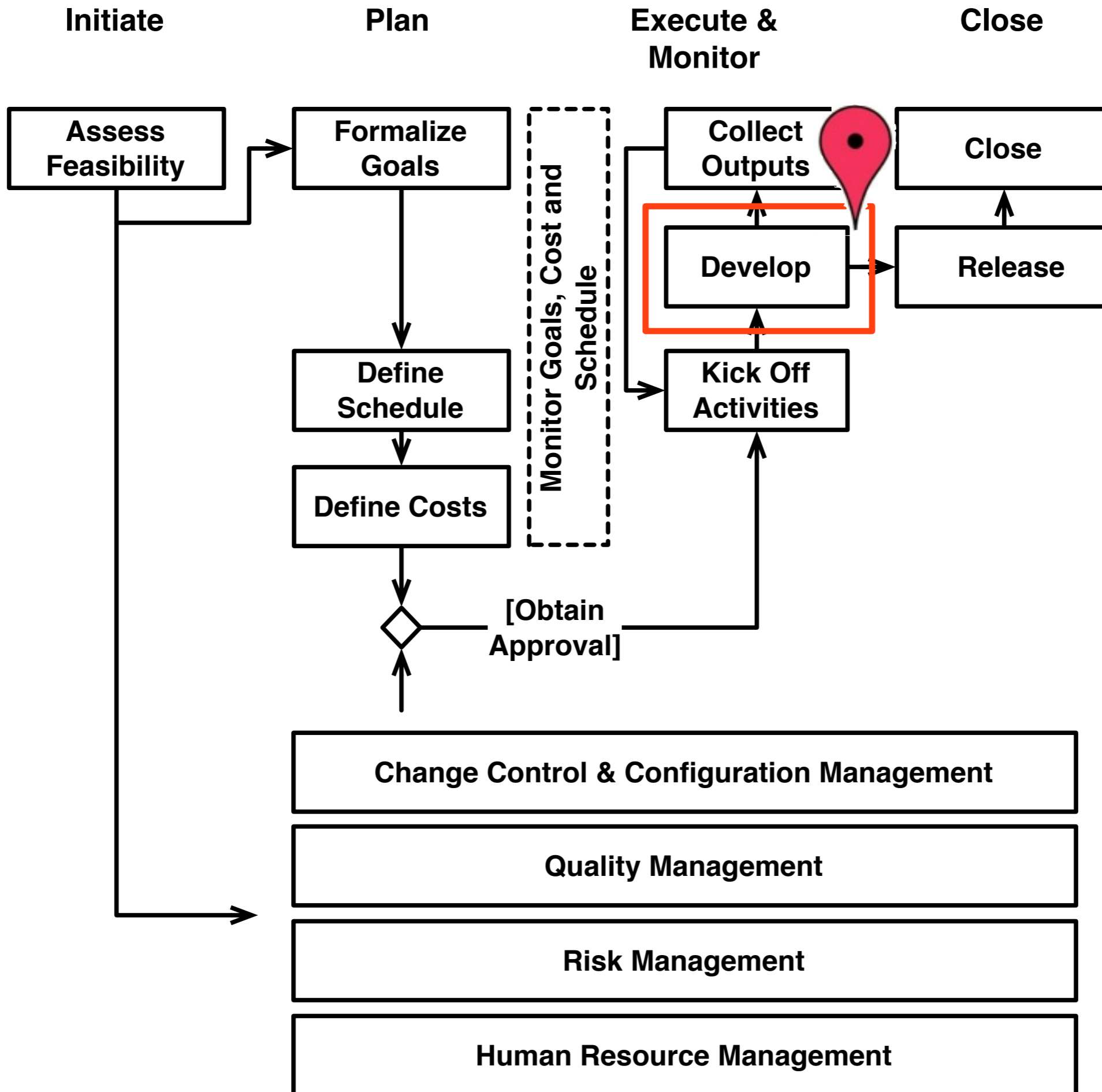


# (Traditional) Software Development Processes

---

# Goals of the Unit

- A gentle and high-level introduction to software development activities
- Understanding what are the building blocks for producing software
- Remarks:
  - This is no substitute for a software engineering course
  - The activities need to be integrated in a coherent process, to make sense
  - Software development projects range from the very small to the very large ... not all activities equally useful or relevant in any context



# Overview

- Software development is a progressive refinement which moves from concept to operations through the following phases:
  - Requirements and User Experience Design
  - Design
  - Implementation
  - Verification and Validation
  - Deployment
  - Operations and Maintenance
- As we move along these phases, we make and commit to specific choices; the cost of changes increases accordingly
- Different processes put different emphasis on each activity or define the order in which these activities can be performed

# Requirements Management

---

# Requirements

- Goal:
  - Forming a shared view about the characteristics of the system to build
- Output:
  - List of requirements, presented as:
    - \* a text document
    - \* a list of user stories
    - \* a set of diagrams (e.g., use case diagrams) and corresponding textual descriptions

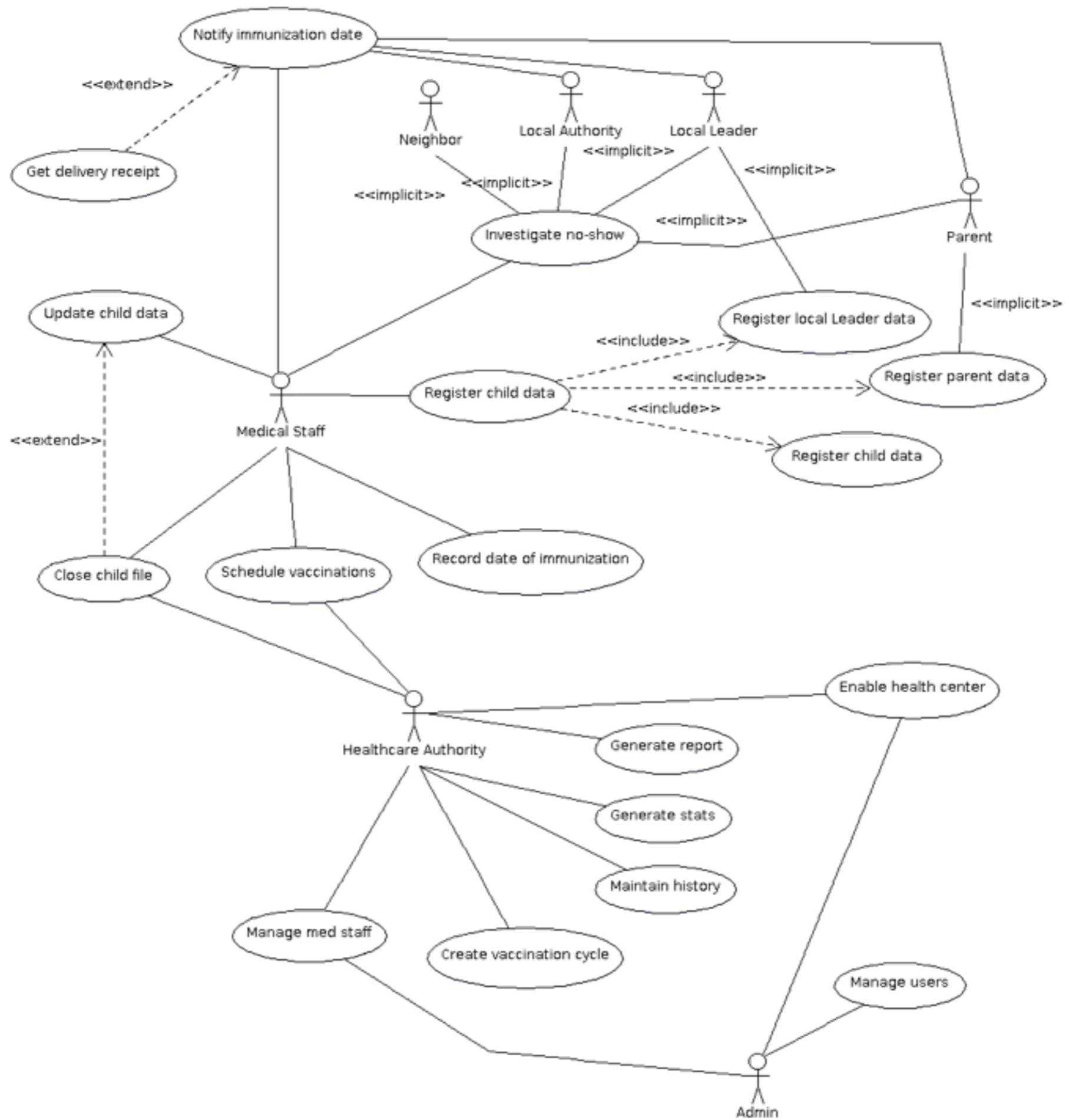
# List of Requirements

- Format:
  - Free or structured text describing the functions and other properties of a system
- Advantages
  - Simple to draft and distribute
  - The format can be used to keep track of changes (versioning)
- Disadvantages
  - No focus on user interaction: it can be difficult to understand for a customer
  - Ambiguities and incoherencies; interactions among requirements

# Use Case Diagrams

- Format:
  - Diagrams describing the interaction between users and the system
  - Textual description of the interaction as a sequence of steps
- Advantages
  - Intuitive, simpler to understand for a customer
  - It focuses on what the system does (user functions)
- Disadvantages
  - Difficult to represent and keep track of non-functional requirements
  - Managing diagrams requires a bit more work than working with text only





# User Stories

- Format:
  - Structured textual descriptions of user functions: *As a [user] I want to do [this] because [of that]*
- Advantages
  - Intuitive, compact, and simple to understand for a customer
  - It focuses on what the system does (user functions)
- Disadvantages
  - Difficult to represent and keep track of non-functional requirements
  - It is a partial specification (many details need to be worked out during the implementation) - used by Agile methodologies

# Requirements Engineering

- Goal:
  - Define and maintain requirements over time
- Activities:
  - Requirements elicitation  
(workshops, brainstormings, focus groups, ...)
  - Requirements structuring
  - User experience design
  - Requirements validation

# Requirements Structuring

- Goal:
  - Improving maintenance of requirements over time
- Tools:
  - Isolated and made identifiable (reason and manipulate each requirement more easily)
  - Organized and classified (e.g., FURPS)
  - Annotated (priority, importance, traceability, ...)

# User Experience Design

- Goal:
  - Providing a coherent and satisfying experience on the different artifacts that constitute a software system, including its design, interface, interaction, and manuals
- Tools:
  - **User-centered analysis:** understanding how users will interact with the system (focus groups, experiments)
  - **User-centered design:** specifying how users will actually interact with the system (mock-ups)

# Requirements Validation

- Find (and address):
  - Inconsistencies
    - \* scenario 1: **R1. A; ...; Rn: not A**
    - \* scenario 2: **R1. forall x. A(x); ...; Rn: not A(c)**
  - Incompleteness
    - \* the behavior is not specified for certain cases and situations (often non-nominal situations)
  - Duplicates
    - \* the same requirements is described twice (possibly in different ways)

# Business Process Modeling and Re-engineering

---

# Organizations and Software

- Software has to be designed to fit an organization's operational structure
- However: software can also change the way in which an organization work
- **Business process modeling** models the way in which an organization works
- **Business process re-engineering** plans the way in which an organization works, to make its operations more efficient (“as is” and “to be”)



# Business Process Modeling

- Articulated and complex, it is **sometimes planned and organized as an independent project**
- Conducted with interviews, document analysis, shadowing
- Information to collect:
  - **Organizational structure**: chain of responsibility and accountability
  - **Business processes**
  - **Existing IT infrastructure**: hardware, systems, databases
  - **Business entities**: data produced and processed by the organization

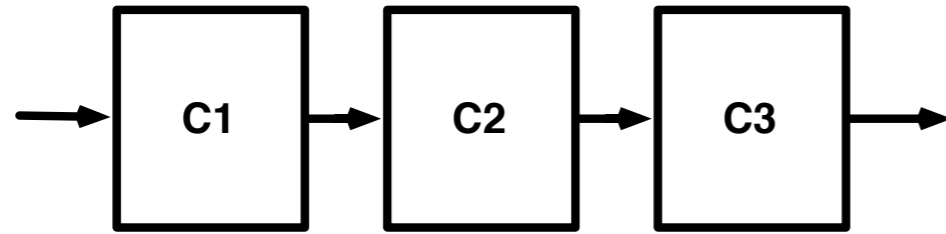
# System Design

---

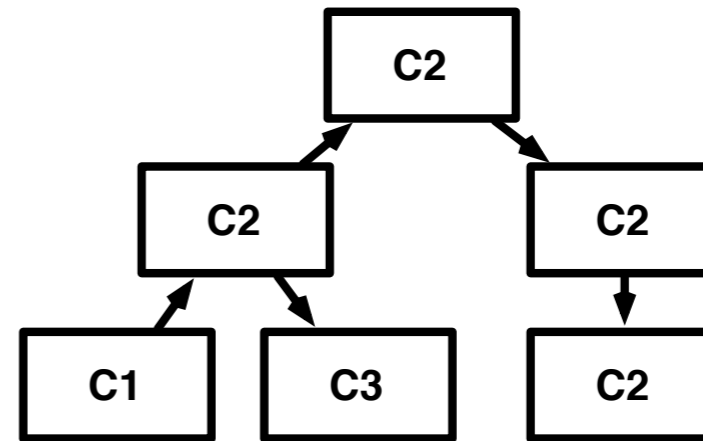
# System Design

- Goal:
  - Defining the structure of the software to build  
(= system architecture)
- Outputs:
  - components which constitute the system
  - functions each component implements
  - how the components are interconnected
- The activity is relevant also for managerial reasons: the system architecture provides a “natural” decomposition of work

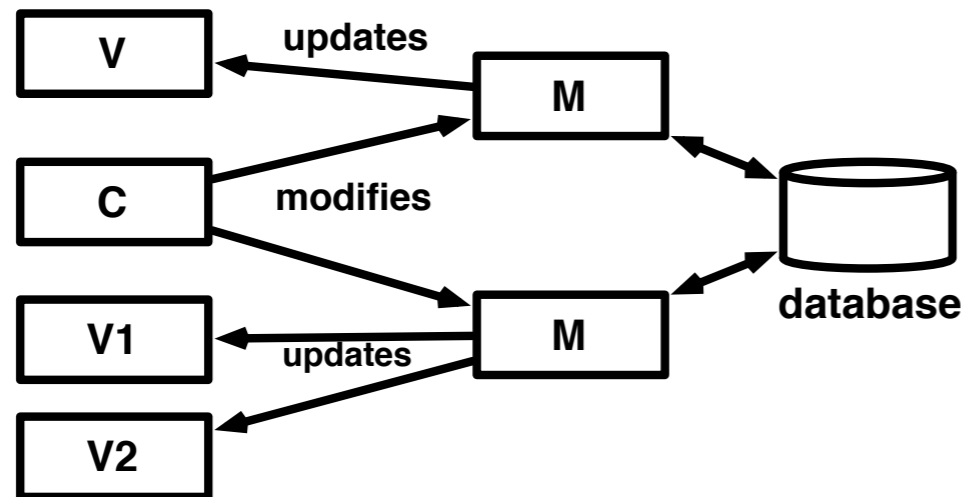
# Architectural Patterns



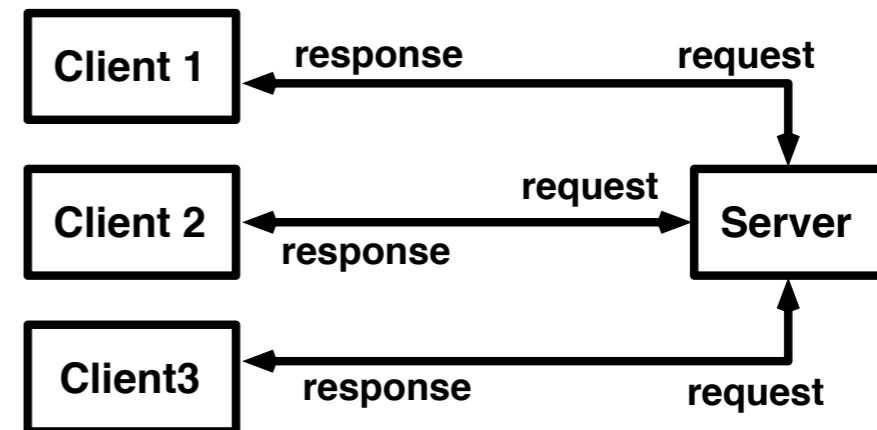
A Pipeline Architecture



A Layered Architecture



A Data-Centric application with two MVCs



A client-server Architecture

# Architectural Patterns

- Pipe and filter

- Composition of data processing units
- Focus: I/O specification

- Layered/Hierarchical

- Hierarchy of components
- Focus: control and information flow; block responsibilities

- Data-Centric

- MVC: data, presentation, and logic
- Focus: data model, operations
- Many web applications and many desktop applications use the data-centric architectural style

- Client-server

- Server (main functions) and clients (requesting services)
- Focus: communication protocols/service specifications

# Implementation

---

# Implementation

- Goal:
  - Writing the code!
- Some of the PM-relevant activities during implementation:
  - Collection of productivity and size metrics
  - Collection of quality metrics
  - Use of coding and documentation standards
  - Code management practices (versioning; code releasing standards)

# Verification and Validation

---



# Verification and Validation

- **Validation** = are we building the right system?
- **Verification** = did we build the system right?
- Collectively known with the acronym **V&V**
- Part of quality management
- The main (but not the only) way of performing V&V for software systems is **testing**

# Types of Testing

- Unit testing

- Scope: a piece of code, such as a class

- Integration testing

- Scope: the interaction between two components
- Mars Climate Orbiter bug: two components used different units (metric and imperial); ~400M USD loss.

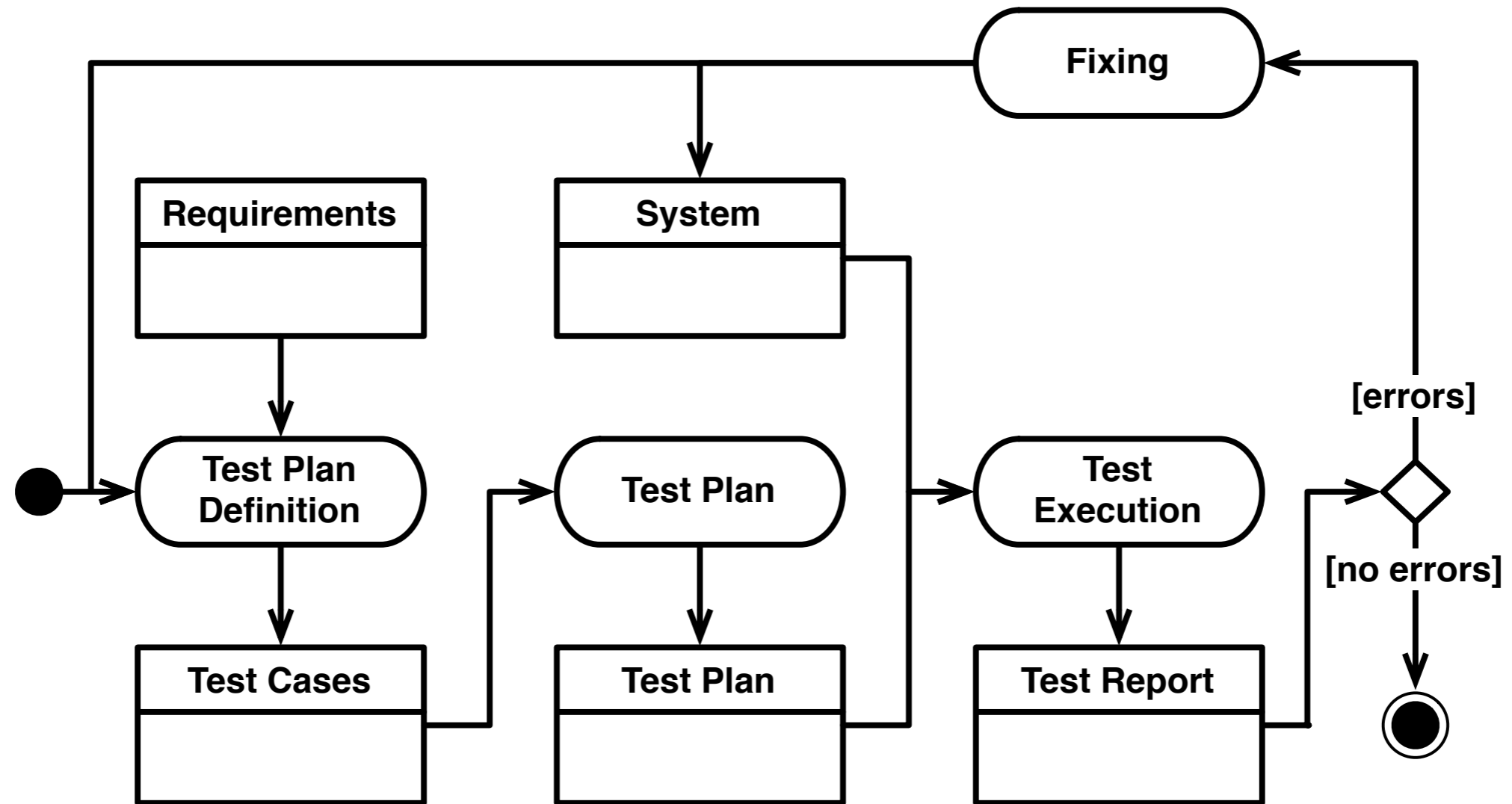
- System testing

- Scope: the system behaves as expected and implements correctly all the requirements
- Test cases

- Usability testing

- Scope: verifying whether the user experience and interaction is intuitive, effective, and satisfying
- Used to reduce the probability of human errors (safety-critical systems).

# The System Testing Process



# Deployment

---

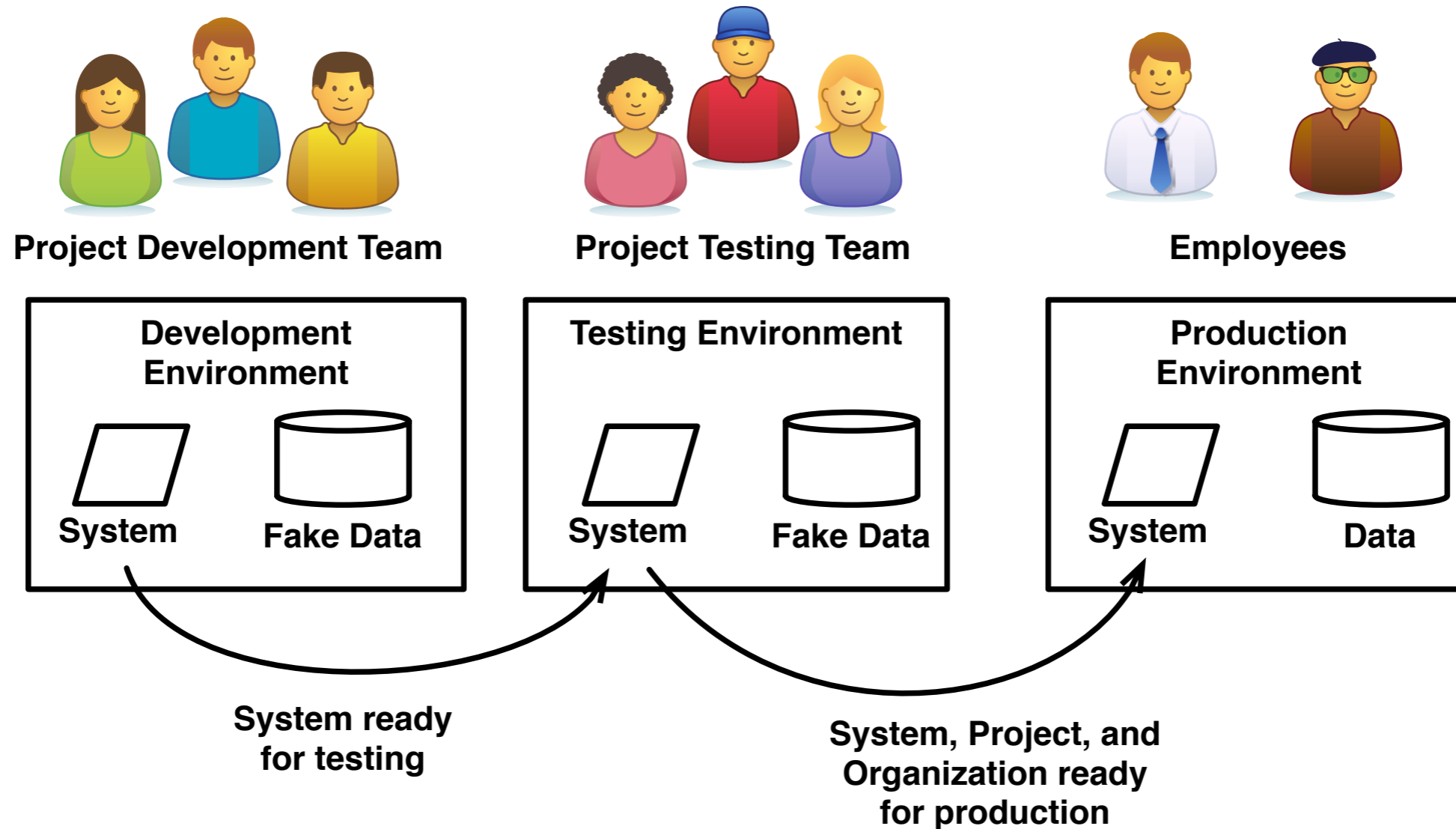
# Deployment

- Goal
  - Installing the new system and making it operational
- Some concerns:
  - Ensuring continuity of business operations
  - Migrating data
  - Transitioning to operations and maintenance
- Factors to consider:
  - The **human factor**: is the people ready to use the system?
  - The **data factor**: is all the data which is needed for the system to run available to the new software?
  - The **hardware factor**: are all interfaces ready and functional?

# Approaches

- **Cut-over:** the new system replaces the old one
- **Parallel Approach:** the old and the new system operate simultaneously for a period
- **Piloting:** the new system is installed for a limited number of users or for a specific business unit
- **Phased Approach:** functions are rolled out incrementally

# Managing Software Evolution



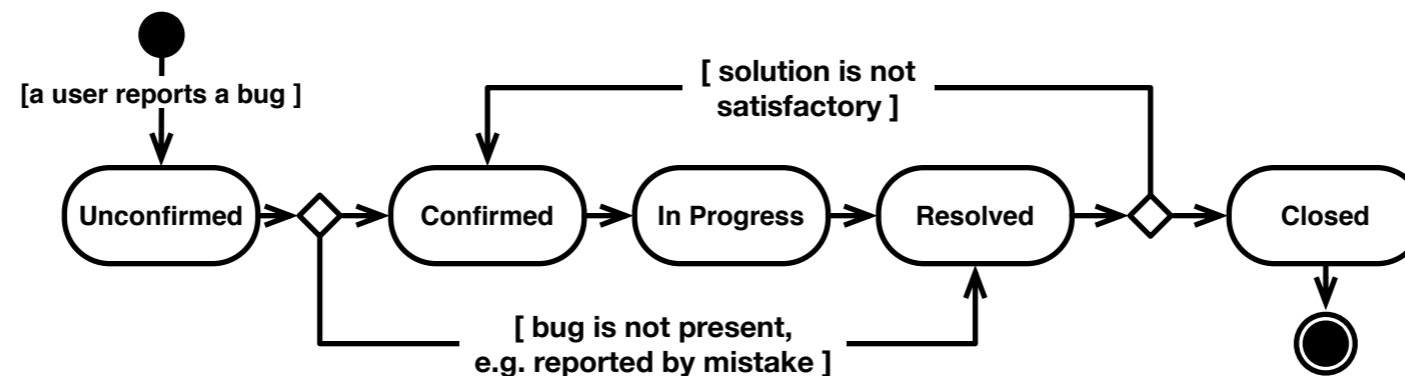
# Operations and Maintenance

---



# Operations and Maintenance

- Goal
  - Ensuring the system runs smoothly
- Activities:
  - Providing Technical Support
  - Monitoring system performance
  - Collecting and managing tickets (clarifications, bugs, requests for improvement)
  - Trigger maintenance activities



# Types of Maintenance

- **Corrective**, if relative to fixing an issue discovered after the release of the system
- **Preventive**, if relative to fixing an issue discovered, but not occurred (or, at least, signaled by users)
- **Adaptive**, if relative to adapt a system to changed external conditions
- **Perfective**, if relative to improve some characteristics of a system, like, for instance, performances