Methodologies

Introduction into Software Engineering
Lecture 21

Bernd Bruegge
Applied Software Engineering
Technische Universitaet Muenchen

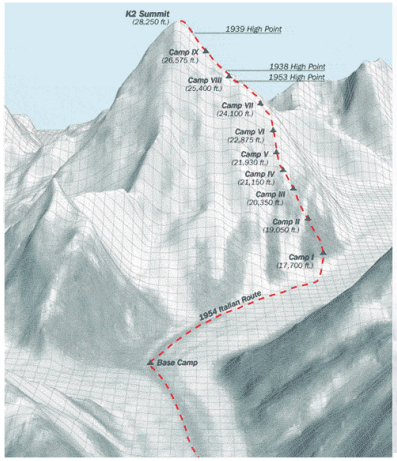
Outline

- A mountaineering example
- Project context
  - Goals, client types
  - Environment, methods, tools, methodology
- Methodology spectrum
  - Planning, design reuse, modeling, process, control&monitoring, redefinition
- Different types of planning
- Different ways to use models
- Use of processes in software development

Key Decisions in an Expedition

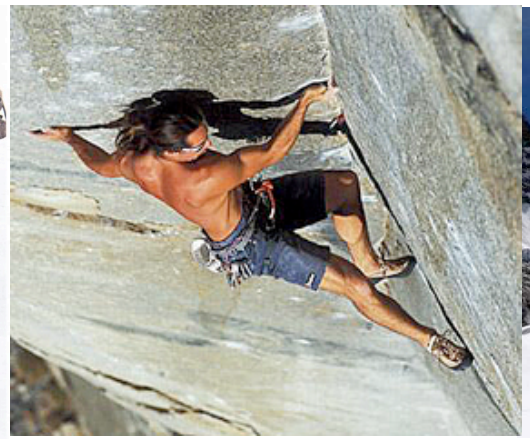
- A leader must answer several key questions to create a successful expedition
  - What mountain should be climbed?
  - What types of tools should be used?
  - Who should be member of the team?
  - Does the expedition need a leader?
- Different answers to these questions lead to different styles:

Siege style



Fixed-rope

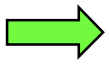
Free Solo



Alpine style

Key Decisions in a Software Project

- Project goals
- Schedule
- Cost
- Project organization
- Software life cycle model
- Tools
- Methods
- Team members and organization

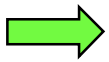


Influenced by Methodology

Methodology

Definition: Software engineering methodology

- Collection of methods and tools for developing and managing a software system to achieve a specific goal in a given project environment



Project environment

- Defined by the client and current state of the development organization. Constrains the project manager (Example: Hierarchical or project-based organization)

Methods

- Techniques to choose from in a given project environment (Example:Object-Oriented Analysis, waterfall model)

Tools

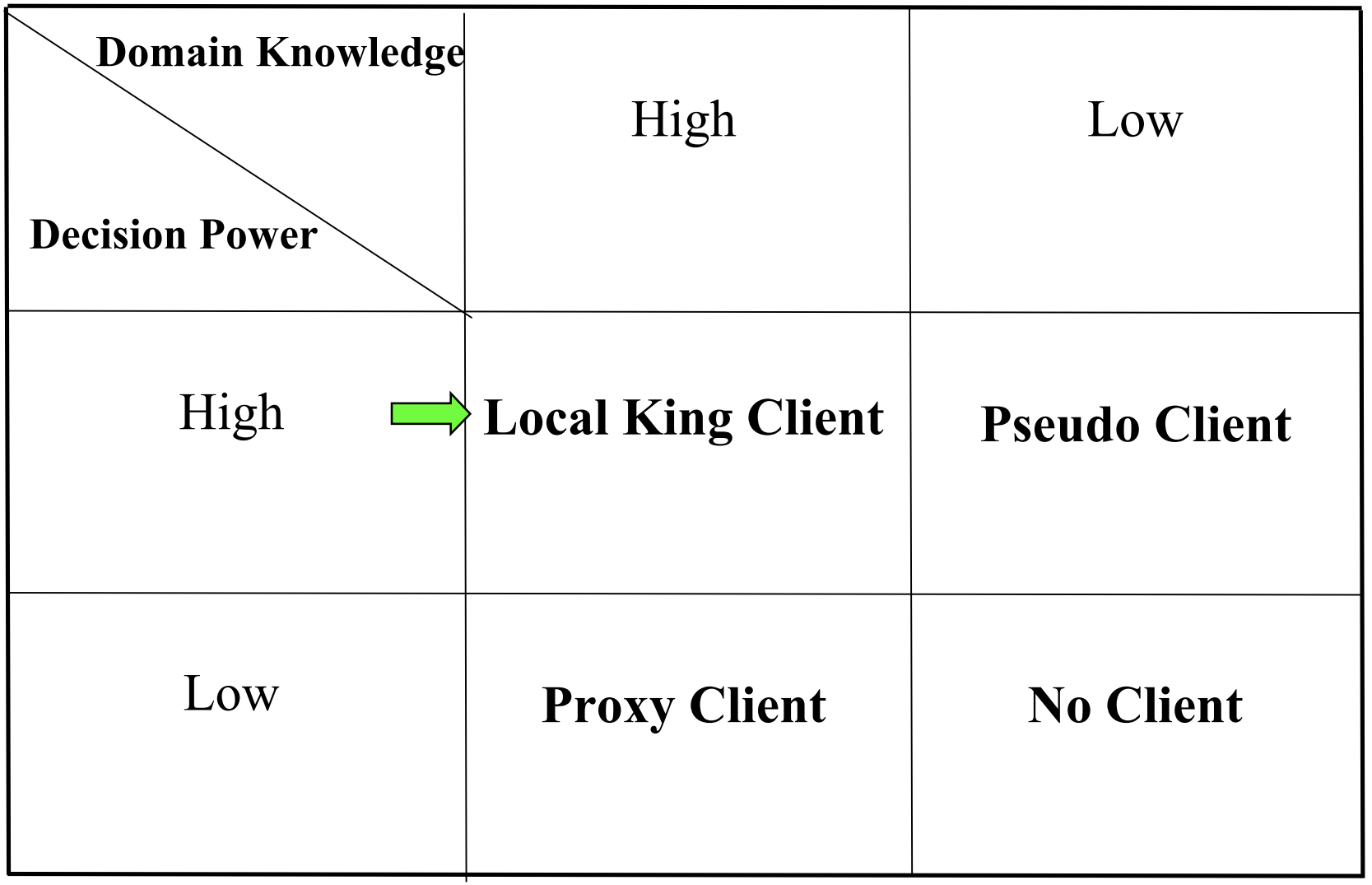
- Devices or programs that support the development and management activities (Example: CASE Tool, IDE)

A methodology specifies for a specific project environment
1) when methods or tools should be used and when not
2) *what to do when unexpected events occur*.

Project Environment

- Participants' expertise
  - Beginner, expert, slow learner, fast learner
- Type of Client
  - Domain knowledge, decision power
- End user access
  - No end user available, end user participates in requirements elicitation, end user participates in usability tests
- Technological climate ("technology enablers")
- Geographical distribution
- Project duration
- Rate of change

Client Type


Domain Knowledge
Decision Power
High
Low
High
Local King Client
Pseudo Client
Low
Proxy Client
No Client

Local King Client

High Domain Knowledge, High Decision Power

- A client who can answer developer questions and make decisions without having to ask anybody else
- Has deep knowledge of the application domain (and/or the solution domain)
- Usually collocated with the project
- Does not have report to anybody else
  - Can effectively collaborate with the project manager and often even with the the developers.

Proxy Client

High Domain Knowledge, Low Decision Power

- Proxy clients are sent for the "real client"

  Reasons:

  - Real client has no time
  - Physical distance would make collaboration of the real client with the project organization difficult
- Proxy clients have sufficient knowledge of the application domain
  - They can answer clarification questions from the developers
- Proxy clients do not have sufficient power
  - They cannot make major decisions, they have to ask somebody else => time delay!

Pseudo Client

Low Domain Knowledge, High Decision Power

- The pseudo client is a member of the development organization
  - Often even developers act as pseudo clients
  - If the system is targeted at a new market segment, the pseudo client often comes from marketing
- Pseudo clients can make decisions within a short time
- Pseudo clients have a limited knowledge of the application domain.

"No Client"

- A project can start without a client
  - Example: A visionary product is developed before a market segment is opened

- In these cases the project manager should still select a client, usually a pseudo client who acts as an end user
  - The stakes of the developers can be balanced against the stakes of the future user.

End User Access

- Clients and end users usually do not have the same interests
- Clients are interested in
  - an early delivery date
  - as much functionality as possible
  - low cost
- End users are interested in
  - a familiar user interface
  - an easy to learn user interface
  - a system that supports their specific task well
- If the project success depends on the usability of the product, then
  - end users should be included in the project
  - usability tests should be conducted with the end users.

Project Environment

- Participants' expertise
  - Beginner, expert, slow learner, fast learner
- Type of Client
  - Domain knowledge, decision power
- End user access
  - No end user available, end user participates in requirements elicitation, end user participates in usability tests
- Technological climate ("technology enablers")
- Geographical distribution
- Project duration
- Rate of change

Technological climate

- Depending on the requirements expressed by the client, a project may be constrained in the technological components it has to use. Examples:
  - A project needs to improve a legacy system
    - It deals with well-known and mature technology but the technology might be out of date
  - A project develops a first-of-a-kind prototype
    - based on a new technology enabler
      - Examples: RFID, GPS
    - Usually has to deal with preliminary versions of components and immature technology
      - GPS in a mobile phone

Geographical Distribution

- "Single room" projects: Participants in a single room
- Reasons for distributed projects:
  - Organization may have resulted from the merger
  - Organization is a consortium, located in different geographical locations
  - Part of the organization must be collocated with client
- Geographical distribution has advantages and disadvantages:
  - ↑ Promise of low cost labor
  - ↑ Increases the availability of skill
  - ↑ May take advantage of different time zones
  - ↓ Slows down communication and decision making
  - ↓ Lowers awareness among teams
  - ↓ Leads to loss of information between sites
  - ↓ High communication cost.

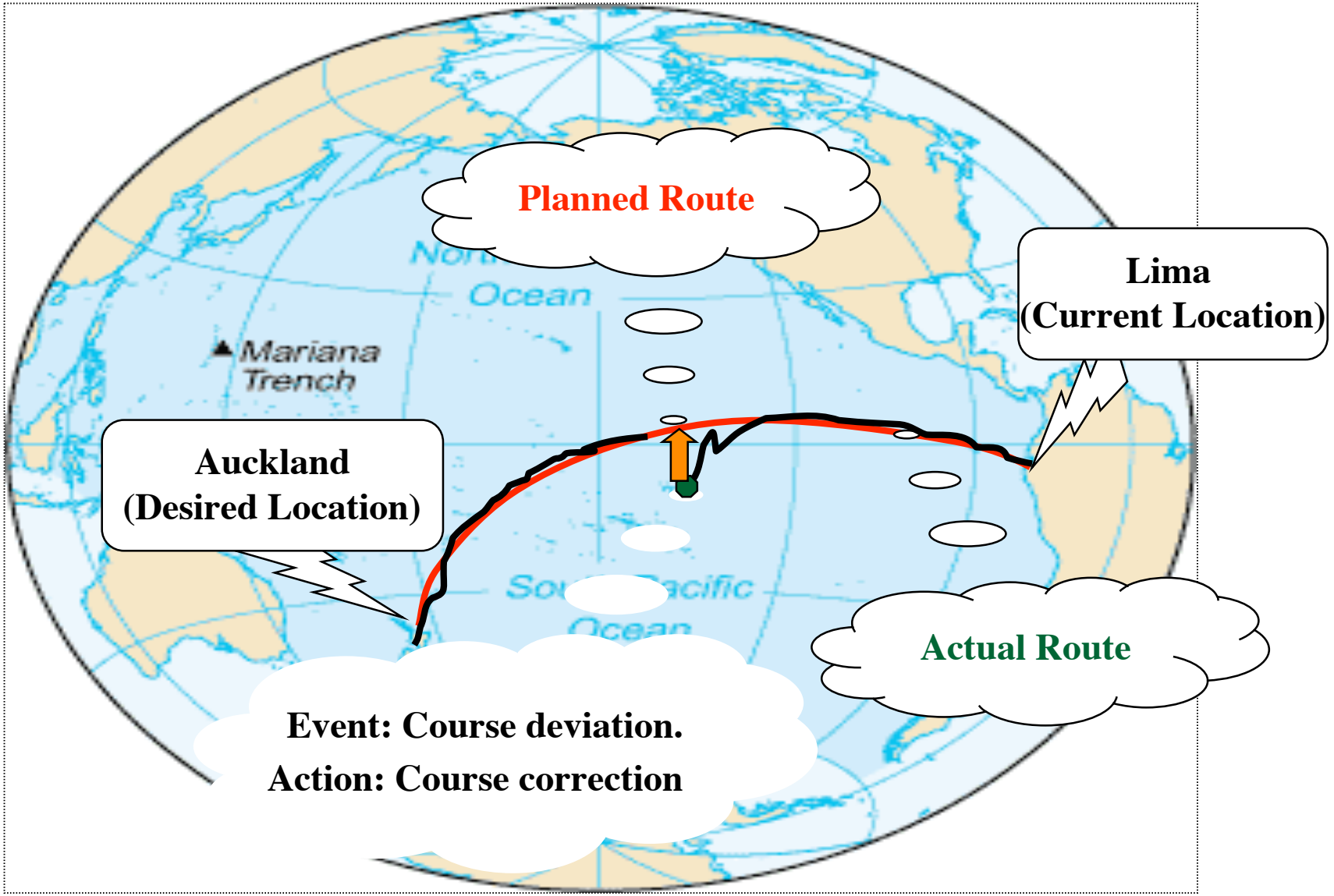
Methodology Issues

- Methodologies provide general principles and strategies for selecting methods and tools in a given project environment
- Key questions for which methodologies provide guidance:
  - How much involvement of the customer?
  - How much planning?
  - How much reuse?
  - How much modeling before coding?
  - How much process?
  - How much control and monitoring?

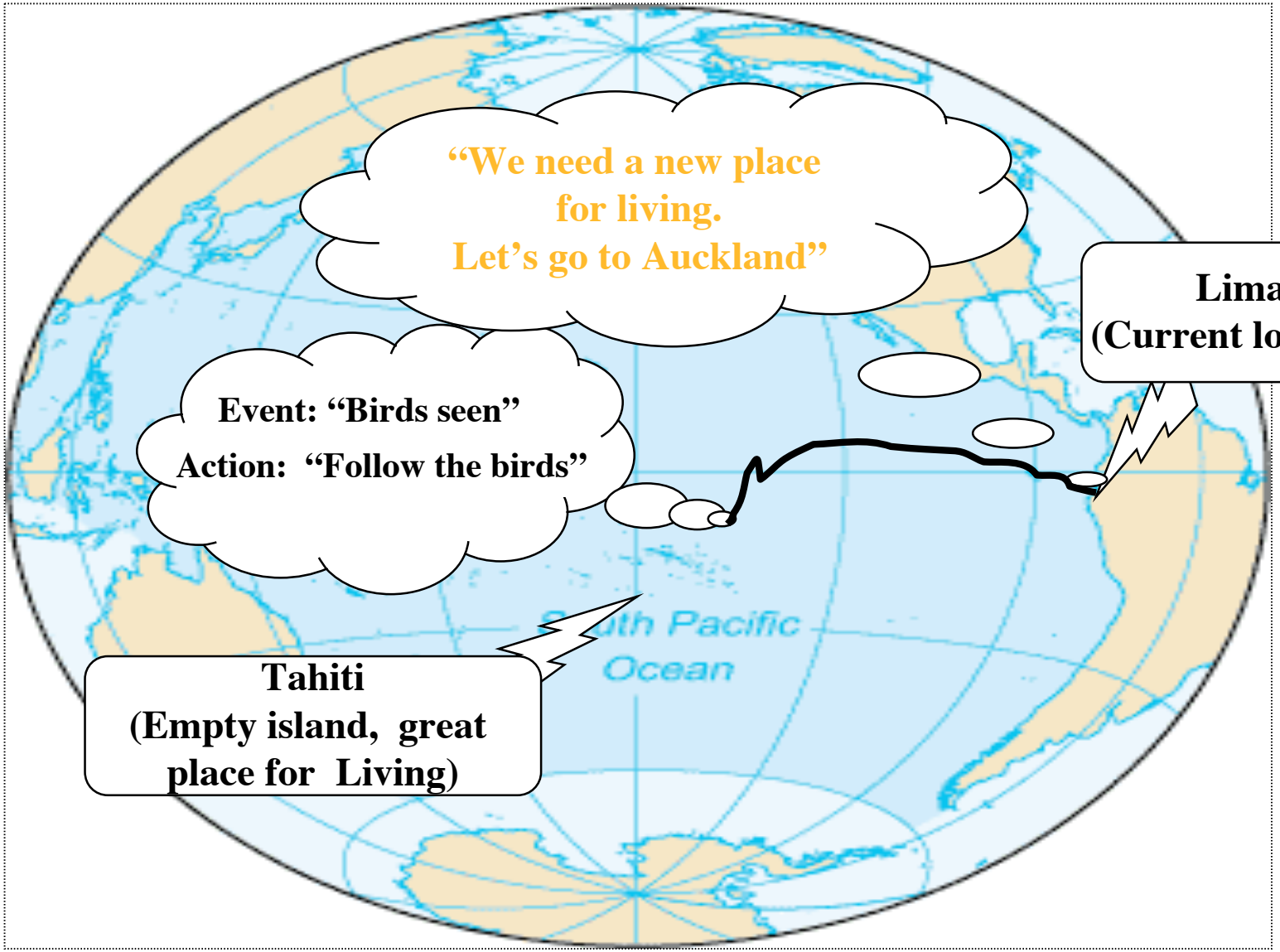
How much Planning?

- Two styles of navigation [Gladwin 1964]
  - European navigation:
    - Current Location and Desired Location
    - Planned Route
    - Route Deviation and Route Correction
  - "Polynesian navigation"

"European Navigation" (Plan-based)



Polynesian Navigation (Situation-based)



Situated action 7 11 2007

- Context-dependent action [Suchman 1990]
  - Selection of action depends on the type of event, the situation and the skill of the developer
- European Navigation is context independent
  - Event: "Course deviation in the morning"
    - Action: "Course correction towards planned route"
  - Event: "Course deviation in the evening"
    - Action: "Course correction towards planned route"
- Polynesian Navigation is context dependent
  - Event: "Birds seen", Context: Morning
    - Action: "Sail opposite to the direction of the birds
  - Event: "Birds seen", Context: Evening
    - Action: "Sail in the direction of the birds".

Outline for Today

- Finishing up this lecture on Methodologies
- Last Lecture: Agile Methods XP and Scrum
- Comments on the Evaluation
- Final:
  - Organizational Issues
  - How to prepare for the Final.

Pros and Cons of Software Project Plans

- Plus
  - Very useful to kick off a software project
  - Useful also if the outcome is predictable or if no major change occurs
- Con:
  - Of limited value to control the project when
    - the outcome is unpredictable
    - when unexpected events occur that change the project environment, tools or methods
- Examples of unexpected events:
  - Appearance of new technology unknown at project start
  - A visionary scenario turns out to be unimplementable
  - Company is merged with another one during the project.

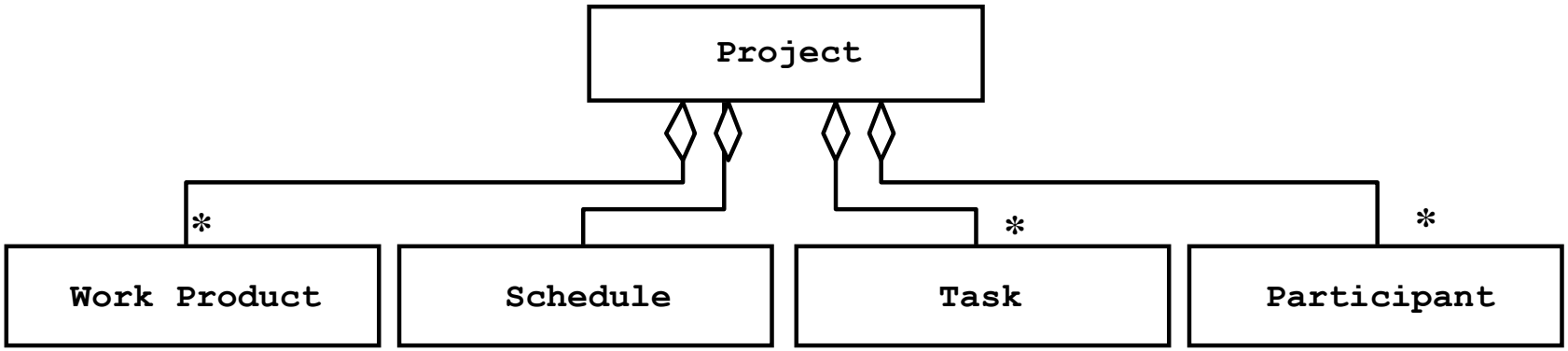
How much Modeling?

- Advantages of modeling:
  - Modeling enables developers to deal with complexity
  - Modeling makes implicit knowledge about the system explicit
  - Modeling formalizes knowledge so that a number of participants can share it
- Problem with modeling:
  - If one is not careful, models can become as complex as the system being modeled.

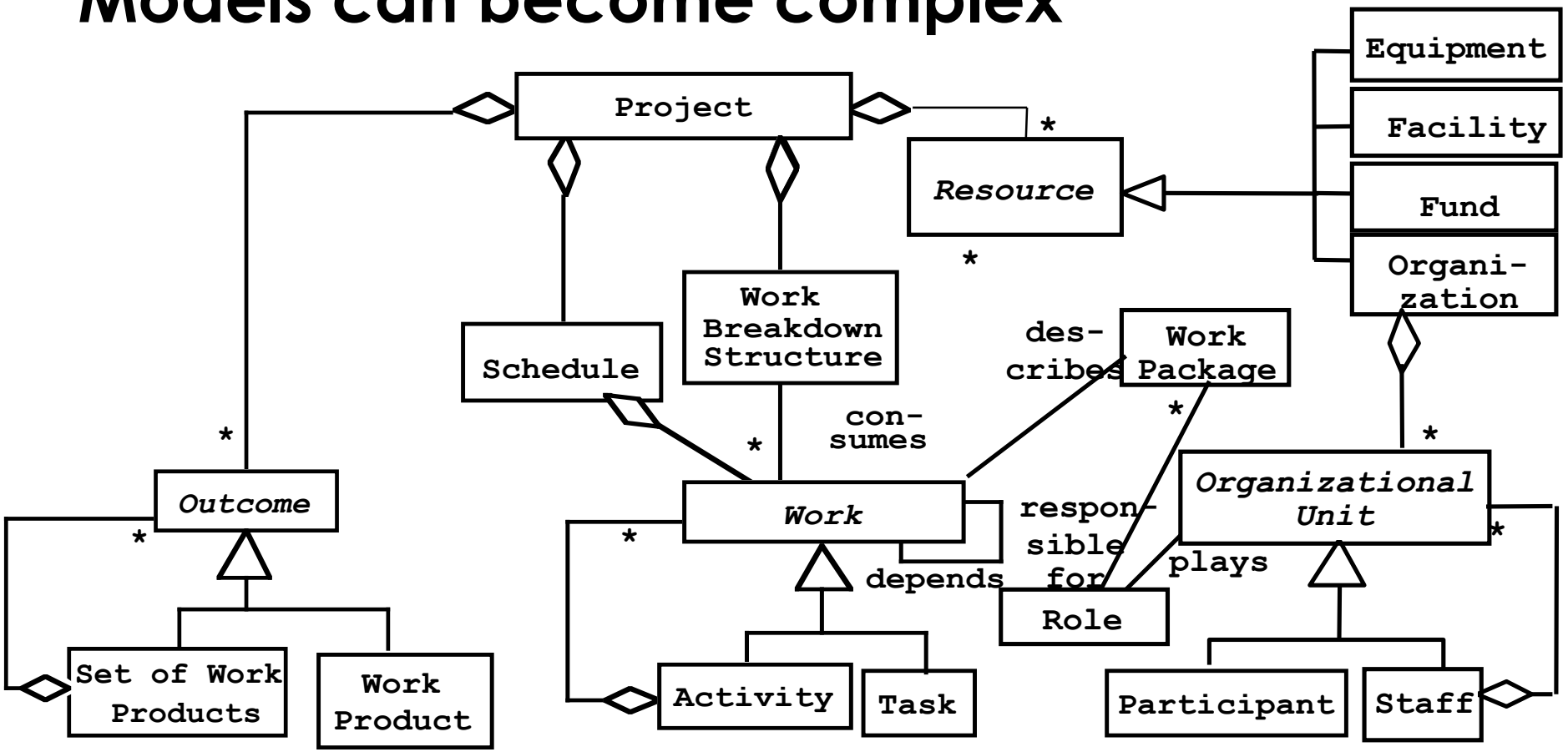
Managerial Challenges of Modeling

- Formalizing knowledge is expensive
  - Takes time and effort from developers
  - Requires validation and consensus
- Models introduce redundancy
  - If the system is changed, the models must be changed
  - If several models depict the same aspects of the system, all of them must be updated
  - If one model becomes out of sync, it loses its value
- Models become complex
  - As the model complexity becomes similar to the complexity of the system, the benefit of having a model is reduced significantly.

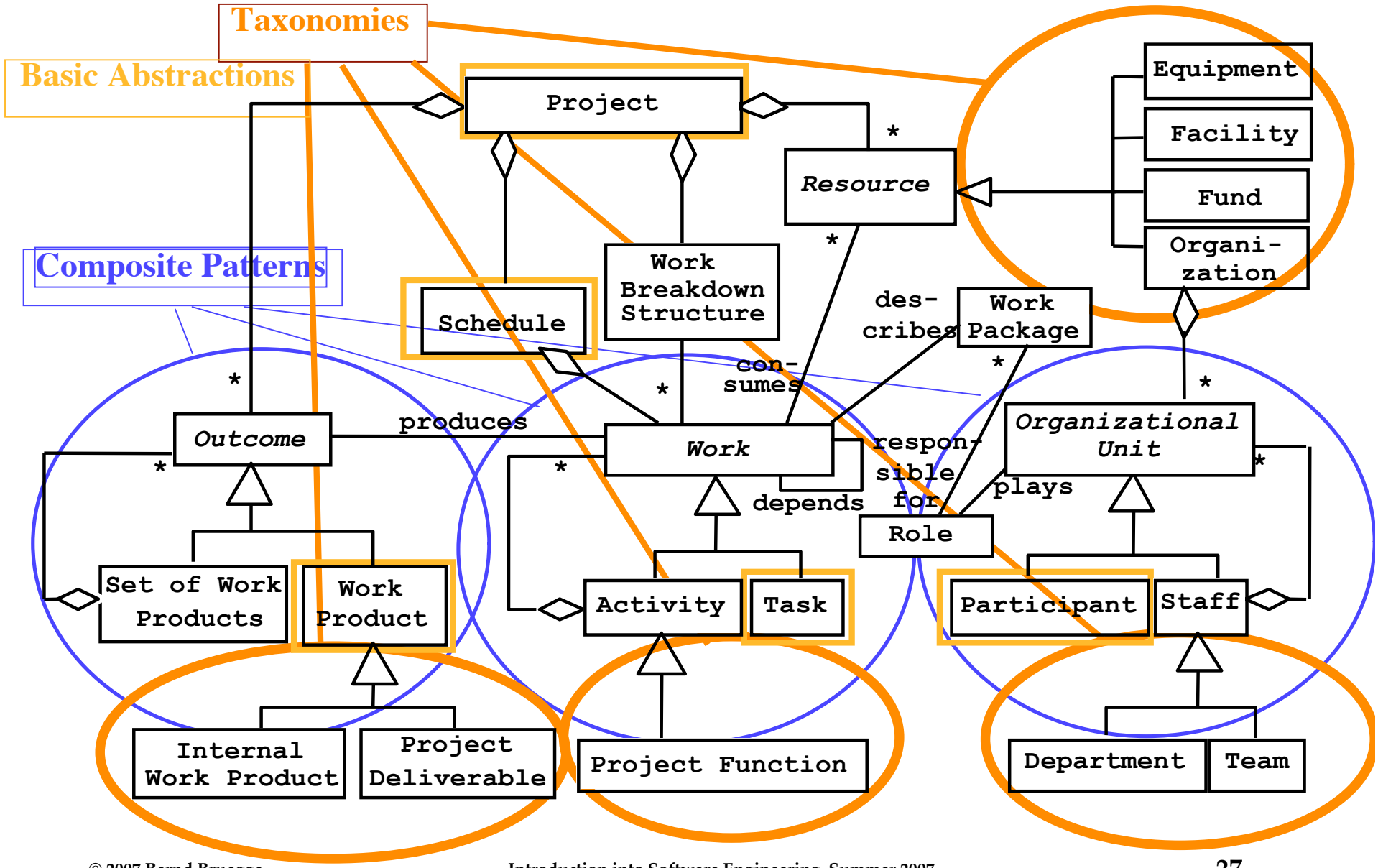
Model of a Software Project



Models can become complex



Use Patterns to Reduce Complexity



Reducing the Complexity of Models

- To reduce the complexity of large model we use navigation and abstraction
- Start with a simplified model and then decorate it incrementally
  - Start with key abstractions (use animation)
  - Then decorate the model with the additional classes
- To reduce the complexity of the model even further
  - Use inheritance (taxonomies, design patterns)
  - If the model is still too complex, show the subclasses on a separate page

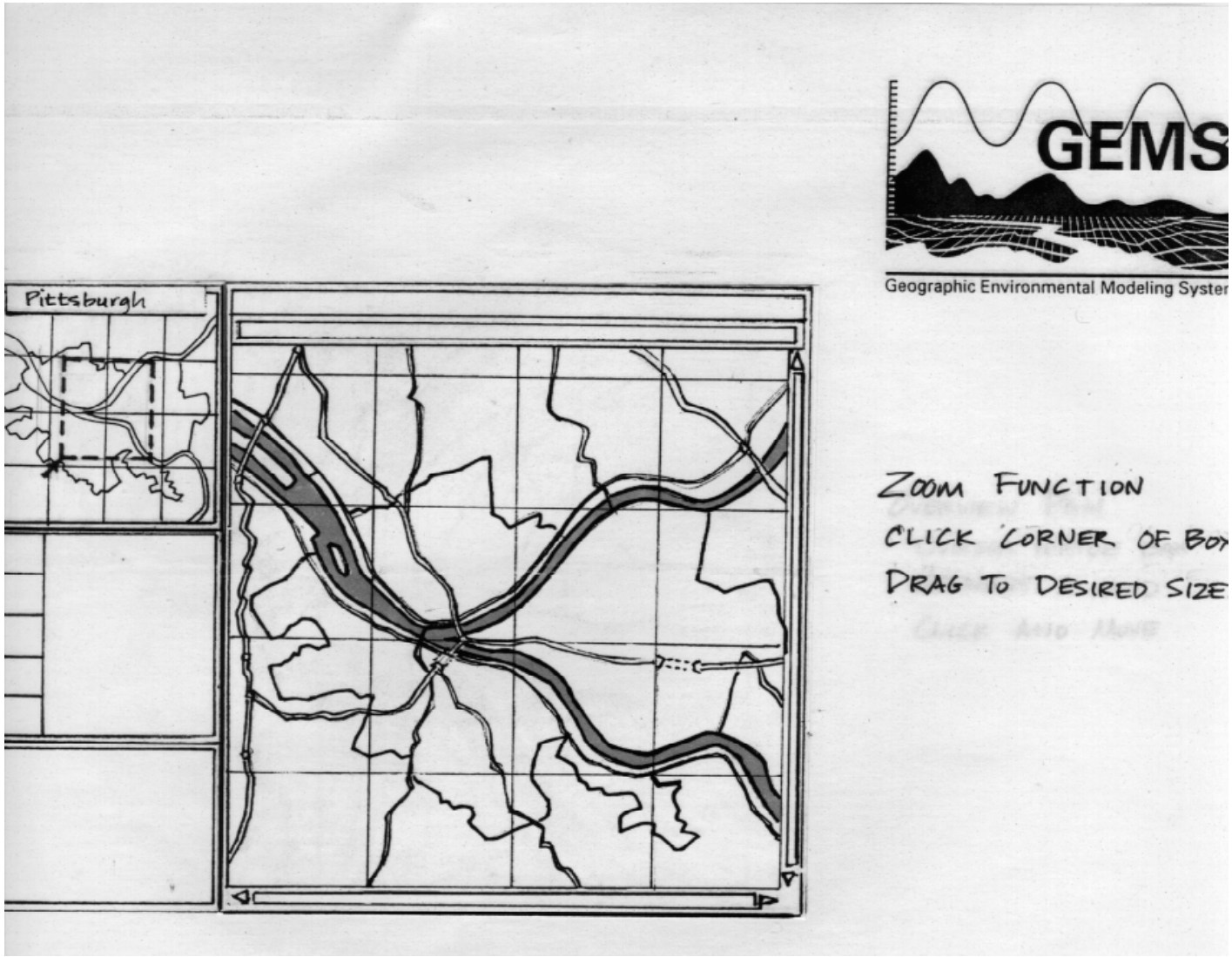
Where do we need Models?

- Models support three different types of activities:
  - Communication: The model provides a common vocabulary. An informal model is often used to communicate an idea
  - Analysis/Design: Models enable developers to reason about the future system
  - Archival: Compact representation for storing the design and rationale of an existing system.

Models to support Communication

- Also called conceptual models
- Most often used in the early phases of a project and during informal communications.
  - The model does not have to be consistent or complete
  - The notation does not even have to be correct
- The model is used only to communicate an idea to ***a person***
  - If the idea is understood, the model has served its purpose
- UML is our preferred notation for models to support communication
- Communication Media:
  - A Whiteboard, a slide or even a napkin.

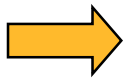
"Napkin Design"



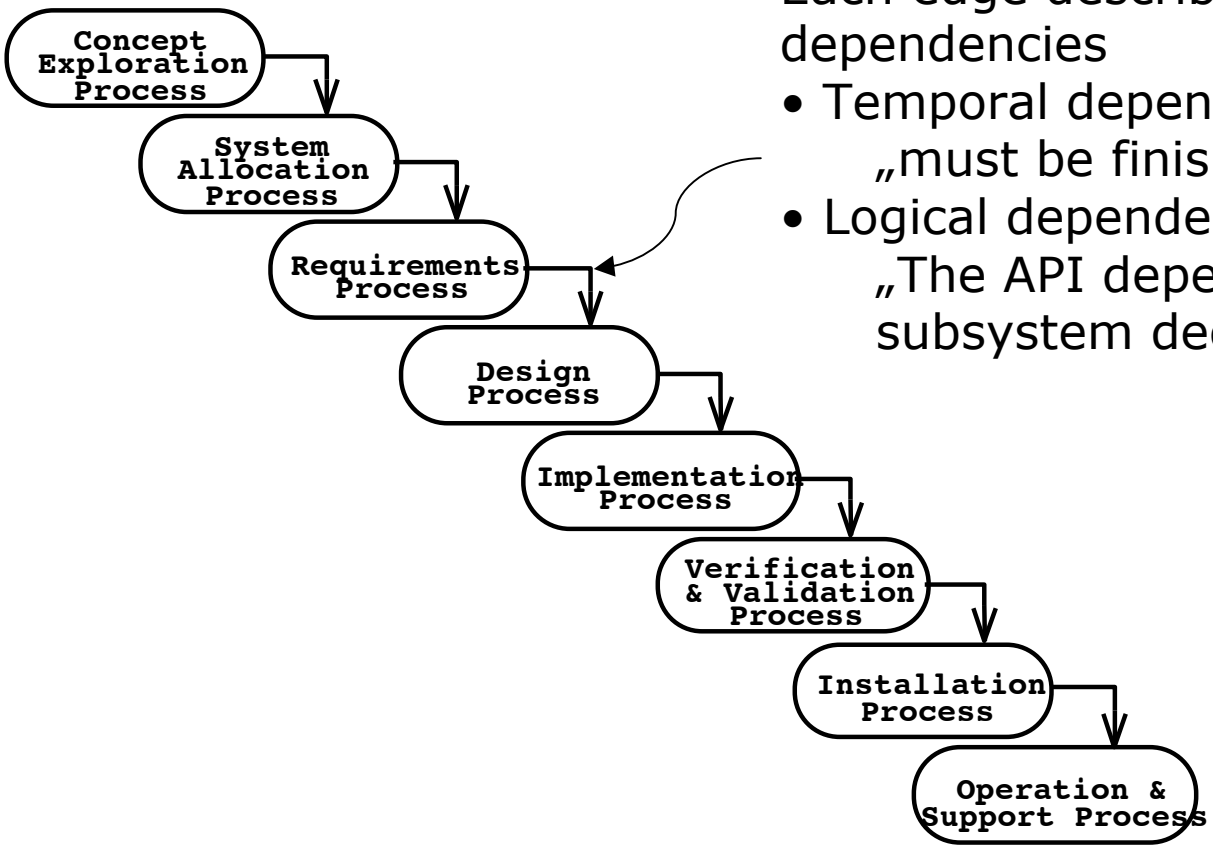
Models to support Analysis and Design

- Also called specification models
- The model provides a representation that enables developers to reason about the system
- The model is used to communicate the idea to ***a computer***
  - The model needs to be made consistent and complete
  - The notation must be correct so the model can be entered into a CASE tool
- UML is our preferred notation for models to models that support analysis and design.

Methodology Issues

- Methodologies provide guidance, general principles and strategies for selecting methods and tools in a given project environment.
- Key questions for which methodologies provide guidance:
  - How much involvement of the customer
  - ✓ How much planning?
  - How much reuse?
  - ✓ How much modeling?
  -  How much process?
  - How much control and monitoring?

Problems with linear Models

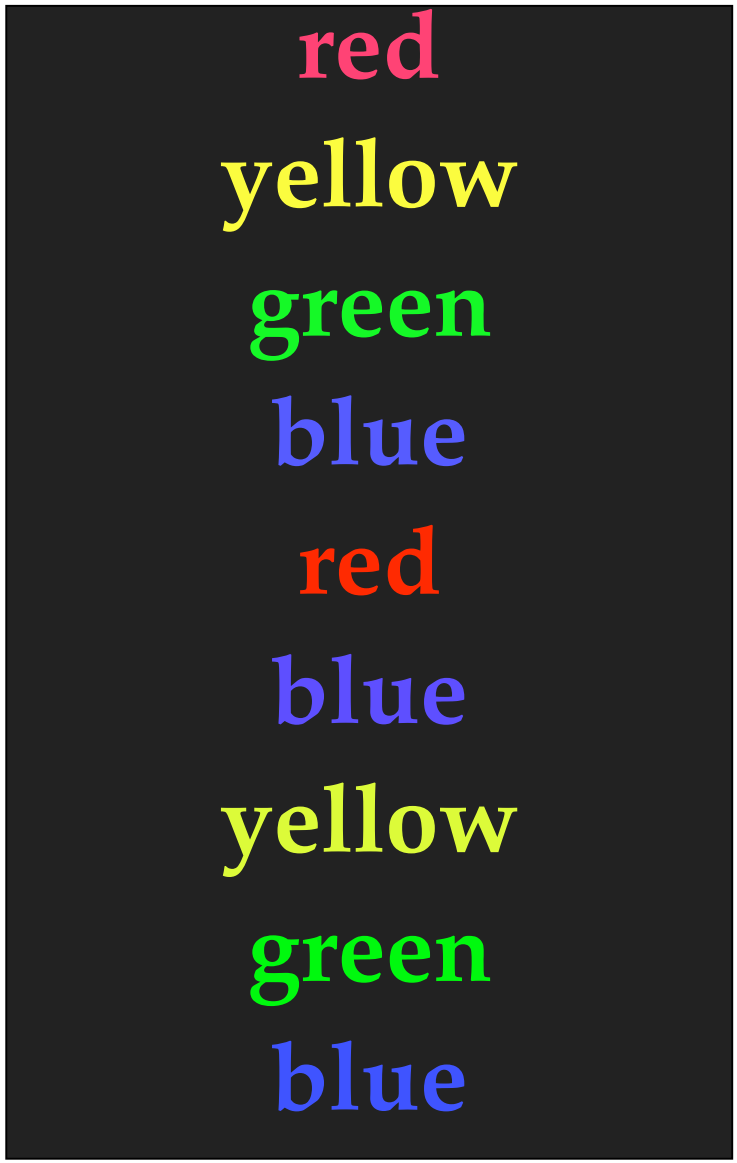


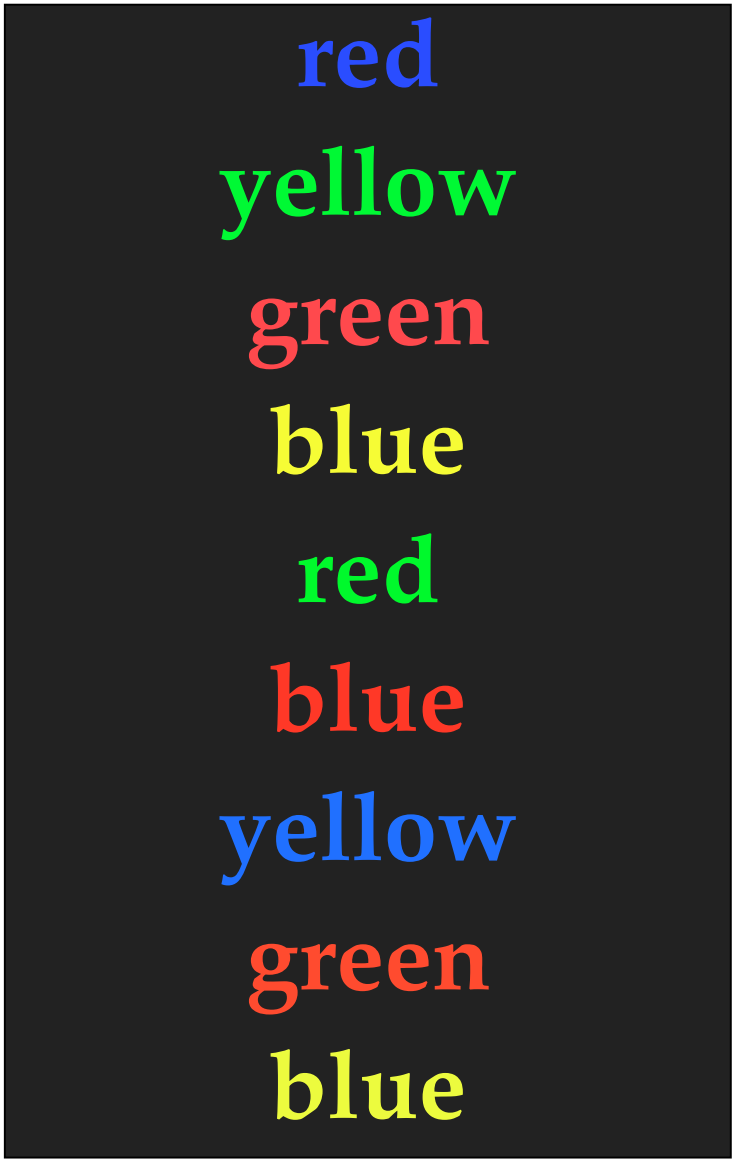
Each edge describes 2 types of dependencies

- Temporal dependency:
  „must be finished before"
- Logical dependency
  „The API depends on the subsystem decomposition"

Waterfall Model

The Waterfall Model is a Dinosaur


red
yellow
green
blue
red
blue
yellow
green
blue


red
yellow
green
blue
red
blue
yellow
green
blue

Controlling Software Development with a Process

How do we control software development? Two opinions:

- Through organizational maturity (Humphrey)
  - Use a defined process, Capability Maturity Model (CMM)
- Through agility (Schwaber):
  - Large parts of software development is empirical in nature; they cannot be modeled with a defined process
  - Use an empirical process
- How can software development be controled?

⇒ Humphrey: with a defined process control model

⇒ Schwaber: with an empirical process control model.

Defined Process Control Model

- Requires that every piece of work is completely understood
- Deviations are seen as errors that need to be corrected
- Given a well-defined set of inputs, the same outputs are generated every time when the defined process is applied
- Precondition to apply this model:
  - All the activities and tasks are well defined to provide repeatability and predictability
- If the preconditions are not satisfied:
  - Lot of surprises, loss of control, incomplete or wrong work products.

Empirical Process Control Model

- The process is imperfectly defined, not all pieces of work need to be completely understood
- Deviations are seen as opportunities that need to be investigated
  - The empirical process "expects the unexpected"
- Control is exercised through frequent inspection
- Conditions when to apply this model:
  - There are frequent changes in the project, unpredictable inputs and unrepeatable outputs.

Summary

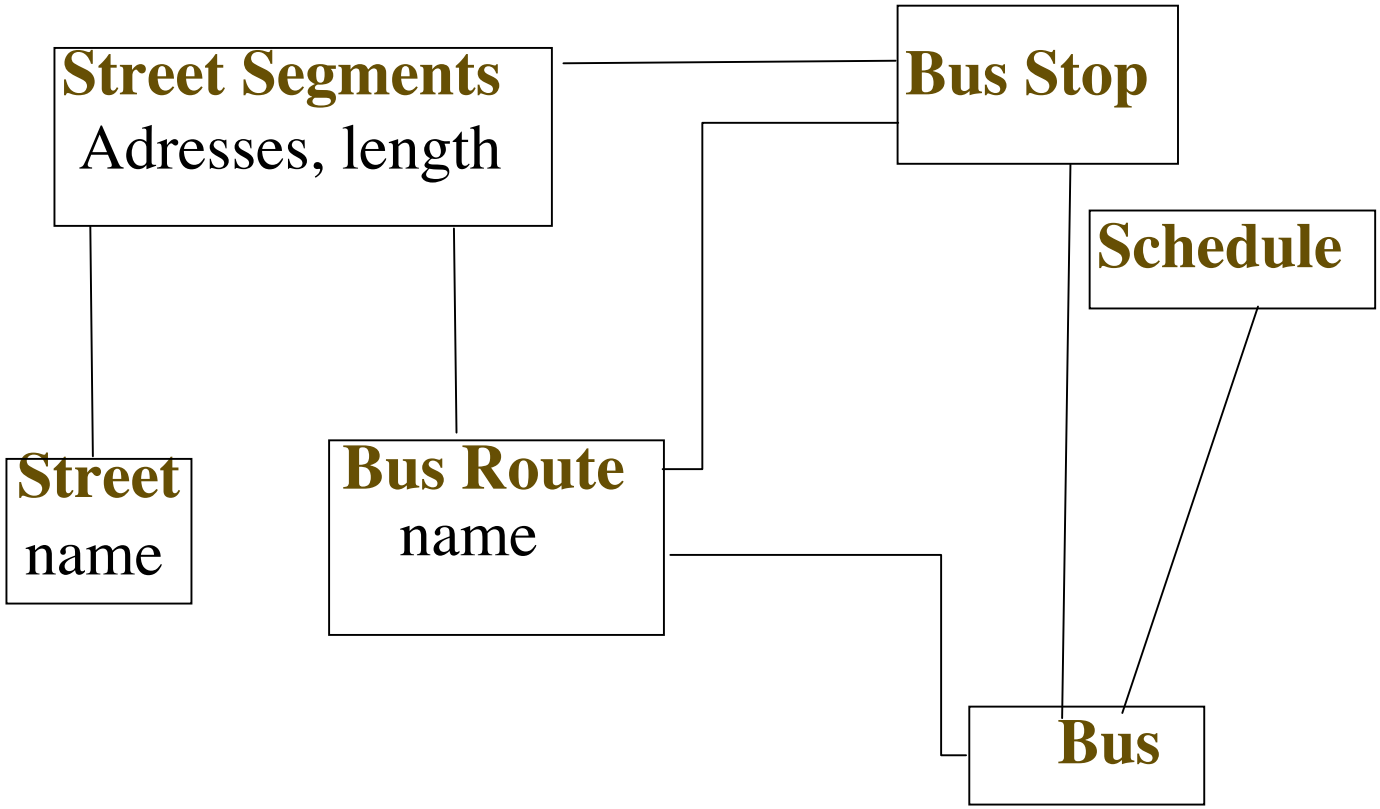
- A project has many contexts
  - Goals, client types
  - Environment, methods, tools, methodology
- Methodology issues
  - Planning, design reuse, modeling, process, control&monitoring
- Different types of planning
  - European vs. Polynesian navigation
- Different types of models
  - For communication, specification and archival
- Different ways to control processes
  - Defined vs. empirical process control models.

Additional References

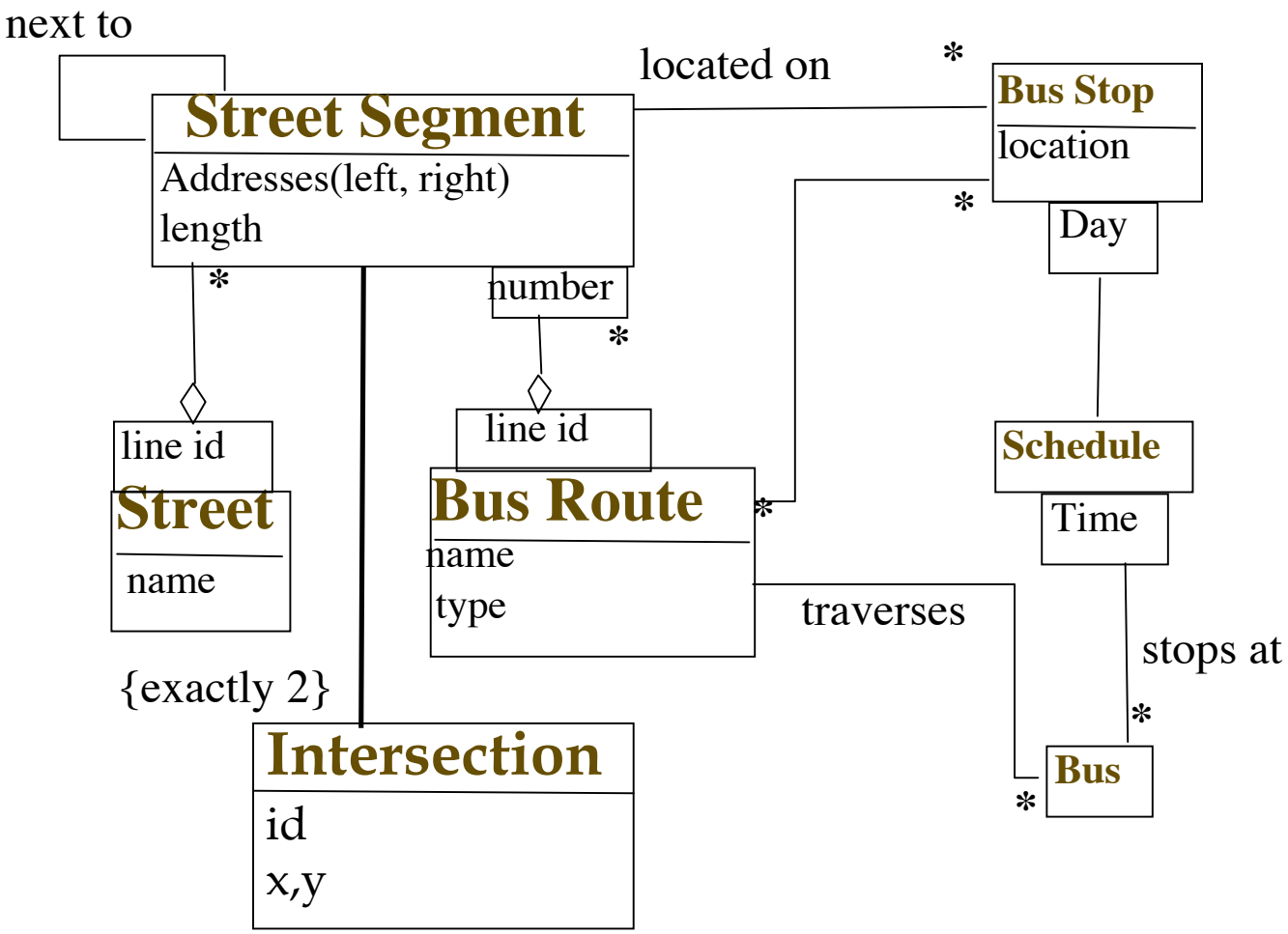
- W. Humphrey
  - Managing the Software Process, Addison-Wesley, Reading MA, 1989
- K. Schwaber, M. Beedle, R. C. Martin
  - Agile Software Development with Scrum, Prentice Hall, Upper Saddle River, NJ, 2001.

Backup and Additional Slides

Model for Bus Stops (used in Slide Presentation)



A UML Model for Bus Stops



For many people, moving away from defined processes means descending into chaos.

However, a process can be controlled even if it cannot be defined

Auckland Project Plan (European Navigation)

- Project Goal: Auckland
- Desired Outcome: Auckland is found
- Team: Captain and 50 sailors
- Organization: Hierarchical
- Tools: Compass, speed meter, map
- Methods: Determine planned course, write planned course before departure.
- Work breakdown structure
    - Task T1: Determine current direction of ship
    - Task T2: Determine deviation from desired course
    - Task T3: Bring ship back on course
- Process:
    - Execute T1 and T2 hourly. If there is a deviation, execute T3
- Schedule: 50 days, if the wind is good; 85 days, if doldrums are encountered.

Auckland Project Plan (Polynesian Navigation)

- Project Goal: Auckland
- Desired Outcome: A new place for living is found
- Team: Captain and 50 sailors Organization: Flat
- Tools: Stars and water temperature for navigation
- Methods: A set of event-action rules. Execution of actions is determined by the given context.
- Work breakdown structure
  - Task T1: Set direction of ship to a certain course
  - Task T2: Look for clouds in the distance
  - Task T3: Look for birds and determine their direction
  - Task T4: Determine new course for ship
- Process: Start with T1. Execute Tasks T2 and T3 regularly. The result (cloud detected, birds detected) is interpreted in the current context. Depending on the interpretation execute task T4 and T1.
- Schedule: None

Enabling Technologies for Light-Weight Processes

- Internet
- Self-Organizing Teams
- Peer-to-Peer Communication
- Ability to Change Plans
- Situated Actions

What type of process do we need?

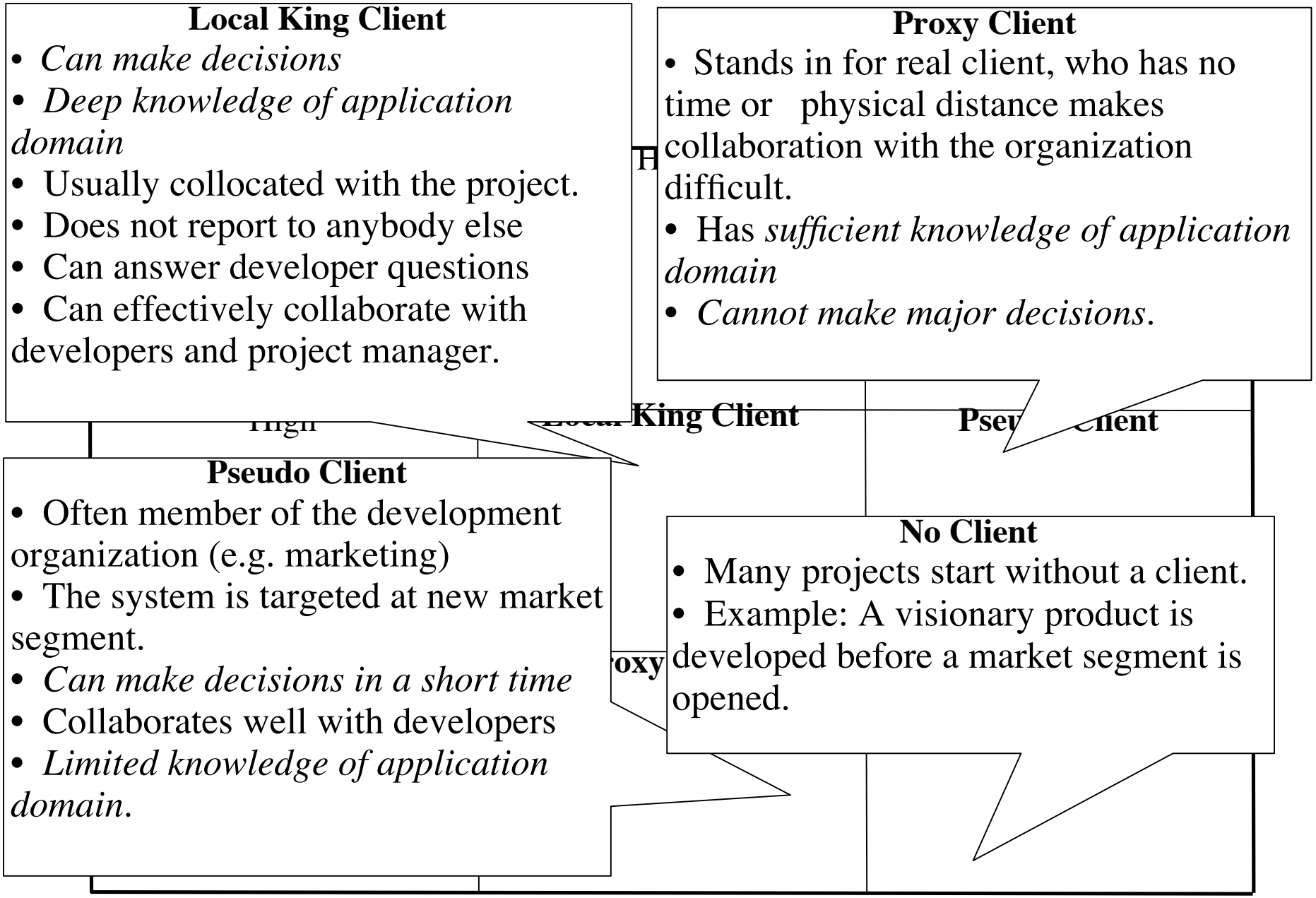
- Big vs Small Systems
  - Space-Shuttle, fly-by-wire
  - OS kernels, searching, sendmail
- Embedded Systems
  - Airbag controllers
  - Brake systems

"Blue Collar" Applications

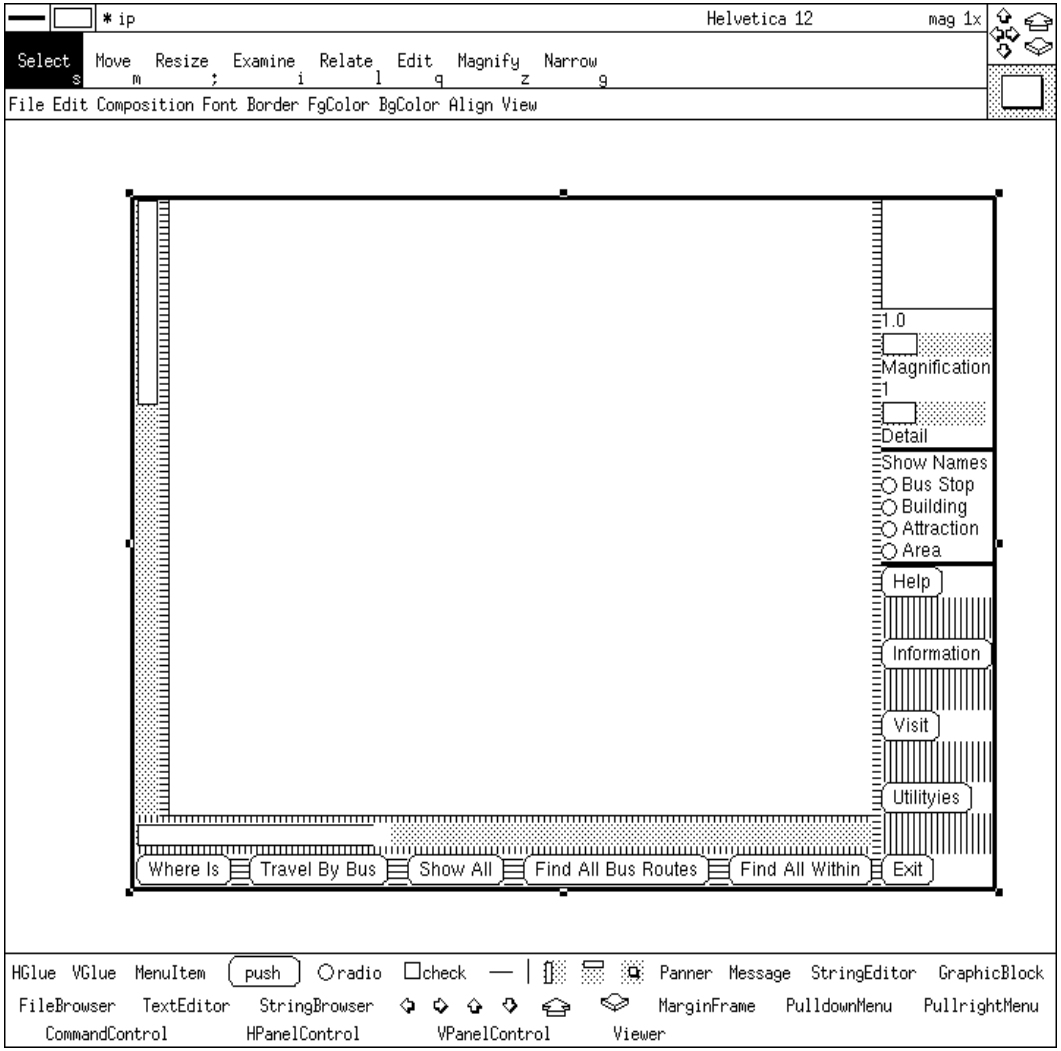
Mobile Systems
Mobile Maintenance, Mobile Health care

Augmented Reality Systems
Overlay of virtual information on real objects in real time

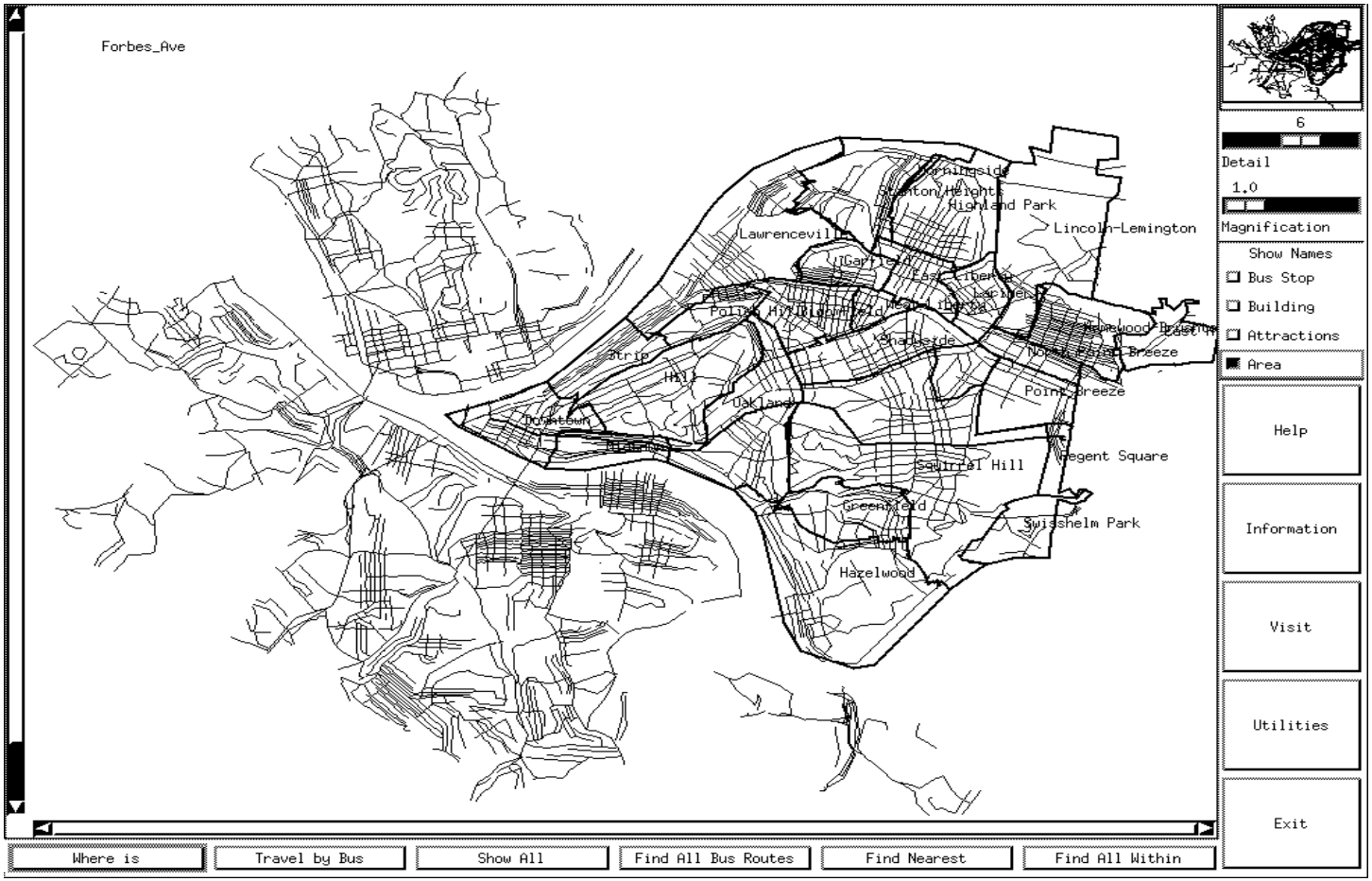
Flying Bridges


Local King Client
• Can make decisions
• Deep knowledge of application domain
• Usually collocated with the project.
• Does not report to anybody else
• Can answer developer questions
• Can effectively collaborate with developers and project manager.
Proxy Client
• Stands in for real client, who has no time or physical distance makes collaboration with the organization difficult.
• Has sufficient knowledge of application domain
• Cannot make major decisions.
Pseudo Client
• Often member of the development organization (e.g. marketing)
• The system is targeted at new market segment.
• Can make decisions in a short time
• Collaborates well with developers
• Limited knowledge of application domain.
No Client
• Many projects start without a client.
• Example: A visionary product is developed before a market segment is opened.

Input for User Interface Generator



Screen Snapshot of Graphical User Interface



UML can model more than Software Systems

- UML has been designed to model software artifacts.
- However, UML is a modeling language that can be used to model a variety of phenomena
  - projects and processes, even philosophical systems.
- The models for projects and processes used in the book are models intended for communication.