Software Lifecycle and Team Programming

Arun Chauhan

COMP 314

Recap of the Last Lecture

BFS

```
for each vertex v in V
       color[v] = white
       d[v] = INFINITY
       p[v] = NULL
color[s] = gray
d[s] = 0
Queue.clear()
Queue.put(s)
while (!Queue.empty())
       v = Queue.get()
       for each u adjacent to v
            if (color[u] == white)
                color[u] = gray
                d[u] = d[v] + 1
                p[u] = v
                Queue.put(u)
       color[v] = black
```

Lemmas

Lemmas

Lemma 1: Let $G = (V, E)$ be a graph, and $s \in V$ a vertex. Then, for any edge $(u, v) \in E$:

$$b(s, v) \leq b(s, u) + 1$$

Lemmas

Lemma 1: Let $G = (V, E)$ be a graph, and $s \in V$ a vertex. Then, for any edge $(u, v) \in E$:

$$b(s, v) \leq b(s, u) + 1$$

Lemma 2: Upon termination, the BFS algorithm computes $d[v]$ for every vertex, and $d[v] \geq b(s, v)$.

Lemmas

Lemma 1: Let $G = (V, E)$ be a graph, and $s \in V$ a vertex. Then, for any edge $(u, v) \in E$:

$$b(s, v) \leq b(s, u) + 1$$

Lemma 2: Upon termination, the BFS algorithm computes $d[v]$ for every vertex, and $d[v] \geq b(s, v)$.

Lemma 3: At all times during the execution of BFS, the queue contains vertices $(v_1, v_2, \ldots v_r)$ such that $d[v_1] \leq d[v_2] \leq d[v_3] \ldots \leq d[v_r]$ AND $d[v_r] \leq d[v_1] + 1$.

Lemmas

Lemma 1: Let $G = (V, E)$ be a graph, and $s \in V$ a vertex. Then, for any edge $(u, v) \in E$:

$$b(s, v) \leq b(s, u) + 1$$

Lemma 2: Upon termination, the BFS algorithm computes $d[v]$ for every vertex, and $d[v] \geq b(s, v)$.

Lemma 3: At all times during the execution of BFS, the queue contains vertices $(v_1,\ v_2,\ \ldots v_r)$ such that $d[v_1] \leq d[v_2] \leq d[v_3] \ldots \leq d[v_r]$ AND $d[v_r] \leq d[v_1] + 1$.

Corollary 4: If vertices u and v are enqueued during execution of BFS, and u is enqueued before v, then $d[u] \leq d[v]$.

Theorem

Theorem: Given $G = (V, E)$ and source vertex s, the BFS algorithm discovers every vertex v reachable from s, and upon termination, $d[v] = b(s, v)$. Moreover, for any vertex v reachable from s, one of the shortest paths from s to v is a path from s to $p[v]$, followed by edge $(p[v], v)$.

Lecture 2

What this Lecture is About

- applied aspects of data-structures and algorithms
- "software engineering"
  - issues in developing large software
  - techniques for managing software development
- software lifecycle
- project management
- extreme programming

State of the Software

If software were an office building, it would be built by a thousand carpenters, electricians and plumbers. Without architects. Or blueprints. It would look spectacular, but inside, the elevators would fail regularly. Thieves would have unfettered access through open vents at street level. Tenants would need consultants to move in. They would discover that the doors unlock whenever someone brews a pot of coffee. The builders would provide a repair kit and promise that such idiosyncrasies would not exist in the next skyscraper they build (which, by the way, tenants will be forced to move into).

Strangely, the tenants would be OK with all this. They'd tolerate the costs and the oddly comforting rhythm of failure and repair that came to dominate their lives. If someone asked, "Why do we put up with this building?" shoulders would be shrugged, hands tossed and sighs heaved. "That's just how it is. Basically, buildings suck."

from an online article on idg.net

Does Software Really Suck?

- slight exaggeration, but not far from the fact
- examples of software failures abound
- but, sofware companies still expect to make money!

Software Failures

Year 1900 Bug

In 1992, Mary from Winona, Minnesota, received an invitation to attend a kindergarten. Mary was 104 at that time.

Software Failures

Interface Misuse

On April 10, 1990, in London, an underground train left the station without its driver. The driver had taped the button that started the train, relying on the system that prevented the train from moving when doors were open. The train operator had left his train to close a door which was stuck. When the door was finally shut, the train simply left.

Software Failures

Late and Over Budget

In 1995, bugs in the automated luggage system of the new Denver International Airport caused suitcases to be chewed up. The airport opened 16 months late, $3.2 billion over-budget, with mostly manual luggage system.

Software Failures

On-Time Delivery

After 18 months of development, a \$200 million system was delivered to a health insurance company in Wisconsin in 1984. However, the system did not work correctly; \$60 million in overpayments were issued. The system took 3 years to fix.

Software Failures

Unnecessary Complexity

The C-17 cargo plane by McDonnel Douglas ran $500 million over budget because of problems with its avionics software. The C-17 included 19 onboard computers, 80 microprocessors, and 6 different programming languages.

Lessons

- we rely more and more on software in our daily lives
- software mistakes are costly
- software reliability is critical
- software usability is very important
- software projects are mostly ill-managed

Why Software Engineering?

- software engineering coined in the late 1960s because:

  > Software developers were not able to set concrete objectives, predict the resources necessary to attain those objectives, and manage the customers' expectations. More often than not, the moon was promised, a lunar rover built, and a pair of square wheels delivered.

Why Software Engineering?

- software engineering coined in the late 1960s because:

  > Software developers were not able to set concrete objectives, predict the resources necessary to attain those objectives, and manage the customers' expectations. More often than not, the moon was promised, a lunar rover built, and a pair of square wheels delivered.

- Arguably, large pieces of software are the most complicated entities ever created by humans!

What is Engineering?

- well established engineering disciplines are applications of natural sciences
- engineering is a quick way to design objects
  - civil engineers have widely used “handbooks” to guide construction
- engineering is a collection of best practices
- engineering is a collection of design patterns

Activities in a Large Project

Activities in a Large Project

- modeling

Activities in a Large Project

- modeling
- problem solving

Activities in a Large Project

- modeling
- problem solving
- knowledge acquisition

Activities in a Large Project

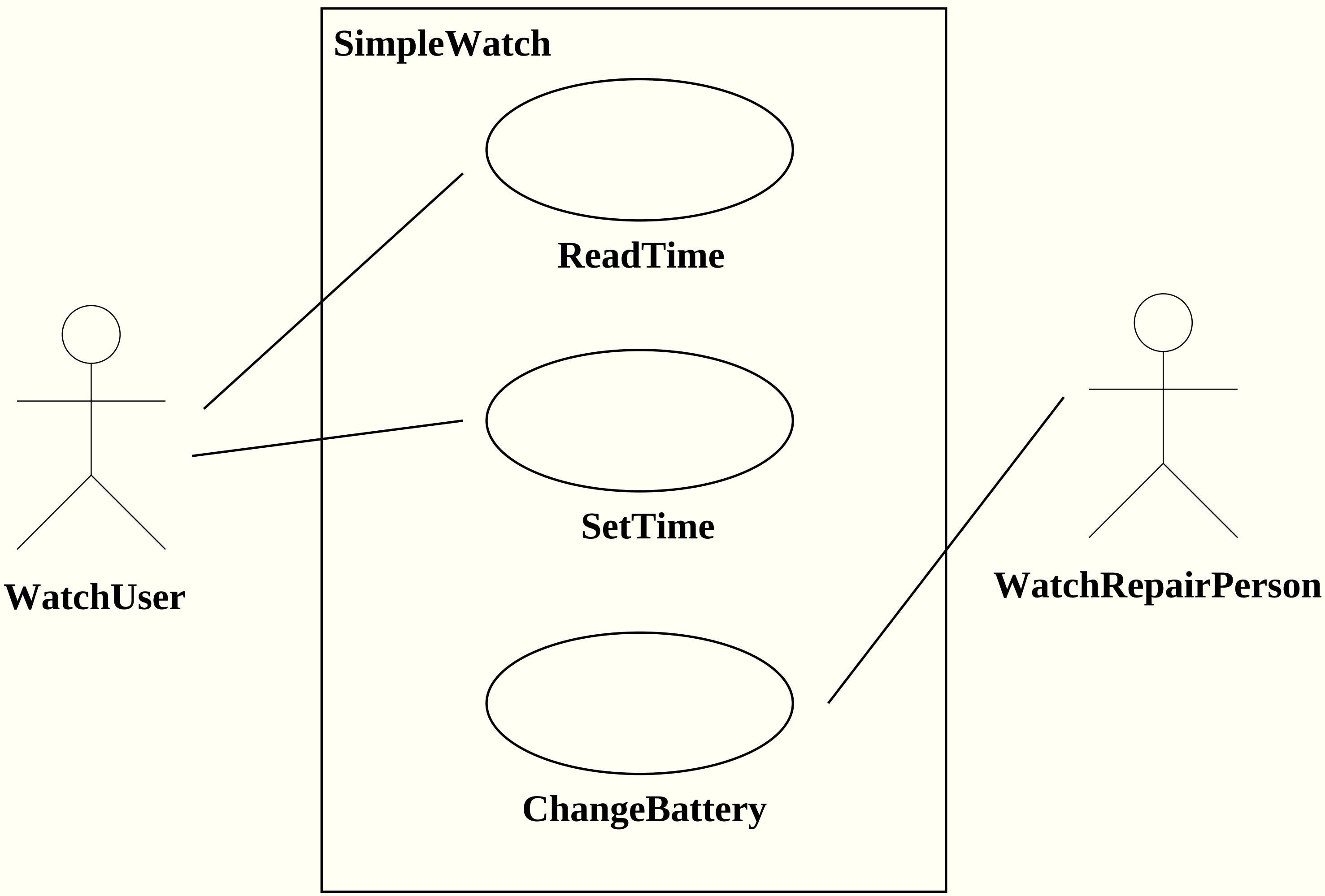
- modeling
- problem solving
- knowledge acquisition
- rationale management

Modeling with UML

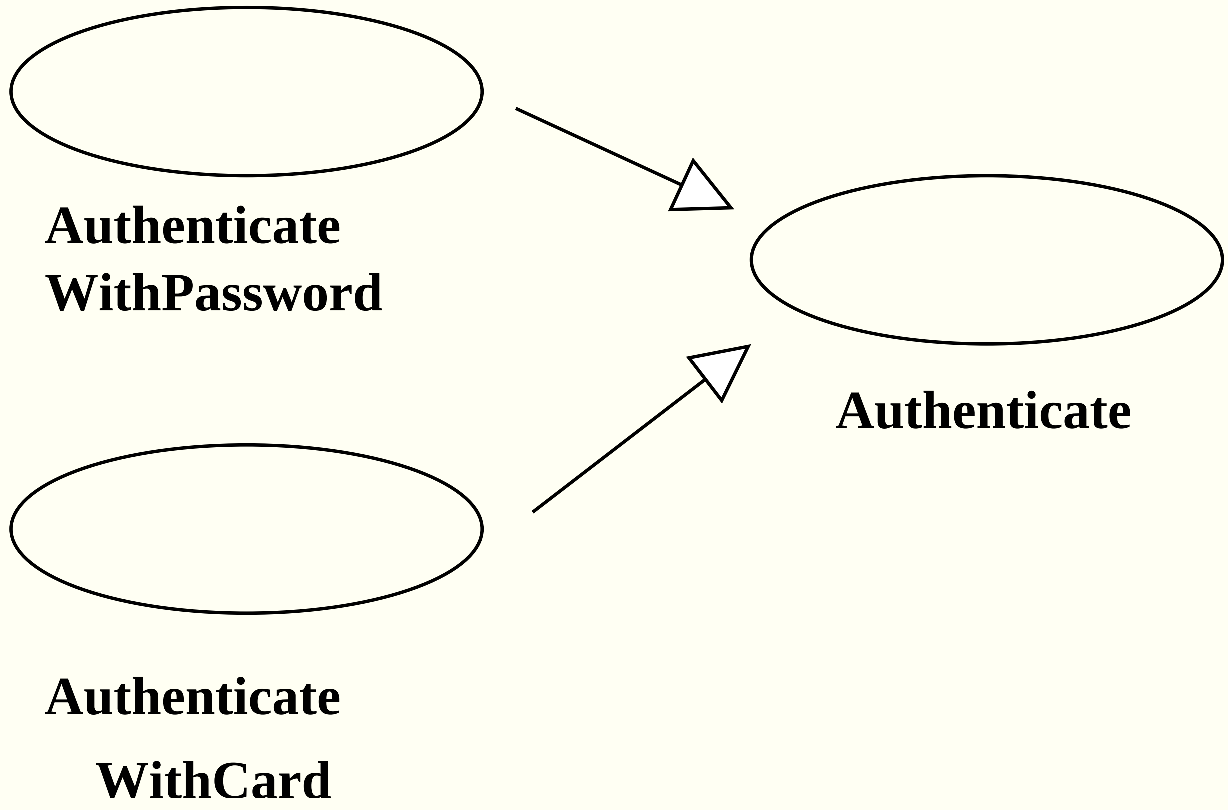
Modeling With UML

- functional model
  - UML use case models
  - functionality of the system from user's point of view
- object model
  - UML class diagrams
  - structure of a system in terms of objects, attributes, associations, and operations
- dynamic model
  - UML sequence diagrams, statechart diagrams, activity diagrams
  - internal behavior of the system

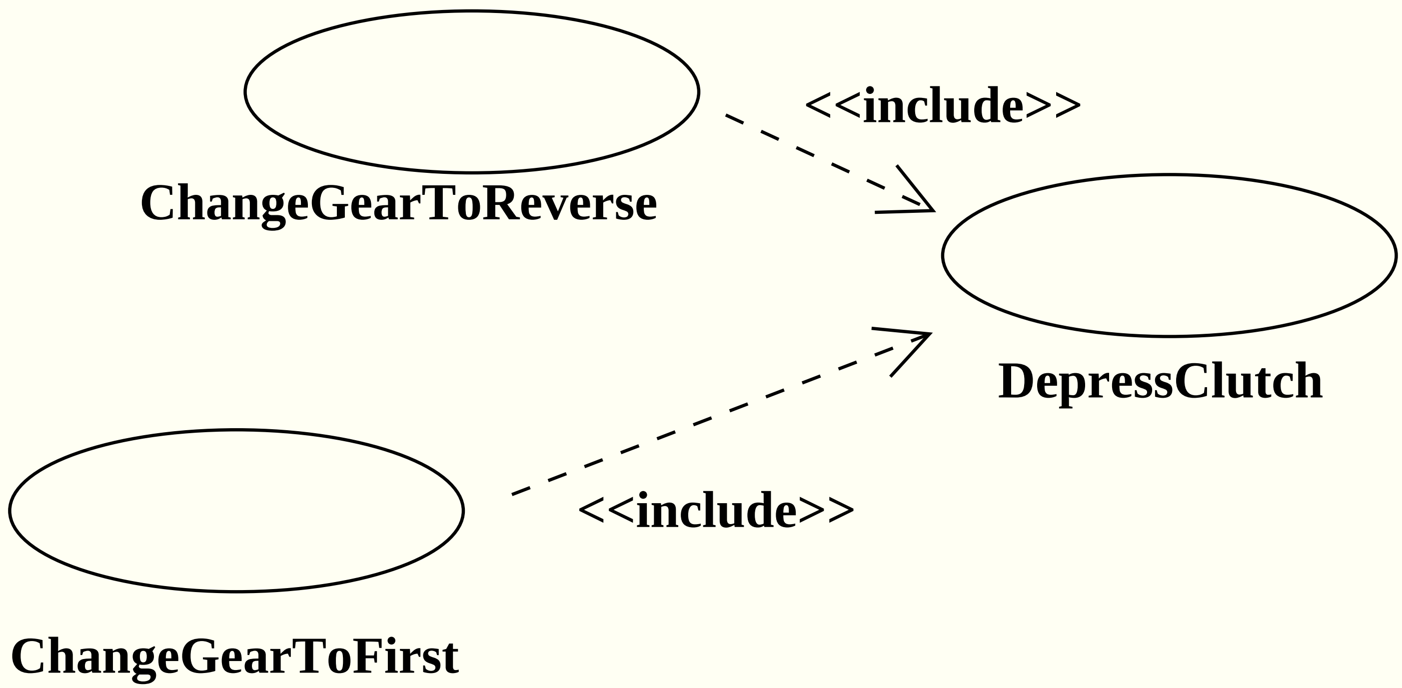
Use Case Diagrams



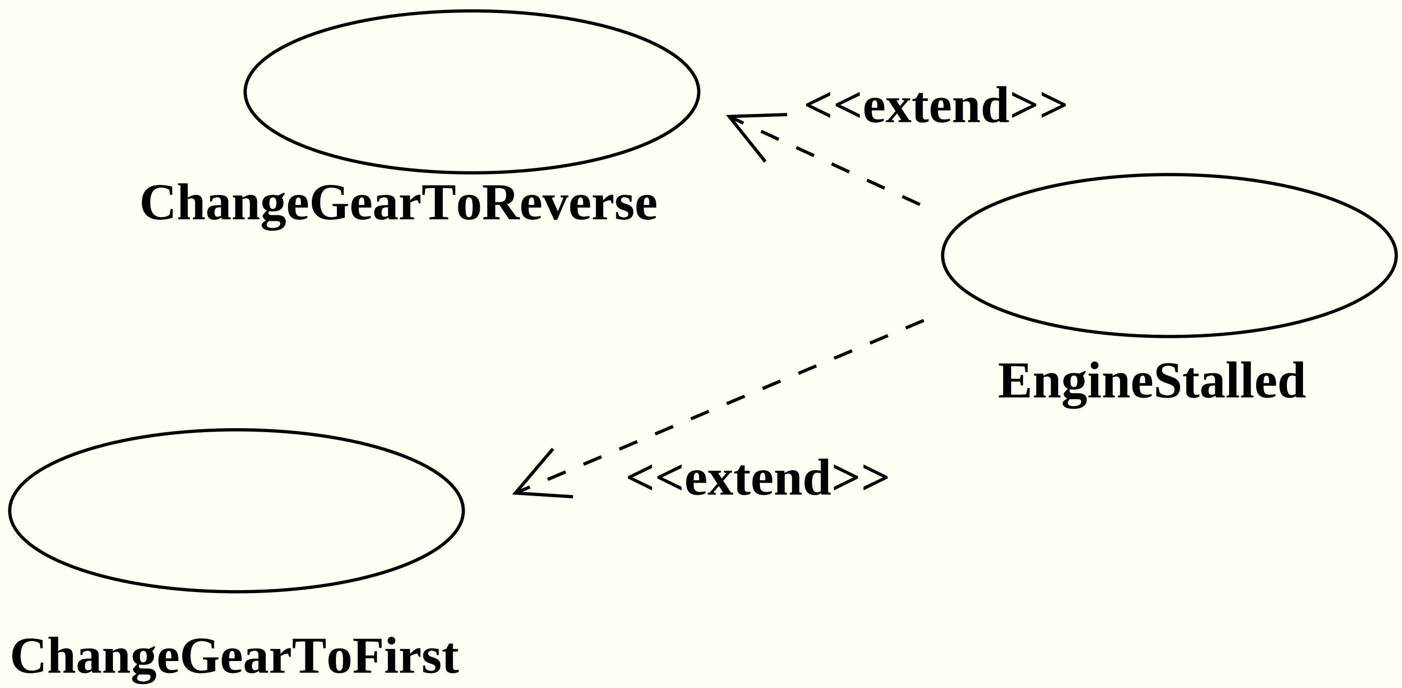
Use Case Diagrams: Generalization



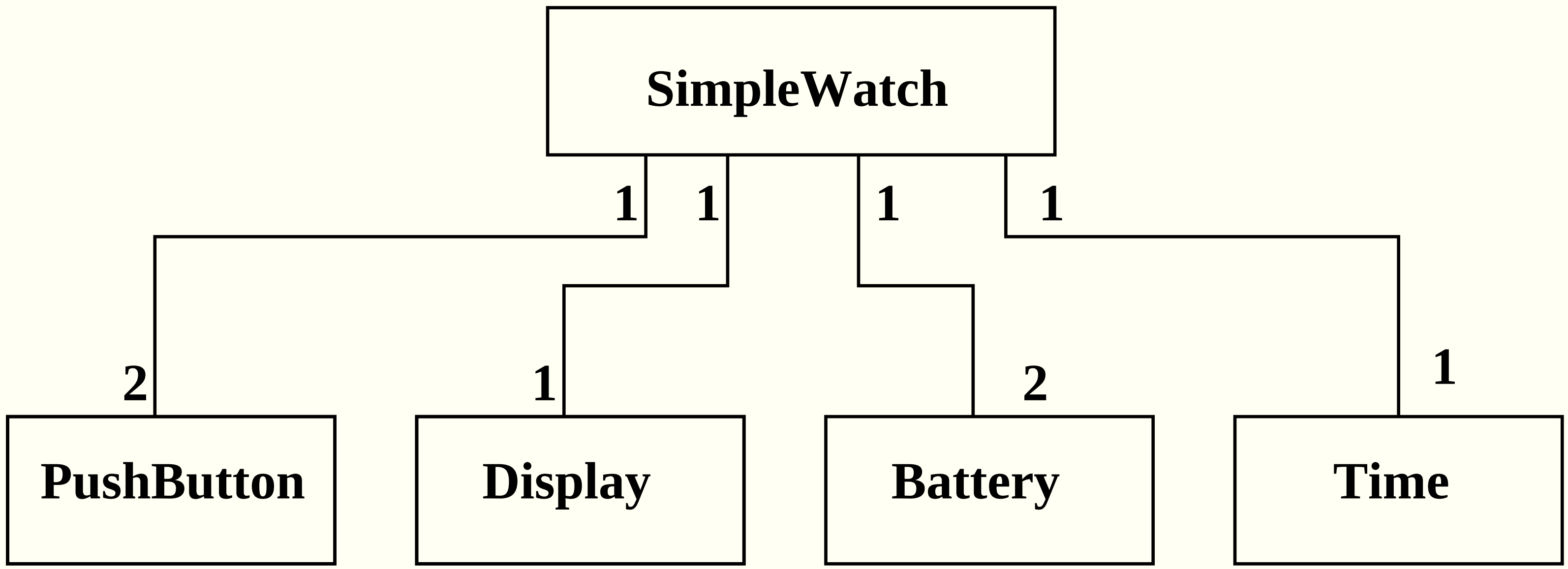
Use Case Diagrams: Include



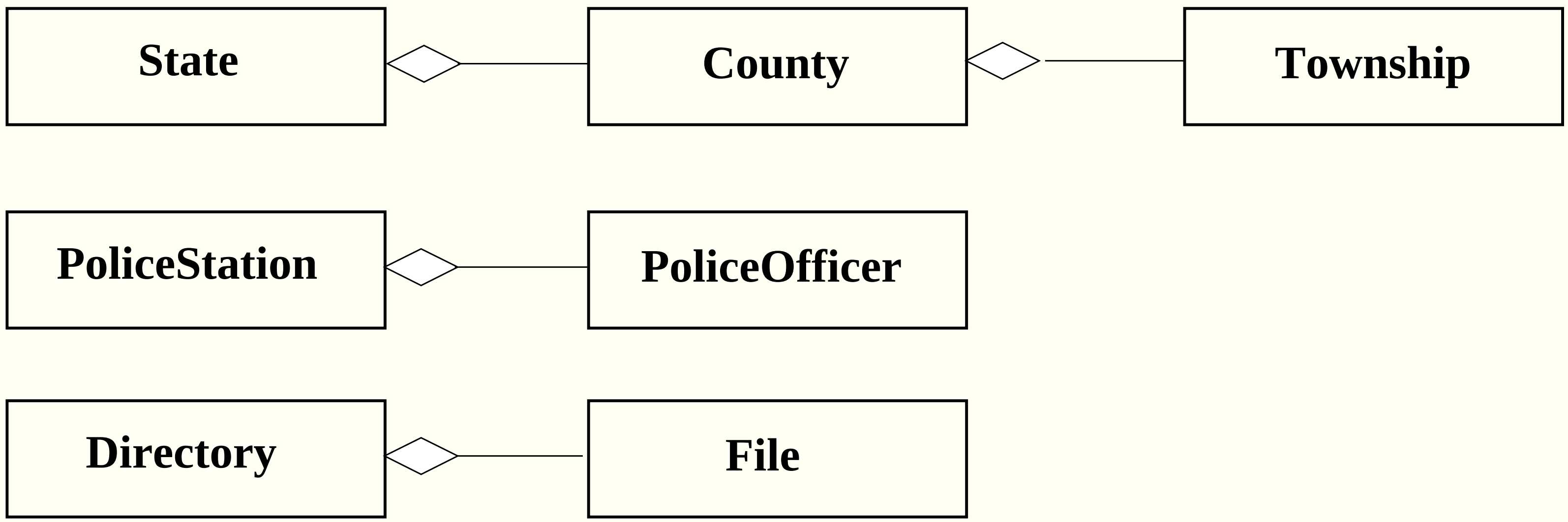
Use Case Diagrams: Extend



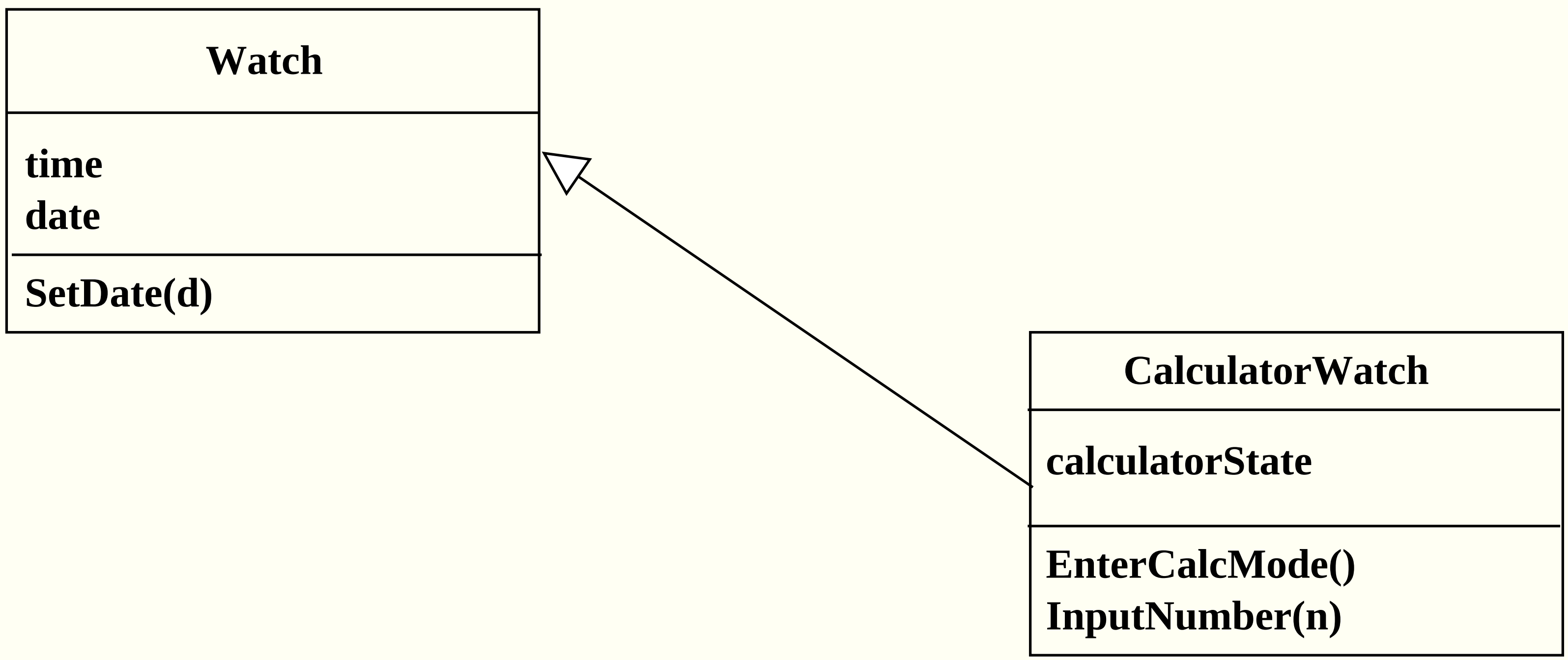
Class Diagrams



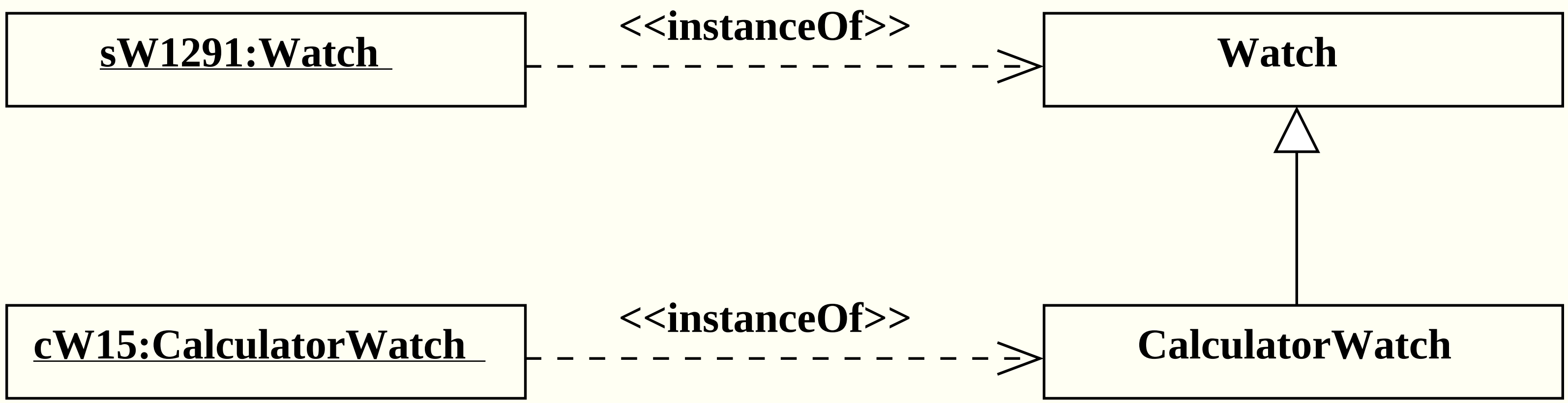
Class Diagrams: Aggregation



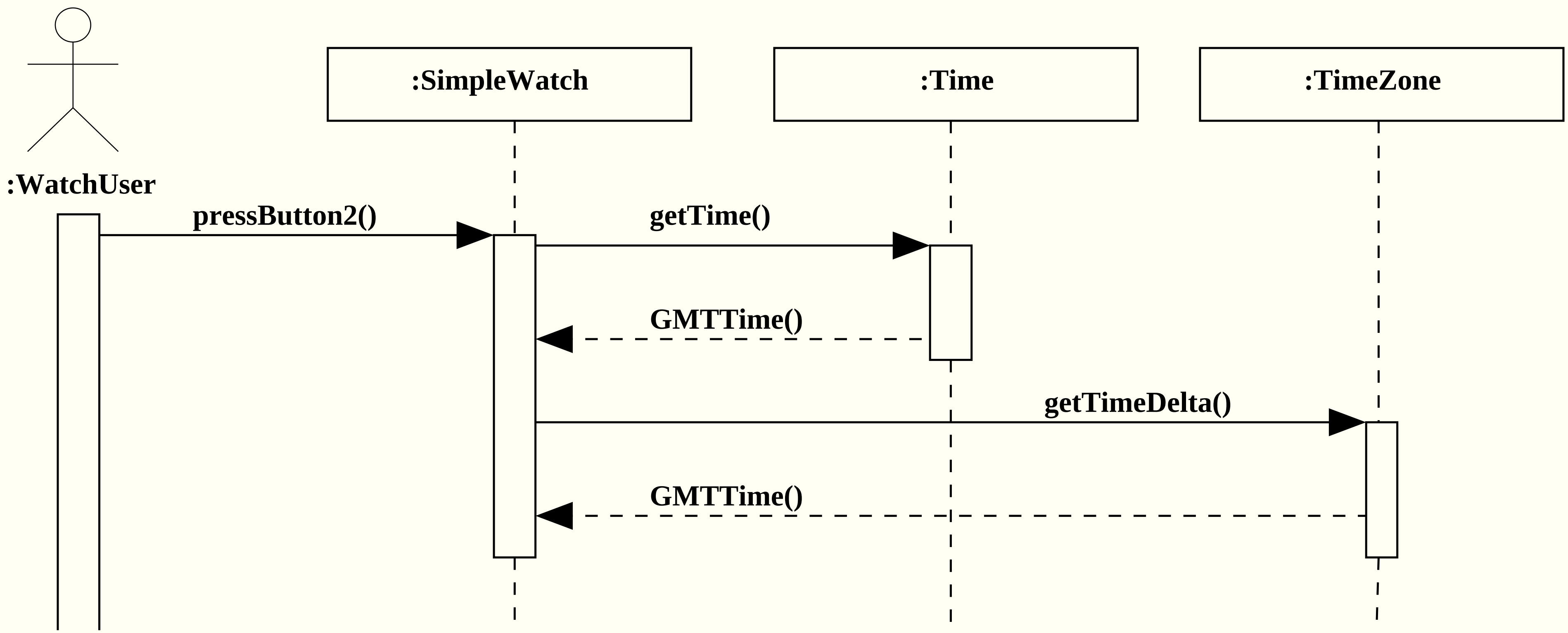
Class Diagrams: Inheritance



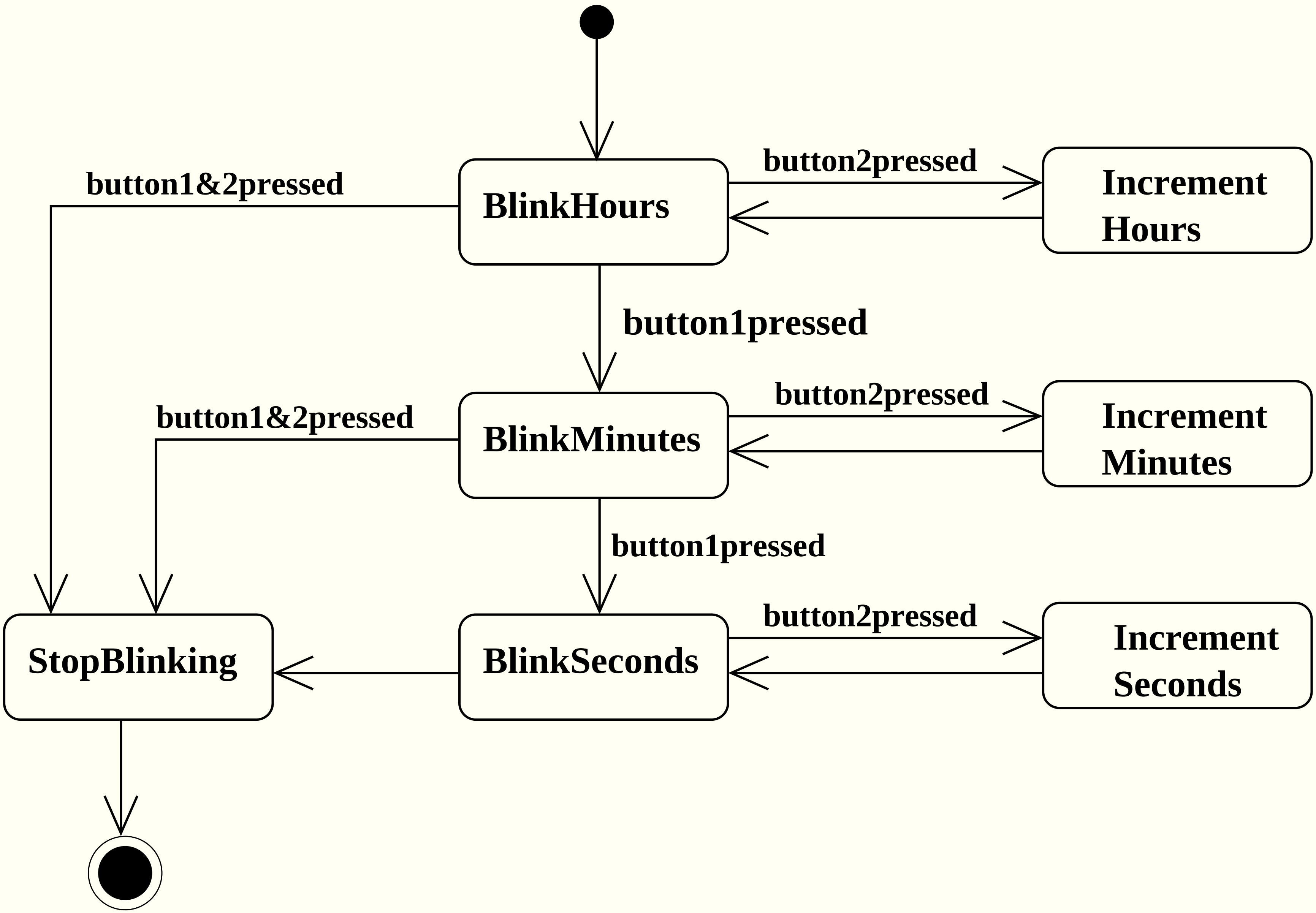
Class Diagrams: Instances



Sequence Diagrams

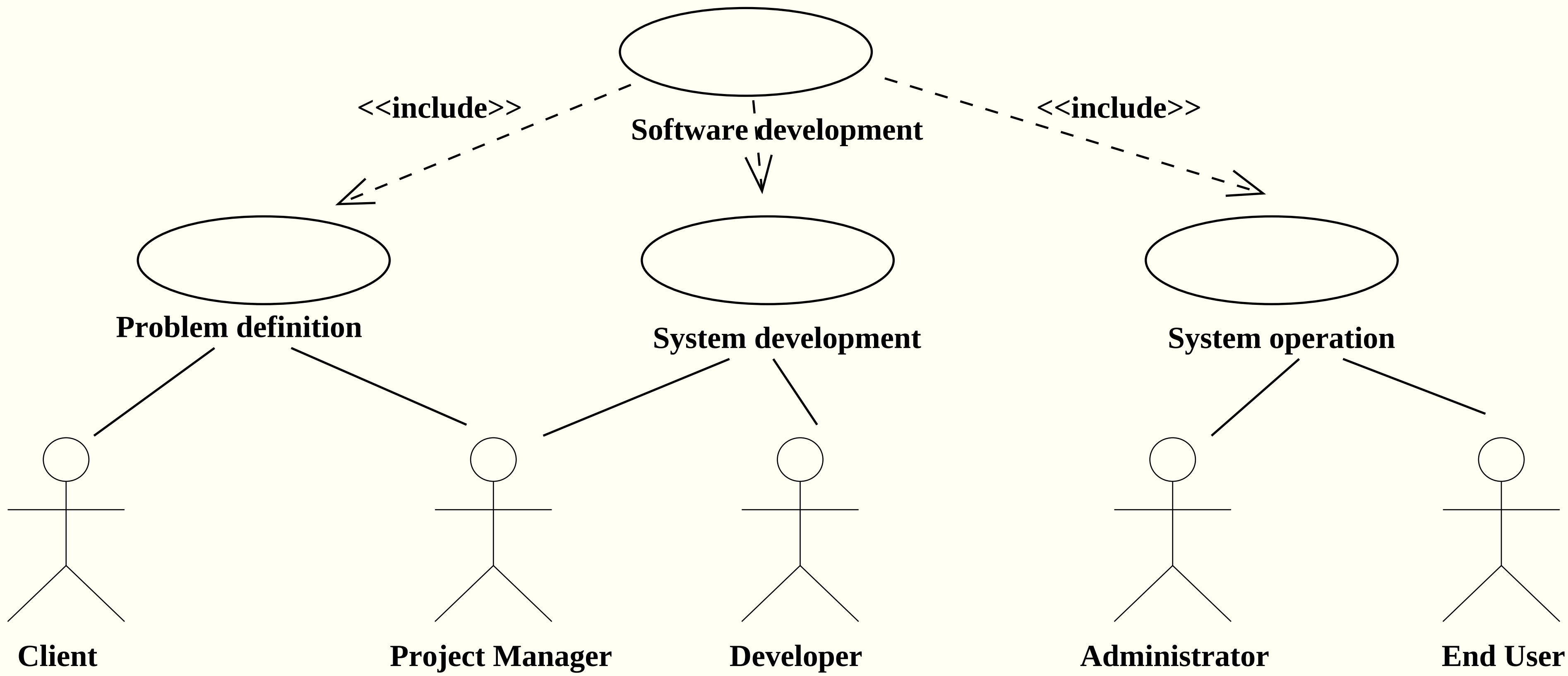


State Chart Diagrams



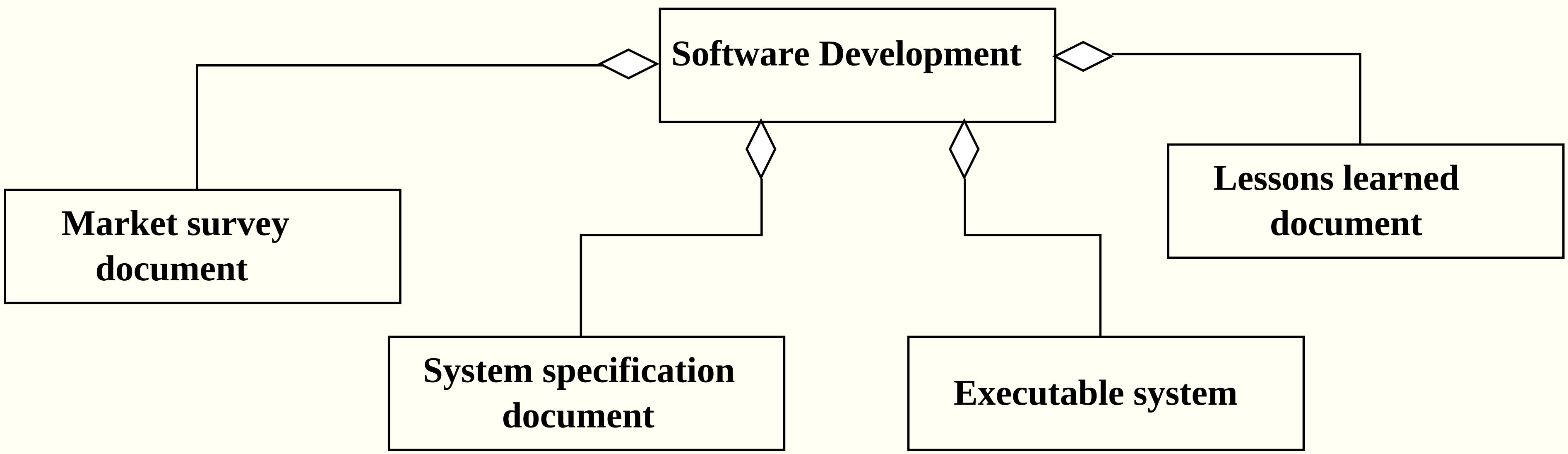
Software Life Cycle

Software Development: Simple View

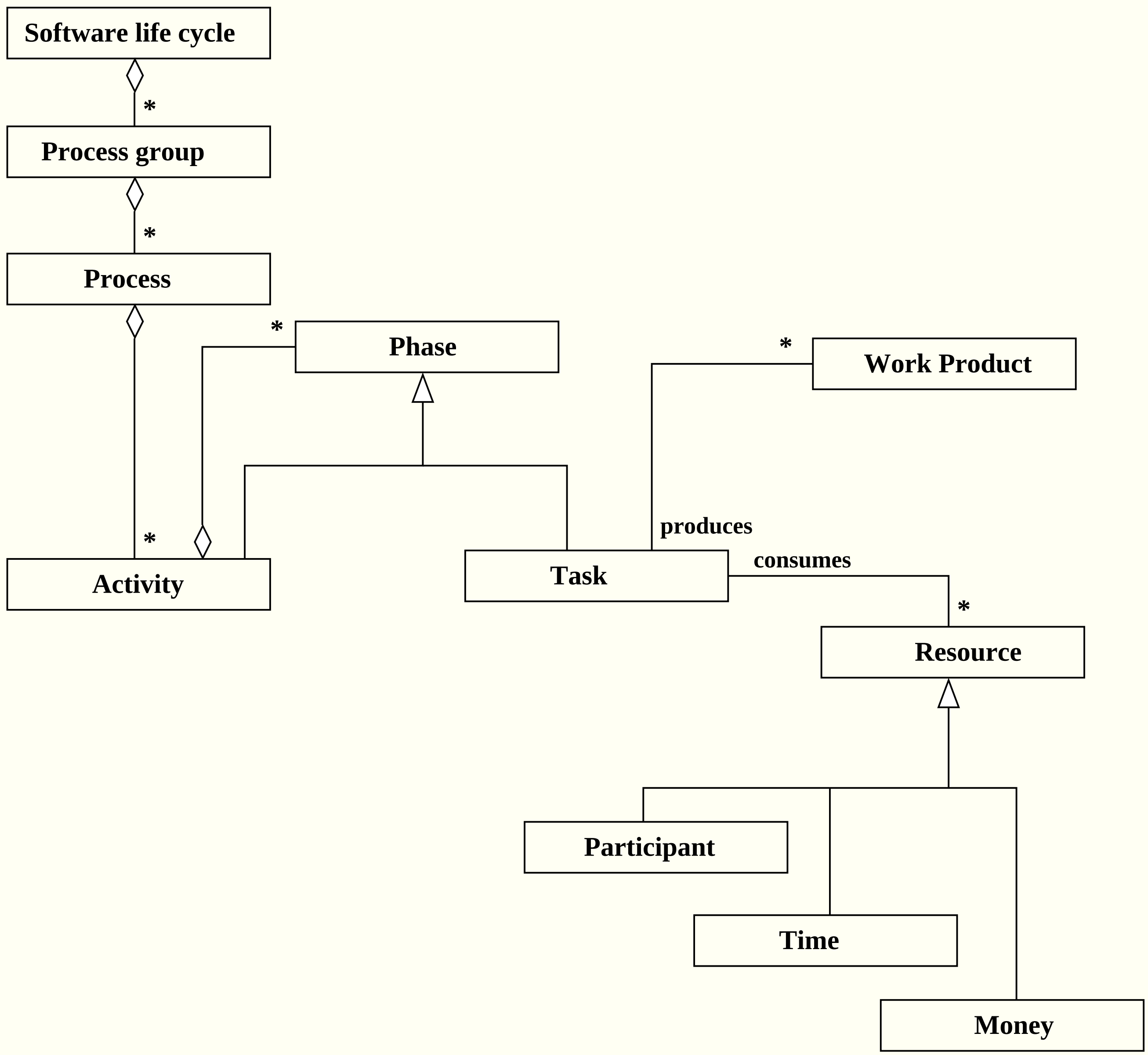


Simple Activity Centric View

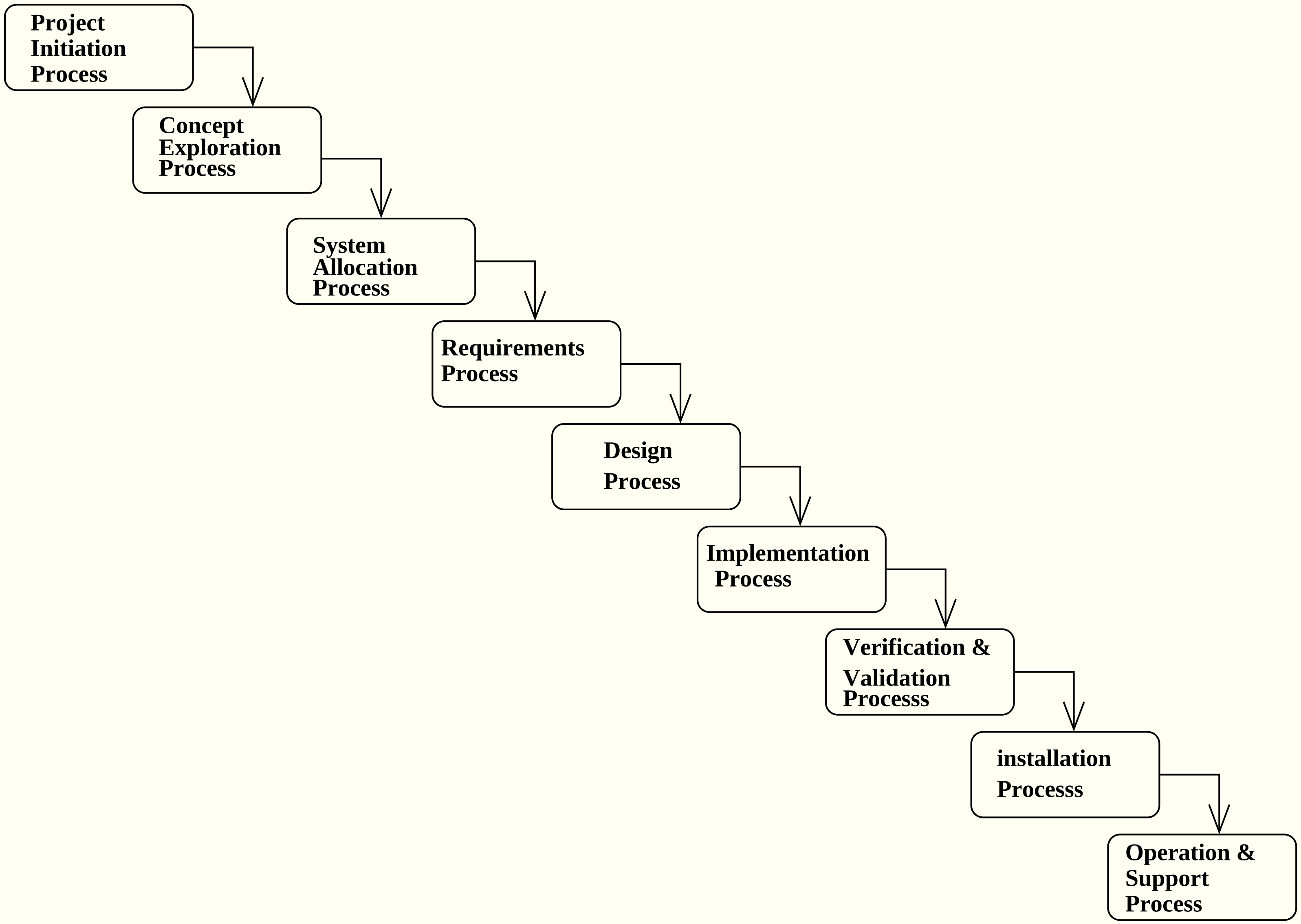
Simple Entity Centric View



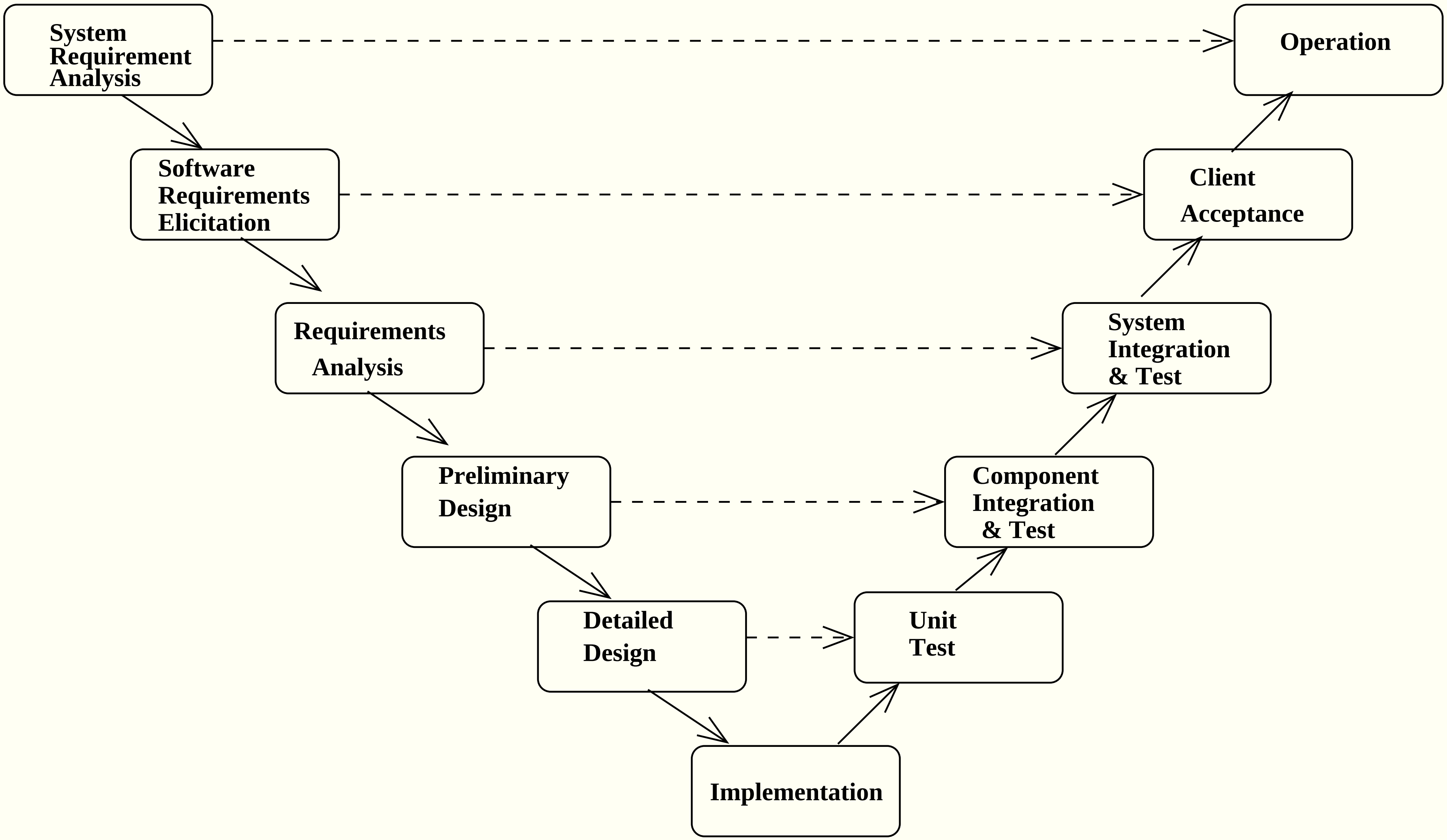
Software Life Cycle: IEEE 1074



Waterfall Model



V-Model



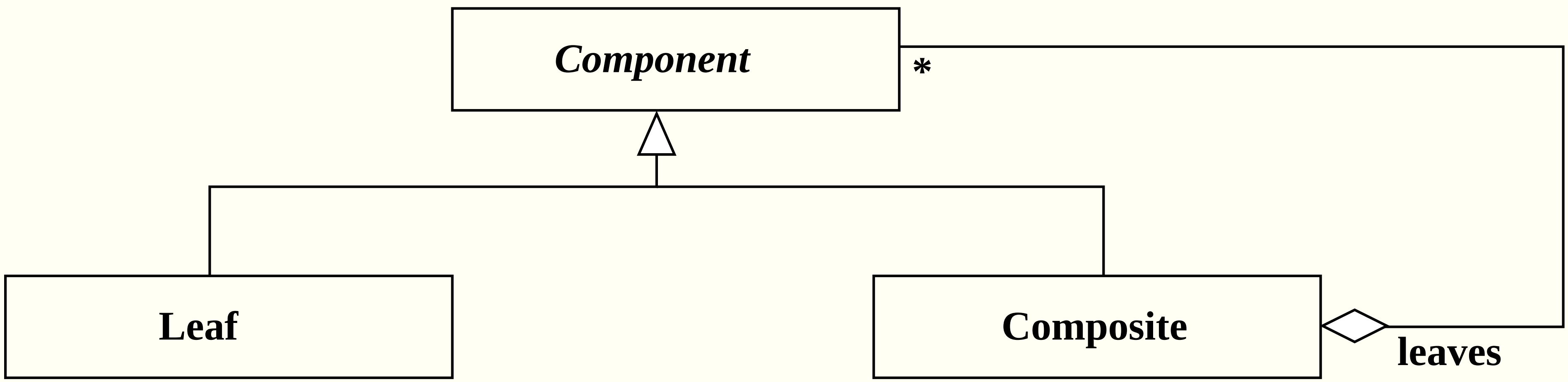
System Requirement Analysis
Operation
Software Requirements Elicitation
Client Acceptance
Requirements Analysis
System Integration & Test
Preliminary Design
Component Integration & Test
Detailed Design
Unit Test
Implementation

Other Models

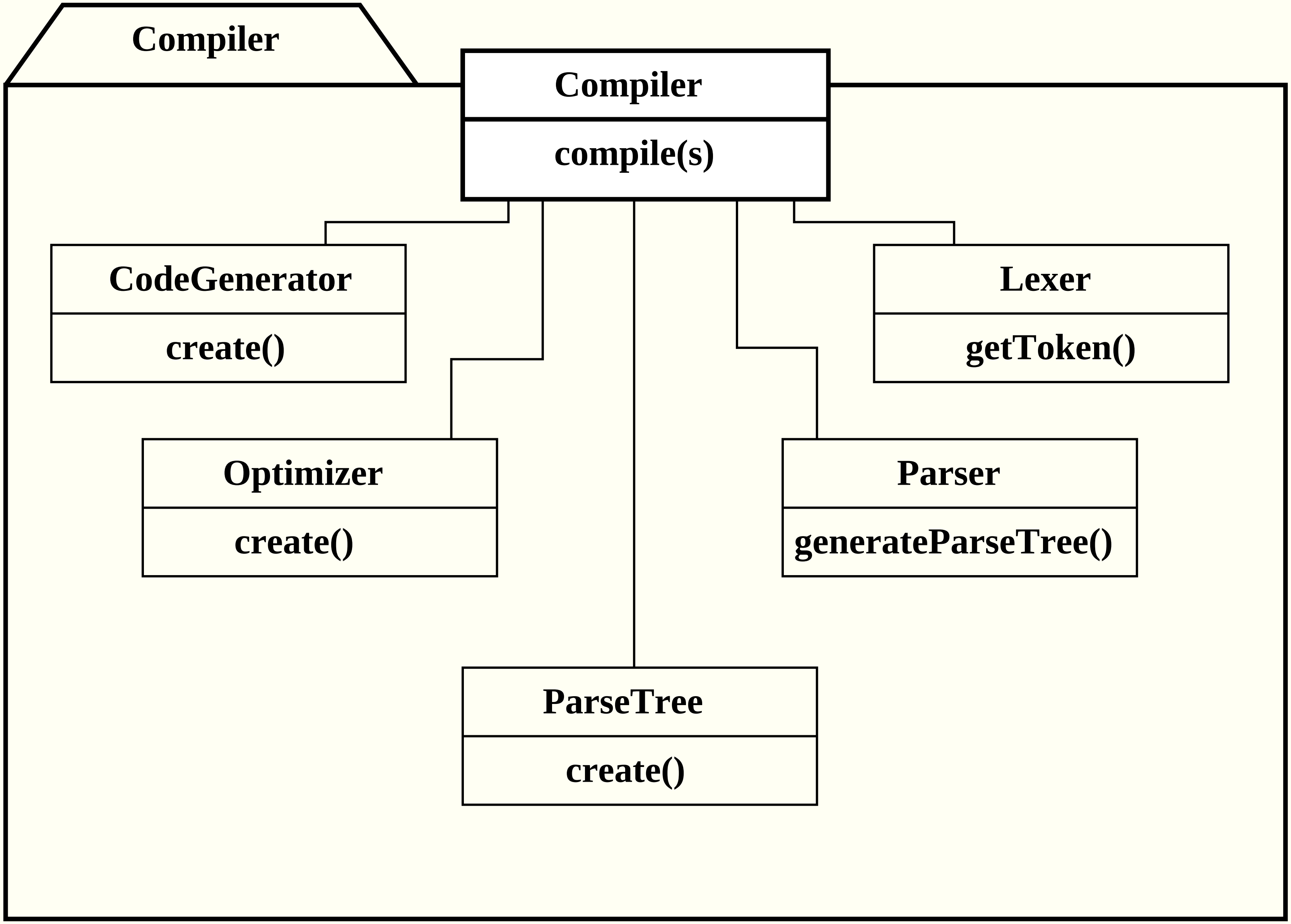
- Boehm's Spiral Model
- Sawtooth Model
- Shark Tooth Model

Design Patterns in UML

Design Patterns



Design Patterns



Extreme Programming

Extreme Programming

- relatively new idea—barely 6 years old
- suited to small projects, of 2-30 people
- relaxes the heavyweight management in favor of informality
- software development process divided into four types of activities
  - planning
  - designing
  - coding
  - testing

XP: Coding (simplified)

XP: Coding (simplified)

- customer is always available

XP: Coding (simplified)

- customer is always available
- code formatted to standards
  - design patterns
  - best practices

XP: Coding (simplified)

- customer is always available
- code formatted to standards
  - design patterns
  - best practices
- unit test coded first
  - unit tests also serve as documentation

XP: Coding (simplified)

- customer is always available
- code formatted to standards
  - design patterns
  - best practices
- unit test coded first
  - unit tests also serve as documentation
- pair programming
  - counter-intuitive, but works!

XP: Coding (simplified)

- customer is always available
- code formatted to standards
  - design patterns
  - best practices
- unit test coded first
  - unit tests also serve as documentation
- pair programming
  - counter-intuitive, but works!
- collective code ownership

XP: Coding (simplified)

- customer is always available
- code formatted to standards
  - design patterns
  - best practices
- unit test coded first
  - unit tests also serve as documentation
- pair programming
  - counter-intuitive, but works!
- collective code ownership
- optimize last

Tools

Modern Tools

Modern Tools

- project tools

Modern Tools

- project tools
- CASE tools

Modern Tools

- project tools
- CASE tools
- project management tools
  - PERT charts (Program Evaluation Review Technique), also called PERT / CRM (Critical Path Management)
  - Gantt charts
  - project management software

Modern Tools

- project tools
- CASE tools
- project management tools
  - PERT charts (Program Evaluation Review Technique), also called PERT / CRM (Critical Path Management)
  - Gantt charts
  - project management software
- high-level languages

Modern Tools

- project tools
- CASE tools
- project management tools
  - PERT charts (Program Evaluation Review Technique), also called PERT / CRM (Critical Path Management)
  - Gantt charts
  - project management software
- high-level languages
- program development environments

Next Lecture: Back to Graphs