Software Requirements Modeling and Design

CS/SWE 321

Dr. Rob Pettit

Fall 2014

Course Logistics

- Web: http://cs.gmu.edu/~rpettit/swe321.html
  - Syllabus, schedule, and project information
  - Lecture notes updated weekly
- Blackboard
  - Assignments
- Piazza (https://piazza.com/gmu/fall2014/swe321/home)
  - Discussion board and announcements
- Office Hours: 8:00-9:00am Tu/Th in Engineering 4437 (Email to confirm)
  - Email Anytime: rpettit@gmu.edu
- Recommended Text:
  - Gomaa - "Software Modeling and Design"
- Recommended Software:
  - StarUML or Papyrus UML (via Eclipse)
- Prerequisites:
  - CS 211

Grading

- Project assignments (40%)
- Project Report (10%)
- Mid-term Exam (25%)
- Final exam (25%)

- Grading Scale:
  - 98+: A+
  - 92-97.9 : A
  - 90-91.9: A-
  - 88-89.9: B+
  - 82-87.9 : B
  - 80-81.9: B-
  - 78-79.9: C+
  - 72-77.9: C
  - 70-71.9: C-
  - 60-69.9: D
  - < 60 : F

About Me…

- Dr. Rob Pettit: email: rpettit@gmu.edu
  - B.S. Computer Science / Mathematics, University of Evansville
  - M.S. Software Systems Engineering, GMU
  - Ph.D. Information Technology / Software Engineering (Software Design and Architectural Analysis), GMU
  - The Aerospace Corporation
    - Lead Flight Software and Embedded Systems Office
    - Oversight of large real-time, object-oriented software analysis and design efforts for mission-critical systems
  - Teaching
    - GMU: SWE 621, SWE 626, SWE 632, CS/SWE 321
    - VT: CS5744, CS5704
  - Research Interests
    - Real-time object-oriented design
    - Software performance analysis

So, what's this course really about?

- From the GMU catalog:
- In a nutshell:
  - Introductory course to software engineering

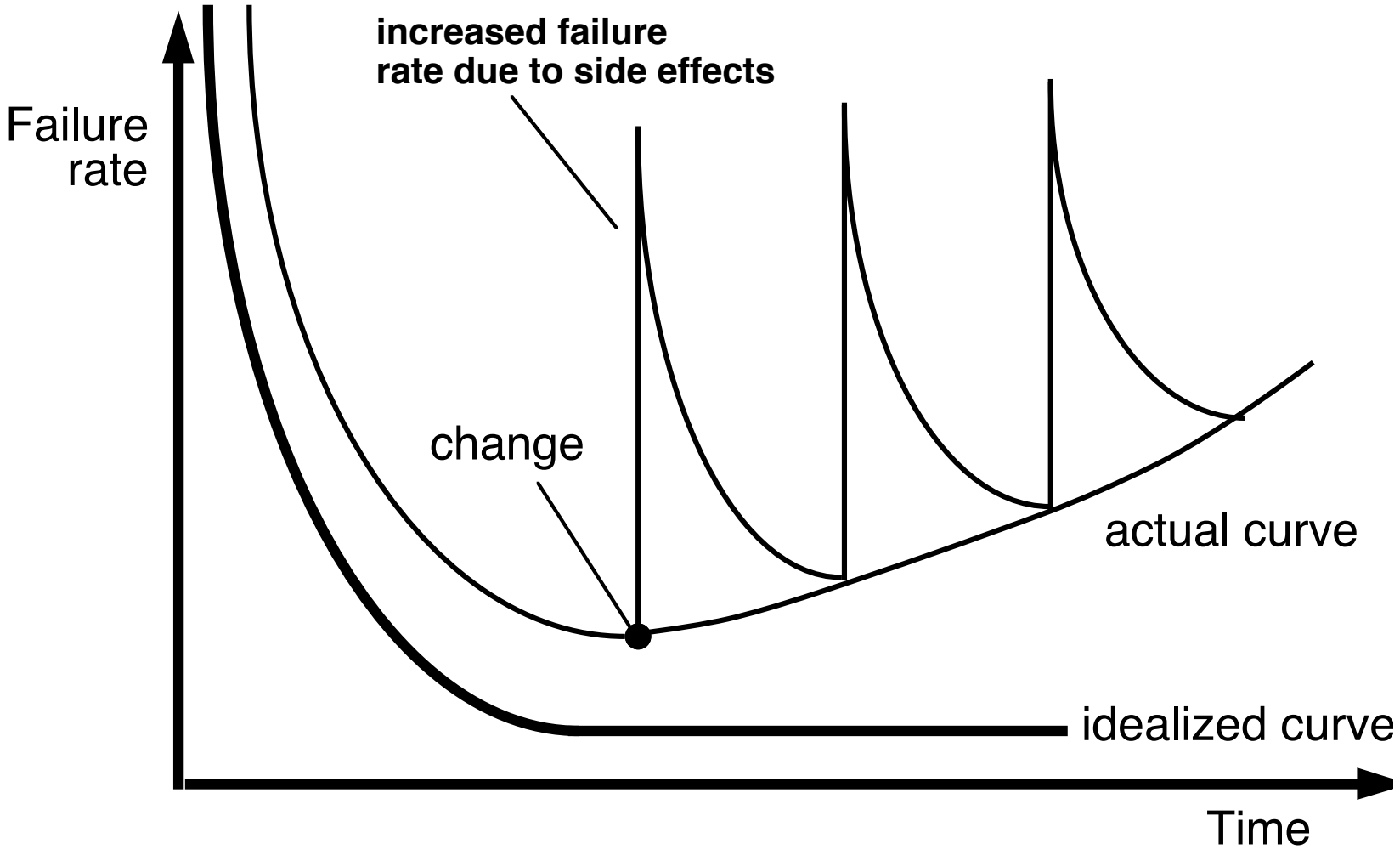
What is Software?

- More than just programs and code
  - Computer instructions
  - Data structures
  - Documentation
  - Models
- Program
  - Typically 50 -500 lines of code
  - Developed by one person
- Software system
  - Much larger, typically consisting of many programs working together
  - Needs a team of software engineers
  - Need project management and organization
  - Need a software life cycle
    - Phased approach to software development

What is Software?

- Software is developed or engineered
  - Not manufactured in the classical sense
- Software doesn't "wear out"
- Software is typically not mass produced
  - Lots of custom-built software
    - At least at the feature level

Wear vs. Deterioration



What is Engineering?

- Engineering is …
  - The application of scientific principles and methods to the construction of useful structures & machines
- Examples
  - Mechanical engineering
  - Civil engineering
  - Chemical engineering
  - Electrical engineering
  - Nuclear engineering
  - Aeronautical engineering

What is Software Engineering?

- Engineering
  - Applied Science
- Electrical engineering
  - Applied Physics
- Software Engineering
  - Applied Computer science

What is Software Engineering?

- The term is 40 years old
  - NATO Conference on “Software Crisis”
  - Garmisch, Germany, October 7-11, 1968

- Software Crisis
  - Software development projects were delivered late
  - Software was full of errors
  - Software did not satisfy requirements
  - Software was difficult to maintain

What is Software Engineering?

- IEEE (Institute of Electrical and Electronics Engineers) definition
  - "The application of a *systematic, disciplined, quantifiable* approach to the *development, operation* and *maintenance* of software, that is, the application of *engineering* to *software*".

- OR…
  - Software engineering is the establishment and use of sound engineering principles in order to obtain economically developed software that is reliable and works efficiently on real machines

Why Are There Difficulties?

- Software Engineering is a unique brand of engineering
  - Software is easy to change
  - Software construction is human-intensive
  - Software is intangible
  - Software problems are very complex
  - Software directly depends upon the hardware
    - It is at the top of the system engineering “food chain”
  - …

Software Processes

- Also known as Software Life Cycles
  - Phased approach to software development
  - Provide guidance on what must be created when
    - And (importantly) guidance on how to create and evaluate artifacts
- Generically consist of framework and umbrella activities

Framework Activities

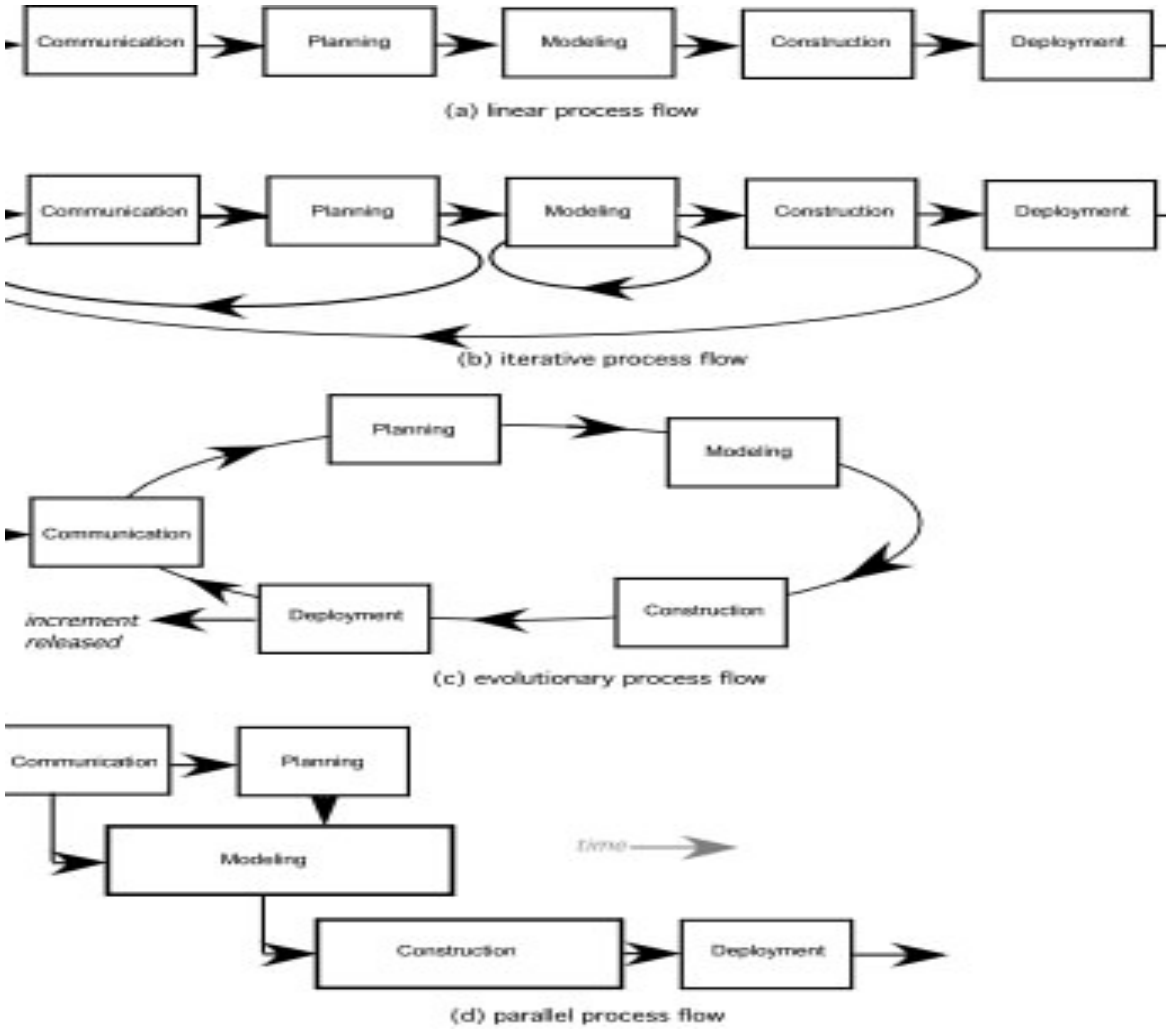
- Specific phases of the software development life cycle can be described in terms of:
  - Communication
  - Planning
  - Modeling
    - Analysis of requirements
    - Design
  - Construction
    - Code generation
    - Testing
  - Deployment
- Almost any software development process / life cycle can be described in terms of these framework activities.

Umbrella Activities

- Umbrella activities are performed throughout the life cycle phases.
  - Software project management
  - Formal technical reviews
  - Software quality assurance
  - Software configuration management
  - Work product preparation and production
  - Reusability management
  - Measurement
  - Risk management
- Umbrella activities focus on quality and management aspects

Process Flow

- Life cycle activities must be paired with a flow model
  - Identified when activities occur



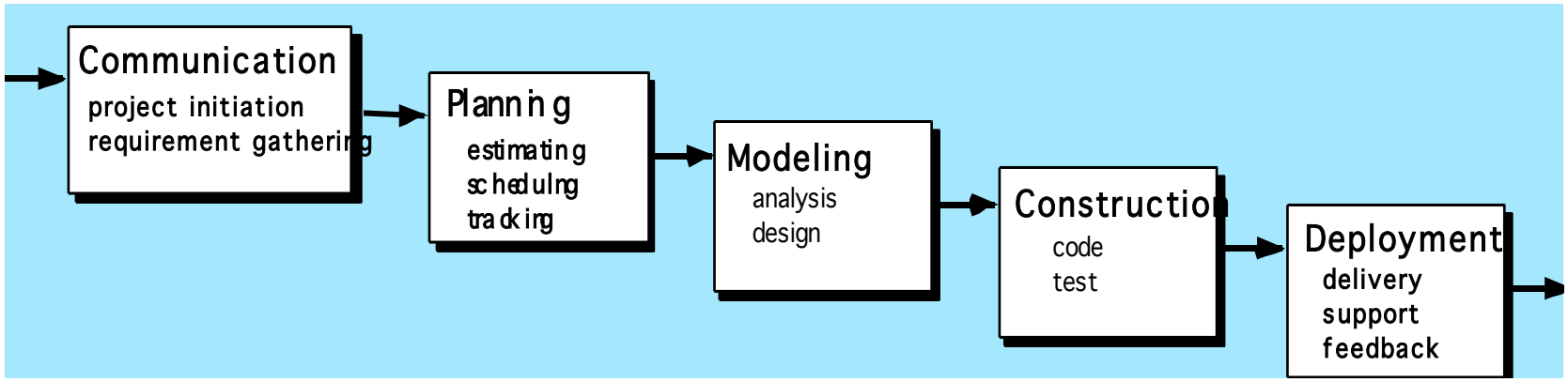
Adapting a Process Model

- Each software development effort must define the process to be used
- Often start with an "off the shelf" process and then tailor it to meet specific project needs
- Final, specific version to be applied is defined in the Software Development Plan (SDP)
- Factors for choosing and tailoring a process model include:
  - the criticality and nature of the system to be developed
  - the overall flow of activities, actions, and tasks
  - the degree to which work products are identified and required
  - the manner in which quality assurance activities are applied
  - the manner in which project tracking and control activities are applied
  - the overall degree of detail and rigor with which the process is described
  - 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

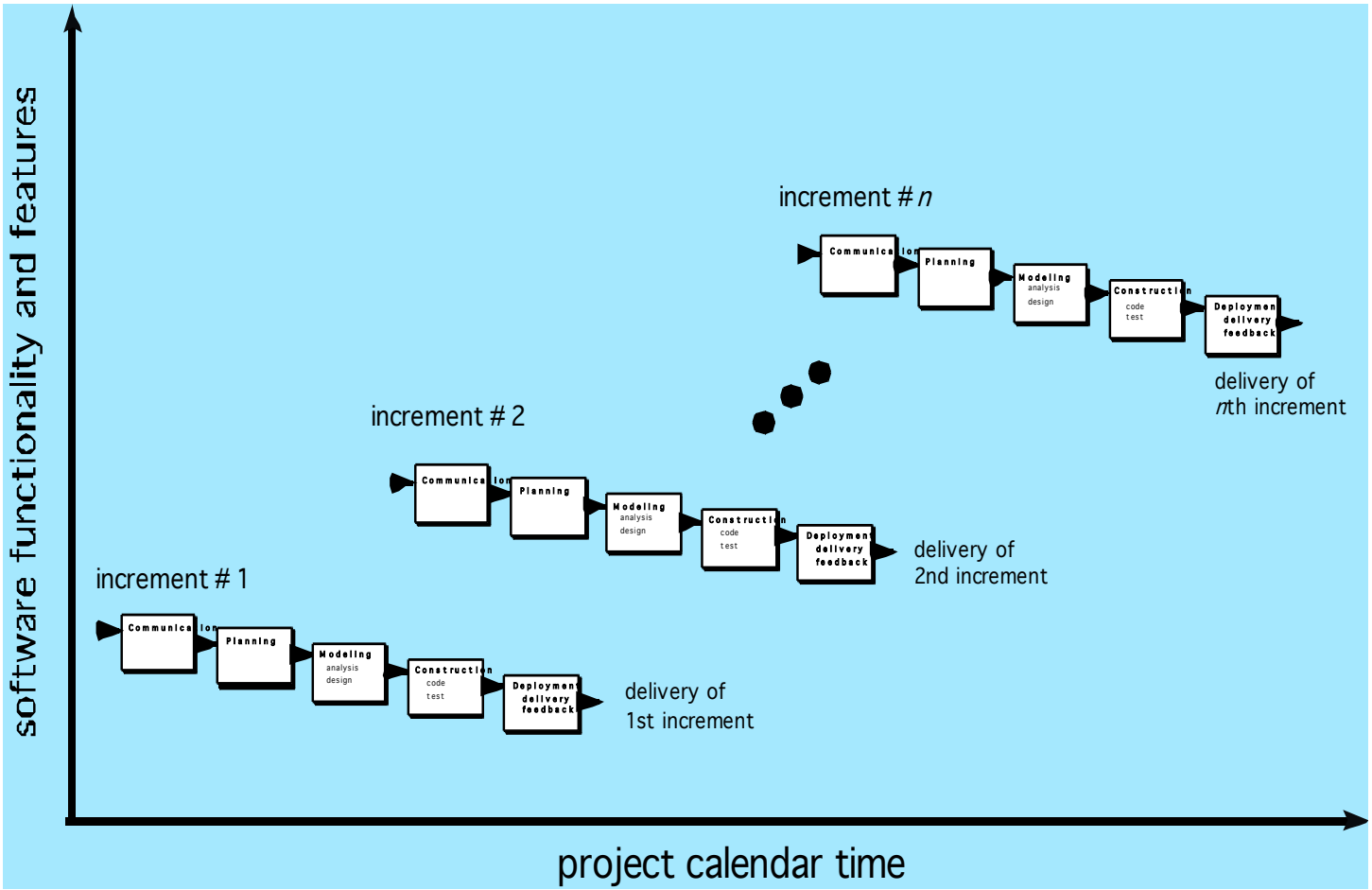
Prescriptive vs. Agile Process Models

- Prescriptive process models advocate an orderly approach to software engineering
  - Waterfall
  - Incremental
  - Evolutionary / Spiral
  - Unified Process
  - COMET (Gomaa book)
- Agile process models advocate flexibility and speed
  - XP (Extreme Programming)
  - Scrum
- Both types of process models have their place in software engineering

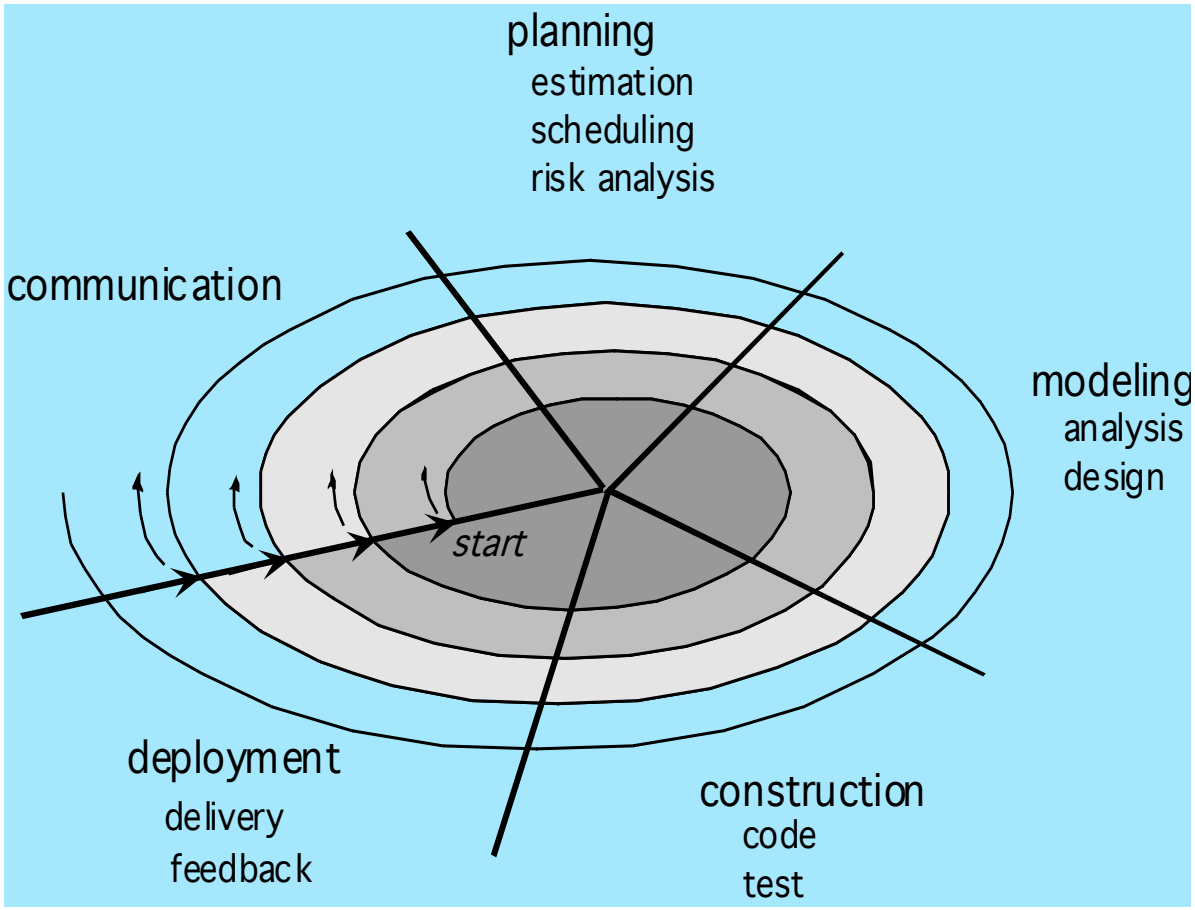
The Waterfall Model



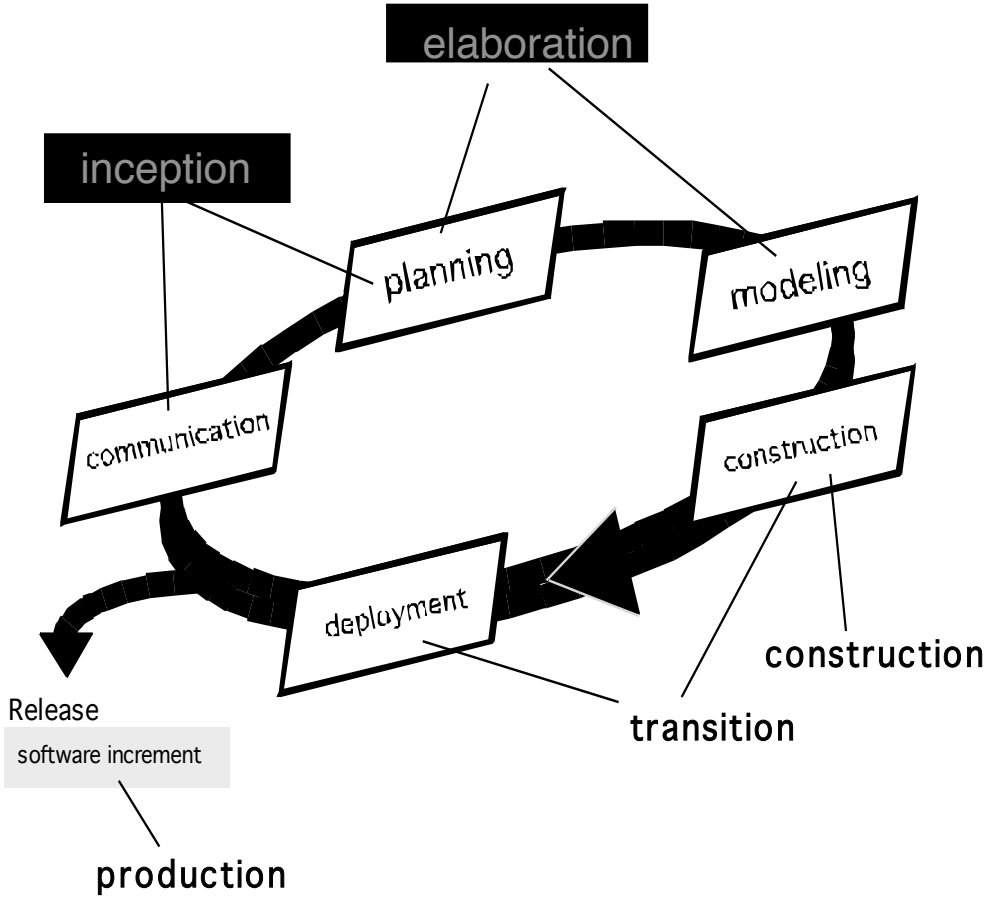
The Incremental Model



Evolutionary Models: The Spiral



The Unified Process (UP)



Collaborative Object Modeling and architectural design mEThod (COMET)

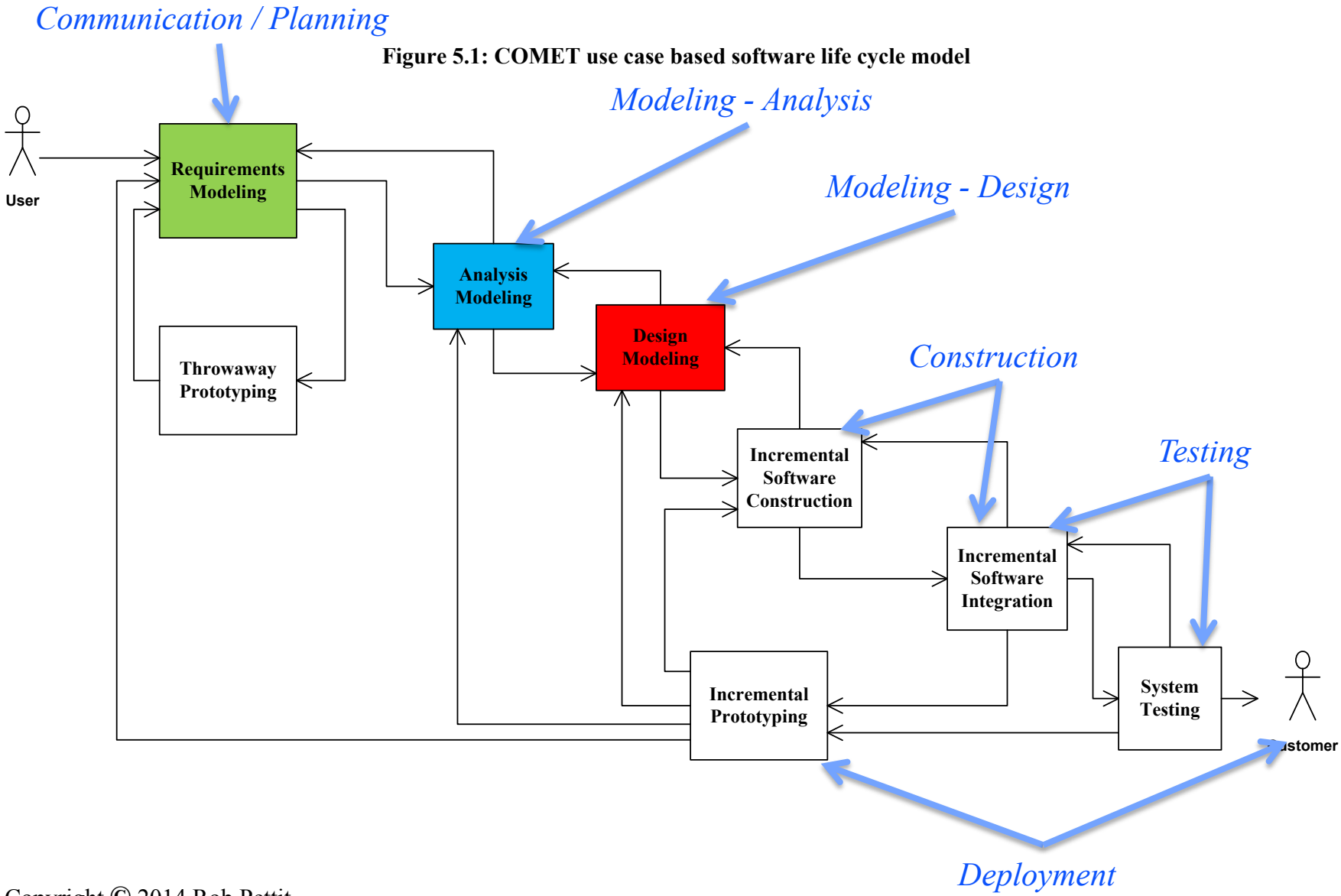


Figure 5.1: COMET use case based software life cycle model

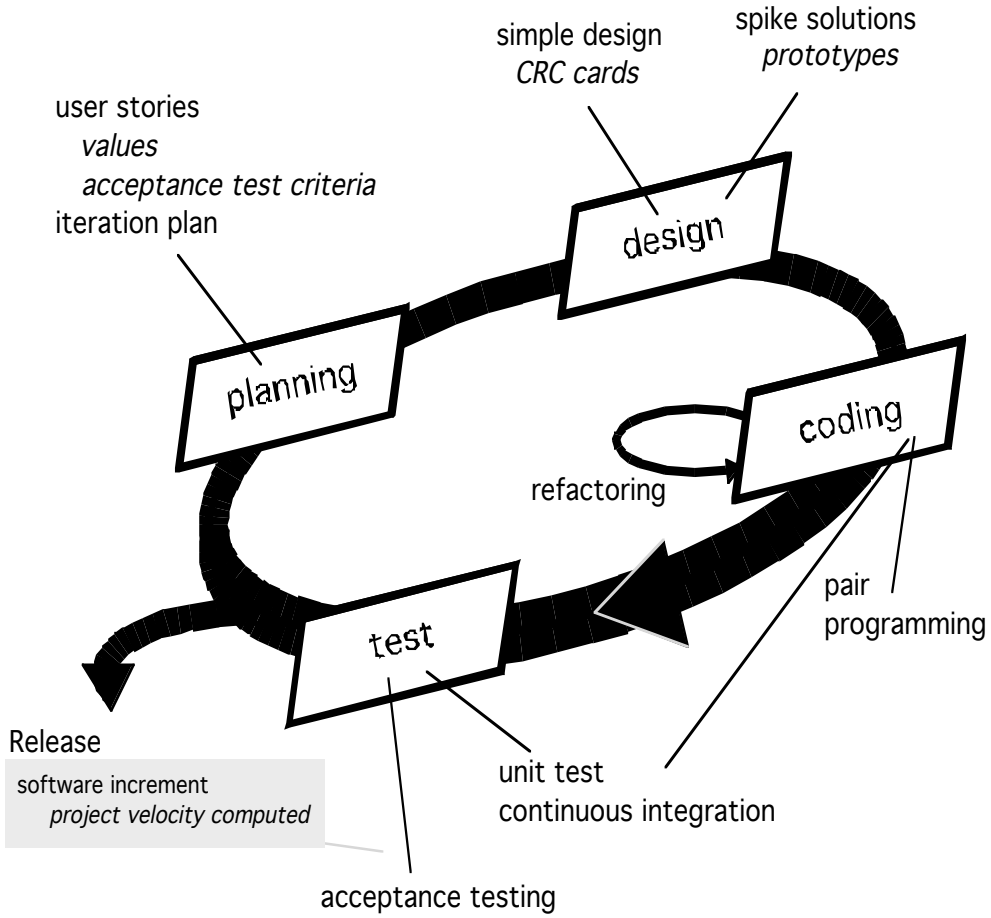
Agile Software Development

- Drivers:
  - Faster delivery of working software to customers without "excessive" process burdens
  - Avoidance of things that "waste time"
- Agile methods emphasize:
  - ***Individuals and interactions*** **over processes and tools**
  - ***Working software*** **over comprehensive documentation**
  - ***Customer collaboration*** **over contract negotiation**
  - ***Responding to change*** **over following a plan**

Extreme Programming (XP)

- The most widely used agile process, originally proposed by Kent Beck
- XP Planning
  - Begins with the creation of "user stories"
  - Agile team assesses each story and assigns a cost
  - Stories are grouped together for a deliverable increment
  - A commitment is made on delivery date
  - After the first increment "project velocity" is used to help define subsequent delivery dates for other increments

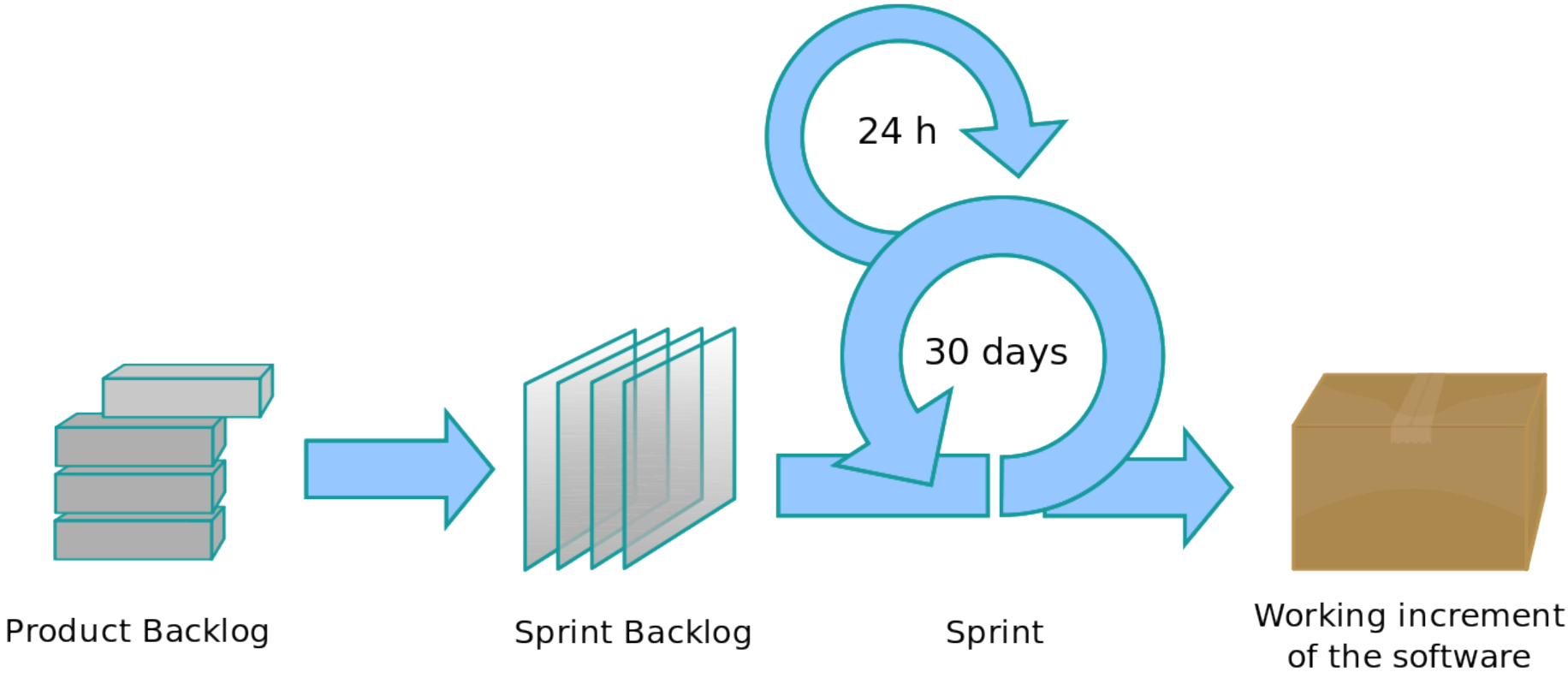
Extreme Programming (XP)



Scrum

- Originally proposed by Schwaber and Beedle
- Scrum—distinguishing features
  - Development work is partitioned into "packets"
  - Testing and documentation are on-going as the product is constructed
  - Work occurs in "sprints" and is derived from a "backlog" of existing requirements
  - Meetings are very short and sometimes conducted without chairs
  - "demos" are delivered to the customer with the time-box allocated

Scrum



Agile vs. Prescriptive Processes

Agile

1. Small products and teams; scalability limited
2. Untested on safety-critical products
3. Good for dynamic, but expensive for stable environments.
4. Require experienced Agile personnel throughout
5. Personnel thrive on freedom and chaos

Prescriptive

1. Large products and teams; hard to scale down
2. Handles highly critical products; hard to scale down
3. Good for stable, but expensive for dynamic environments
4. Require experienced personnel only at start if stable environment
5. Personnel thrive on structure and order

Review
Software Engineering in a Nutshell

- Development of software systems whose size/ complexity warrants team(s) of engineers
  - Multi-person construction of multi-version software
- Scope
  - Software process (life cycle)
  - Software development principles
  - Software methods and notations
- Goals
  - Production of quality software,
  - Delivered on time, within budget,
  - Satisfying customers' requirements and users' needs

SOFTWARE MODELING

Software Modeling and Design

- Origins of Modeling
  - Vitruvius, *De Architectura*, 1st century B.C.
  - Architectural models
- Modeling in science and engineering
  - Build model of system at some level of precision and detail
  - Analyze model to get better understanding of system
- Software Modeling
  - Modeling is designing of software applications before coding

The Need for Models

- A model is…
  - an *abstraction* that allows us to represent varying layers of complex information
- Models help us…
  - Organize
  - Communicate
  - Reason
  - Analyze

  Tools we need to develop and maintain complex software systems
- Tradeoffs
  - Larger effort
  - Delayed return
    - *Where's my SLOC?*
  - Additional skills required

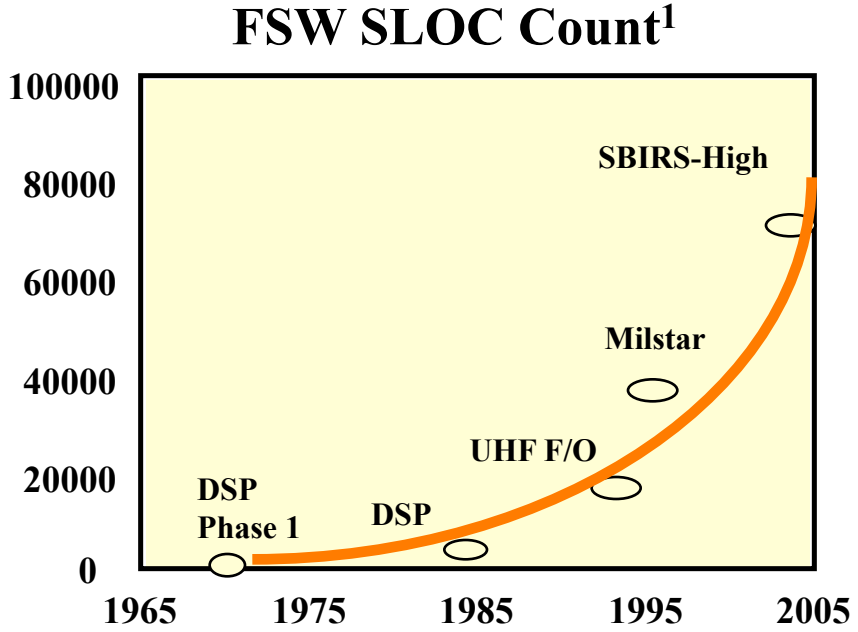
Why Do We Bother?

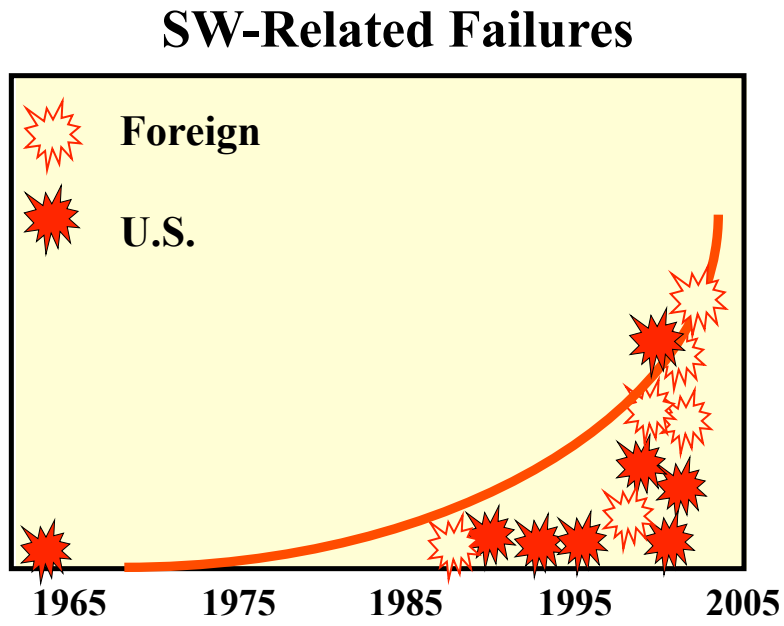
- Programming in the small is no longer feasible for most applications
- Software size is increasing exponentially
  - Example from space missions:
    - 1970's: 3 KSLOC
    - 1980's: 8 K
    - 1990's: 32 K
    - Current satellite system: multi-millions
  - Abstraction is essential to contain the complexity
- Most problems with software systems occur when different pieces have to interact
  - Still a poorly understood problem
  - Problems often discovered late and with great cost
  - Often leads to performance issues too
- **Half of all modern space system anomolies can be traced to software!**

Modeling and Analysis for Risk Mitigation

- Early modeling and analysis can reduce *incidental* complexity
  - FSW has inherent essentially complexity by nature
  - Incidental complexity arises from choices we make during requirements, architecture, design, and coding
- Model-based methods can
  - Ensure consistency from requirements ➔ architecture ➔ design ➔ code
  - Detect deviations from development standards
  - Assist trade studies in hardware/software architectures
  - Point to problems with performance and reliability in the early stages
  - Locate potential issues such as deadlocks and race conditions while they can still be repaired

Flight Software Impact on Mission Success





FSW SLOC = Flight Software Source Lines of Codes

- Software is growing in size and complexity
- Recent trends have seen significant grown in mission critical failures
- Approximately half of all modern space systems anomolies are related to software[2]

[1]Cheng, Paul, "Ground Software Errors Can Cause Satellites to Fail Too", *GSAW 2003*
[2]Hecht, Myron, and Douglas Buettner, "Software Testing in Space Programs", *Crosslink*, 6(3), Fall 2005

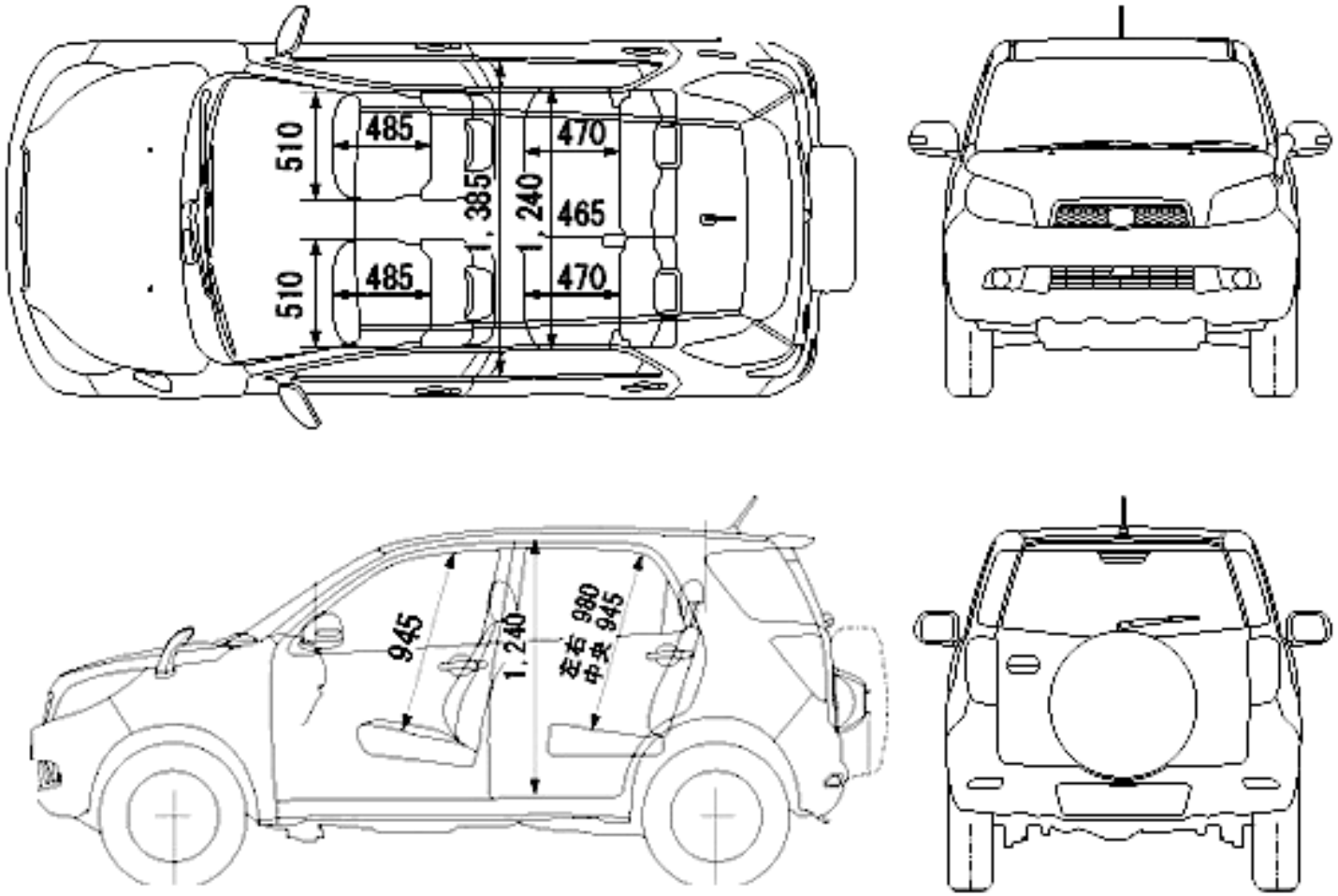
Model-Driven Software Engineering

- MDE, MDD, MDA, MDSE, …
- Software engineering has a long history of raising levels of abstraction
  - Binary⇨Assembly ⇨3GL ⇨OO Code ⇨UML ⇨Patterns ⇨…
- Models become primary artifacts of software development
  - May or may not include code generation
  - Much more rigorous models than previously used in software design

Unified Modeling Language (UML)

- UML is the standard modeling language for object-oriented software designs
  - Version 1.4 – large legacy base supported by many tools
  - Version 2.0 – recently adopted update, most modern tool releases now support this
  - Version 2.4.1 – absolute latest - limited tool support
- Types of Models Inclucde…
  - Use Case Models
    - Capture black-box functional requirements
  - Activity Models
    - Model detailed interactions within use case
  - Static Models
    - Capture structural elements
  - Dynamic Models
    - Capture behavioral elements

Different modeling views…



… help to understand different aspects of the system AND allow us to use abstraction to focus on one piece of the puzzle at a time

Overview of Software Modeling and Design Method

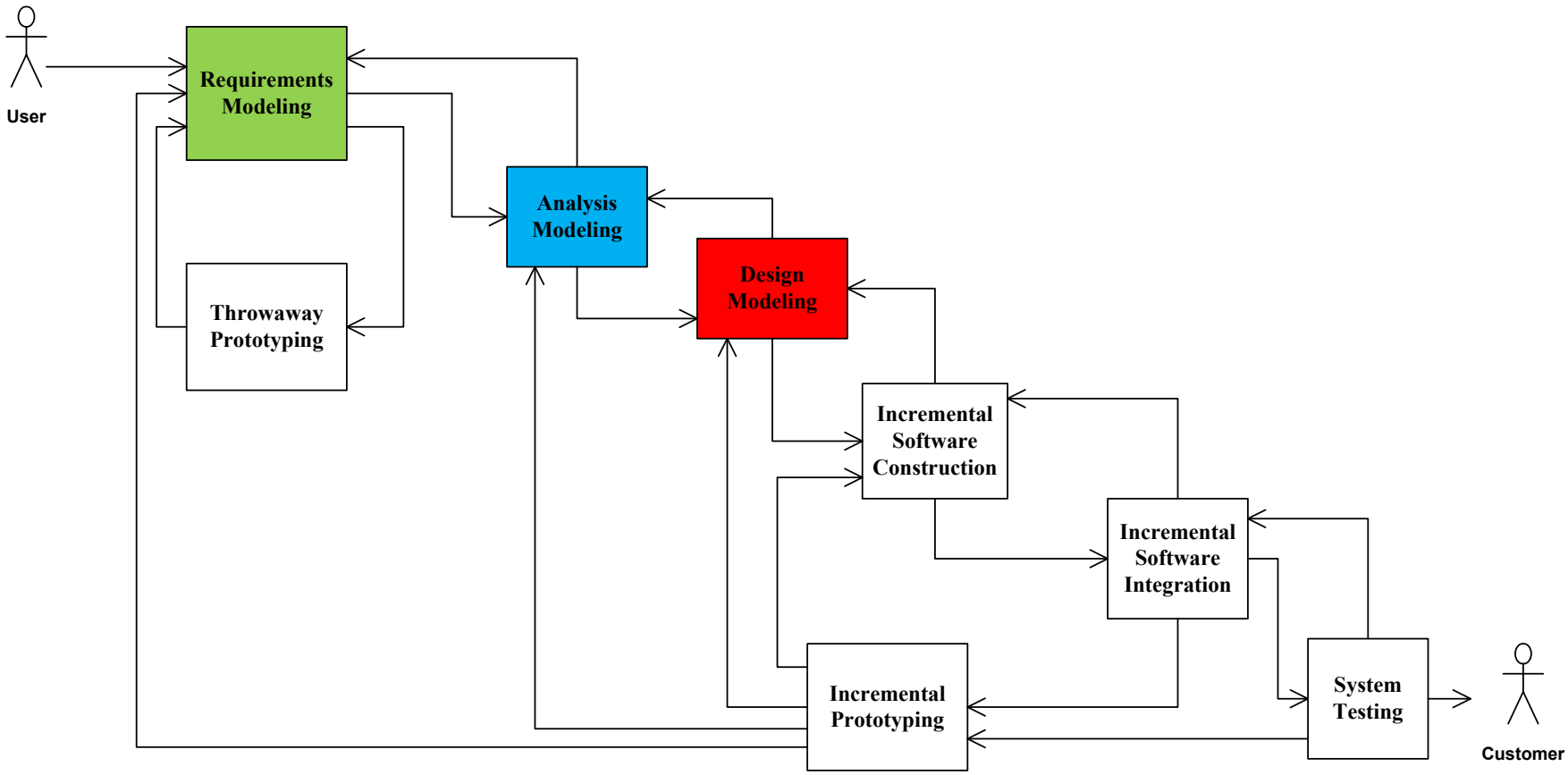
- Collaborative Object Modeling and architectural design mEThod (COMET)
  - Object Oriented Analysis and Design Method
  - Uses UML (Unified Modeling Language) notation
    - Standard approach for describing a software design
  - COMET = UML + Method

Overview of Software Modeling and Design Method

- Collaborative Object Modeling and architectural design mEThod (COMET)
  - Object Oriented Analysis and Design Method
  - Uses UML (Unified Modeling Language) notation
    - Standard approach for describing a software design
  - COMET = UML + Method
- **Provides steps and guidelines for**
  - **Software Modeling and Design**
  - **From Use Case Models to Software Architecture**
- **Course text: H. Gomaa, *Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures*, Cambridge University Press, 2011**

Figure 5.1 COMET use case based software life cycle model

Figure 5.1: COMET use case based software life cycle model



Requirements Modeling

- Use Case Modeling
  - Define software functional requirements in terms of use cases and actors

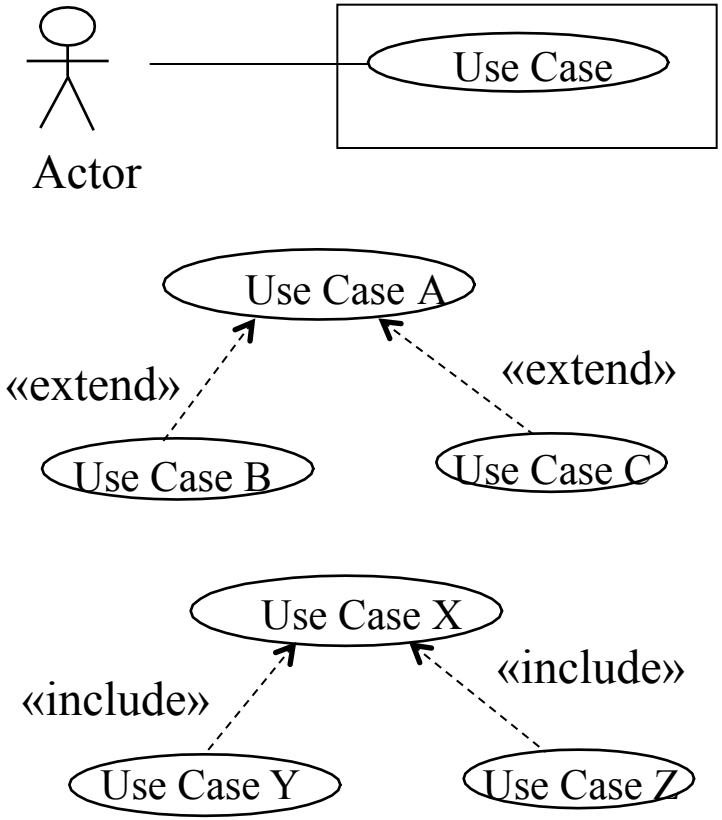
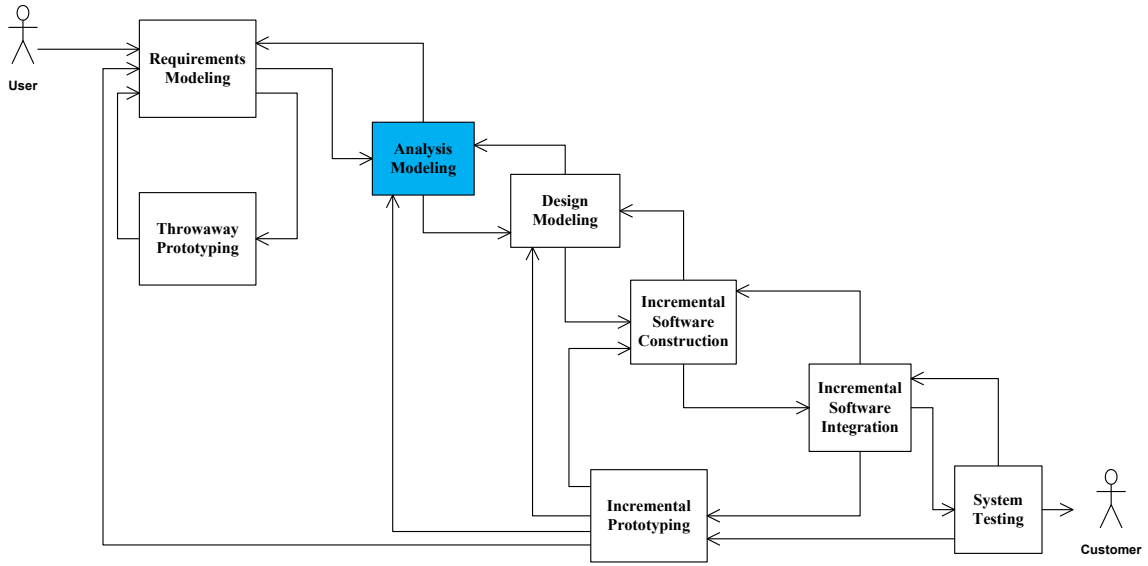


Figure 2.1 UML notation for use case diagram

Analysis Modeling

- Analysis Modeling consists of
  - Static Modeling
    - View of system that **does not** change with time
  - Dynamic Modeling
    - View of system that **does** change with time

Figure 5.1: COMET use case based software life cycle model



Analysis Modeling

- Static Modeling
  - Define structural relationships between classes
  - Depict classes and their relationships on class diagrams

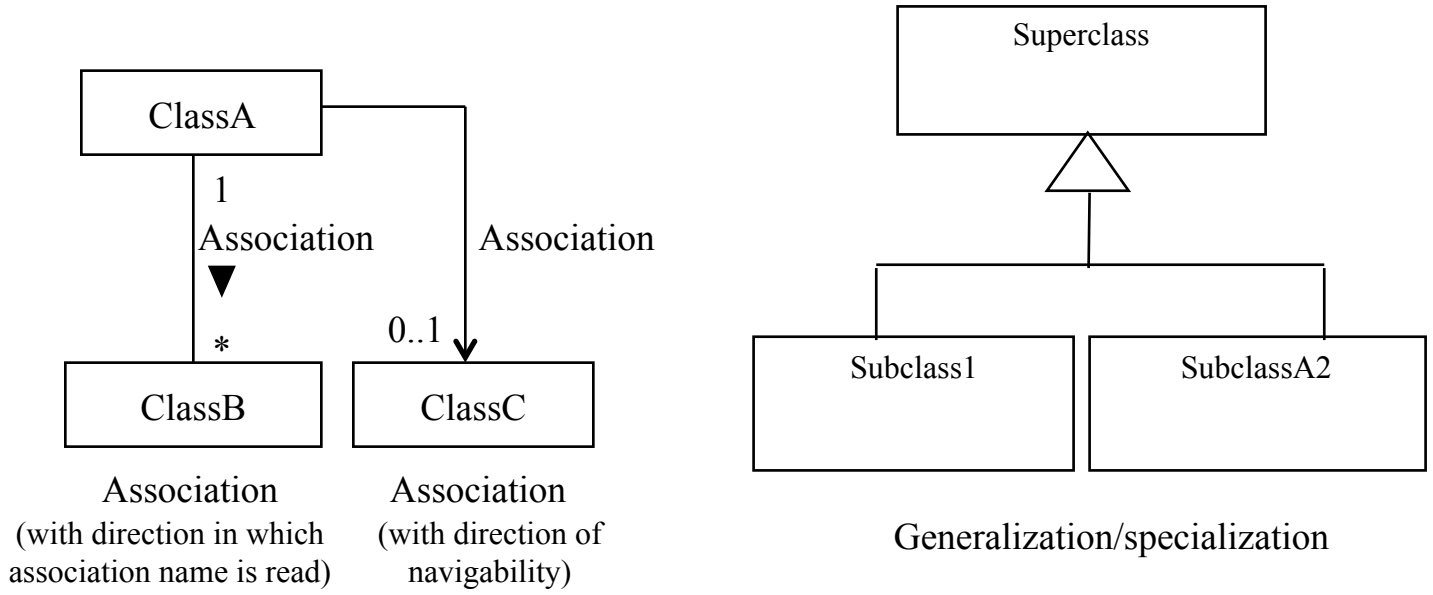
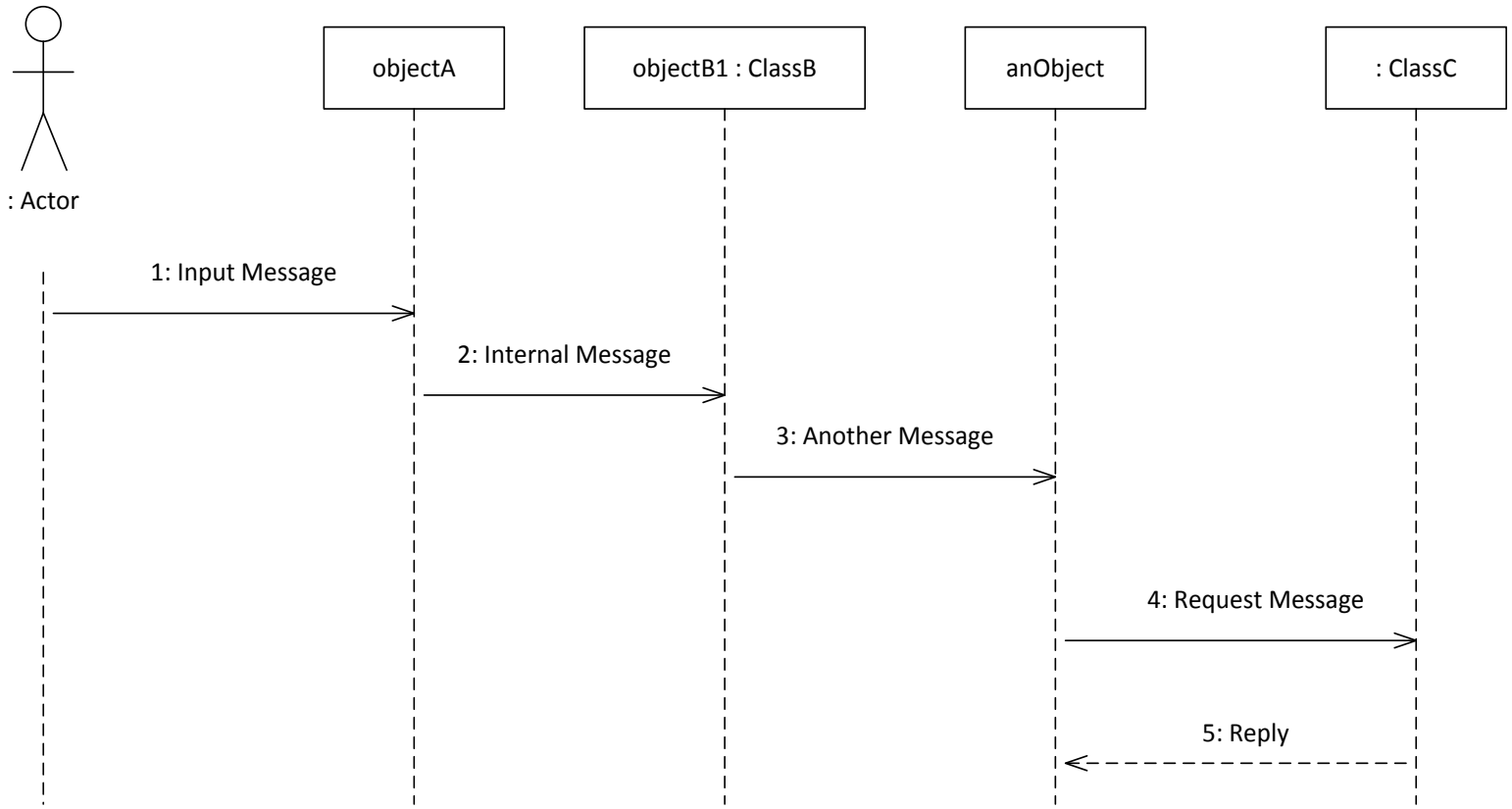


Figure 2.3 UML notation for classes

Analysis Modeling

- **Dynamic Modeling**
  - Defines sequence of objects communicating with each other using communication diagrams or sequence diagrams

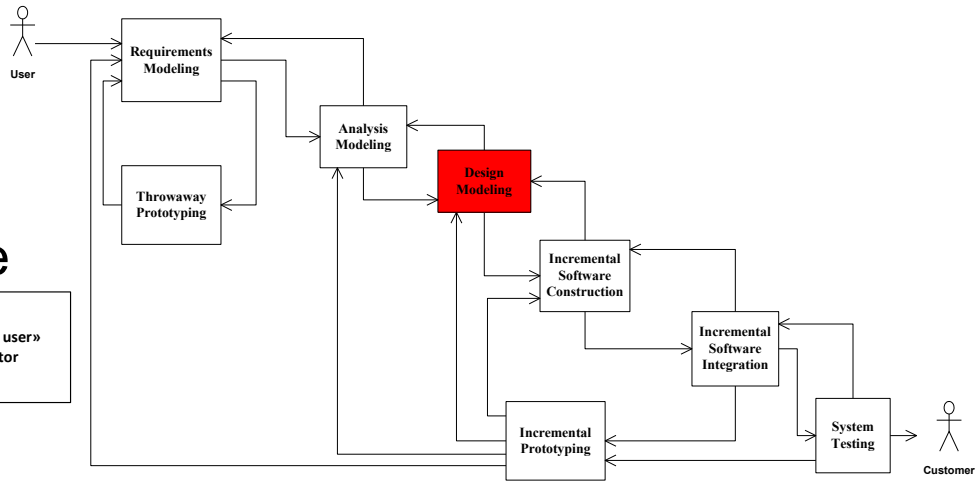
Figure 2.6: UML notation for sequence diagram

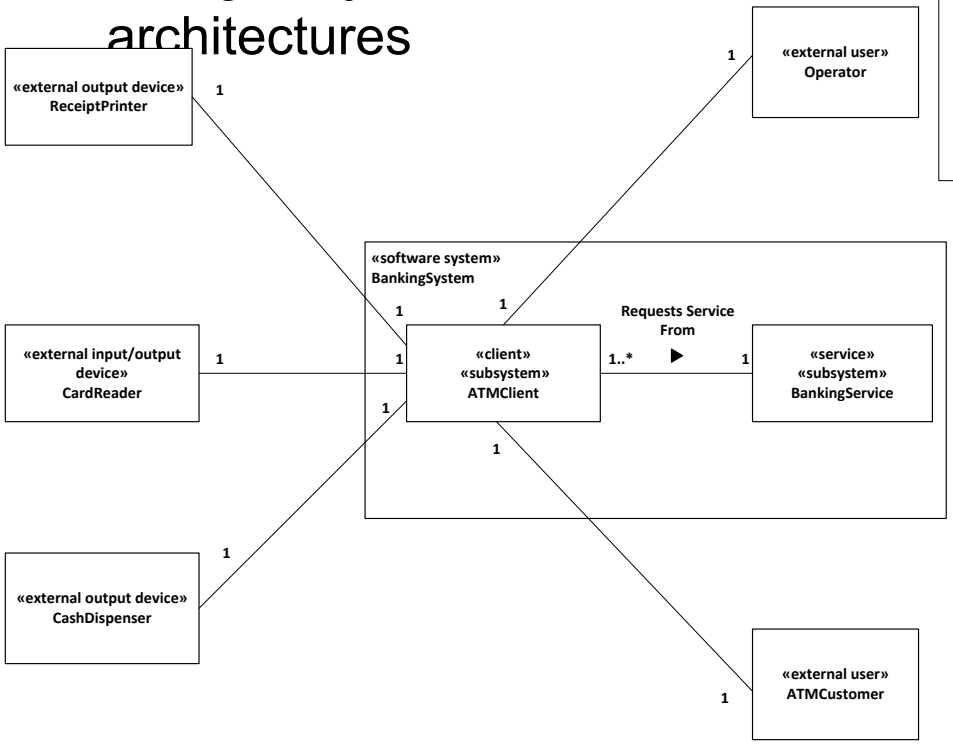


Design Modeling

- Develop overall software architecture
  - Structure system into subsystems
  - Design object-oriented software architectures

Figure 5.1: COMET use case based software life cycle model

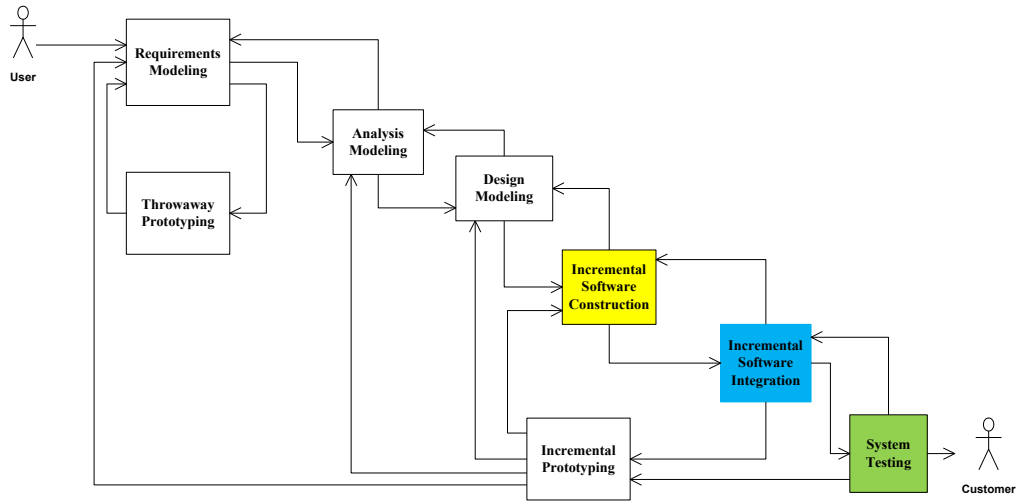




COMET Software Life Cycle

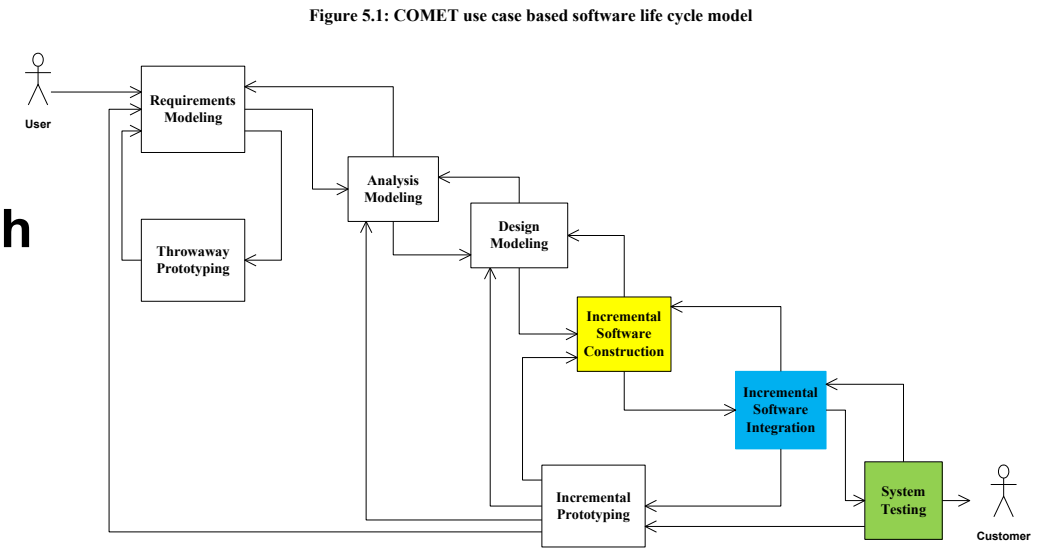
- Incremental Software Construction
  - Select subset of system based on use cases
  - For each class in subset
    - Detailed design in Pseudocode
      - Structured English
    - Coding
      - E.g., Java
    - Unit test
      - Test individual objects
      - (instantiated from classes)

Figure 5.1: COMET use case based software life cycle model



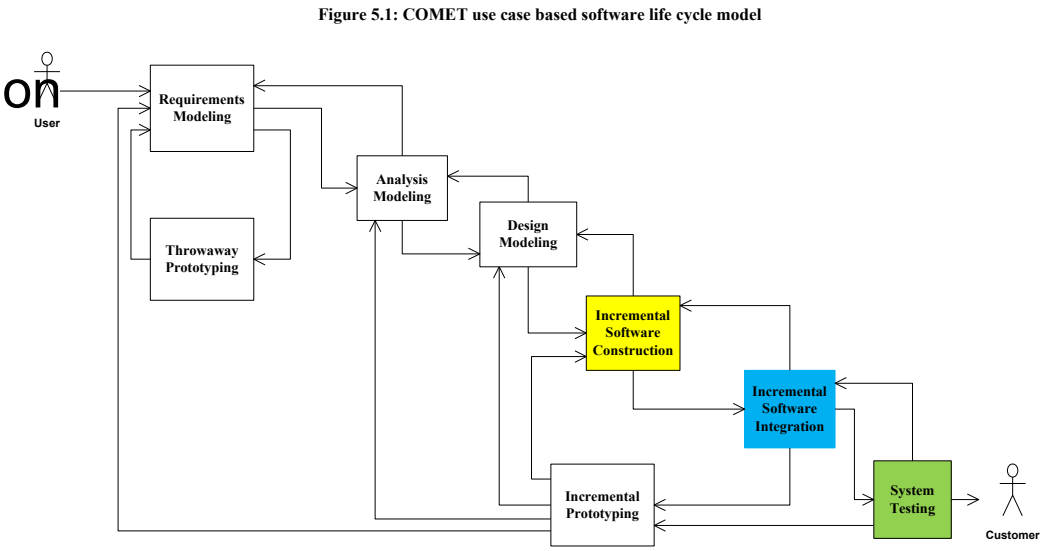
COMET Software Life Cycle

- Incremental Software Construction
- **Incremental Software Integration**
  - **Integration testing of each system increment**
  - **Integration test based on use cases**



COMET Software Life Cycle

- Incremental Software Construction
- Incremental Software Integration
- **System Testing**
  - **Testing of software functional requirements**
  - **Based on use cases**



For Next Week…

- Complete individual bio sketch (see Blackboard assignment)
- Form teams of 4-5 students per team
  - Complete team formation report
  - *Team leader* – email report to me before class next Tuesday.

Backup material

ADDITIONAL DEFINITIONS

Software Engineering ≠ Software Programming

- Software programming
  - Single developer
  - Small applications
  - Short lifespan
  - Single or few stakeholders
    - Architect = Developer = Manager = Tester = Customer = User
  - One-of-a-kind systems
  - Built from scratch
  - Minimal maintenance

Software Engineering ≠ Software Programming

- Software engineering
  - Teams of developers with multiple roles
  - Complex systems
  - Indefinite lifespan
  - Numerous stakeholders
    - Architect ≠ Developer ≠ Manager ≠ Tester ≠ Customer ≠ User
  - System families
  - Reuse to amortize costs
  - Maintenance can account for over 60% of overall development costs

Definitions

- Systematic
  - Characterized by the use of order and planning
- Disciplined
  - Controlled, managed, kept within certain bounds
- Quantifiable
  - Measureable

Definitions

- Software development
  - The production of software
  - From analyzing user requirements to testing of software

Definitions

- Software development
  - The production of software
  - From analyzing user requirements to testing of software
- **Operation**
  - **Environment in which software runs:**
    - **Hardware platform (e.g., PC, Mac)**
    - **Operating system (e.g., Windows, Linux)**
    - **Networks**
  - **Software deployment**
    - **Installation of working software**

Definitions

- Software development
  - The production of software
  - From analyzing user requirements to testing of software
- Operation
  - Environment in which software runs:
    - Hardware platform (e.g., PC, Mac)
    - Operating system (e.g., Windows, Linux)
    - Networks
  - Software deployment
    - Installation of working software
- **Maintenance**
  - **Modification of software after installation**