do you conduct formal technical reviews?**
- **Are CASE tools used for analysis, design and testing?**

Technology Risks

Questions that must be answered:

- **Is the technology new to your organization?**
- **Are new algorithms, I/O technology required?**
- **Is new or unproven hardware involved?**
- **Does the application interface with new software?**
- **Is a specialized user interface required?**
- **Are you using new software engineering methods?**
- **Are you using unconventional software development methods, such as formal methods, AI-based approaches, artificial neural networks?**

Staff/People Risks

Questions that must be answered:

- **Are the best people available?**
- **Does staff have the right skills?**
- **Are enough people available?**
- **Are staff committed for entire duration?**
- **Will some people work part time?**
- **Do staff have the right expectations?**
- **Have staff received necessary training?**
- **Will turnover among staff be low?**

Recording Risk Information

Project: Embedded software for XYZ system
Risk type: schedule risk
Priority (1 low ... 5 critical): 4
Risk factor: Project completion will depend on tests which require hardware component under development. Hardware component delivery may be delayed
Probability: 60 %
Impact: Project completion will be delayed for each day that hardware is unavailable for use in software testing
Monitoring approach:
Scheduled milestone reviews with hardware group
Contingency plan:
Modification of testing strategy to accommodate delay using software simulation
Estimated resources: 6 additional person months beginning in July