

# Feasibility Study

Before beginning a project, a short, low-cost study to **identify**

- Client
- Scope
- Potential benefits
- Resources needed:  
    staff, time, equipment, etc.
- Potential obstacles



**Where are the risks? How can they be minimized?**

# How to Minimize Risk?

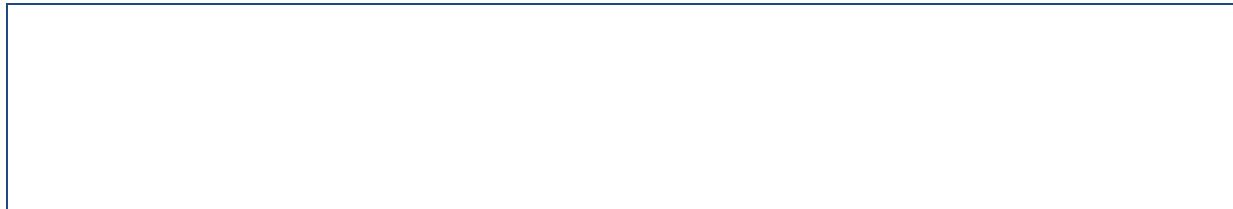
## *CS 501 Projects*

- Several target levels of functionality:  
*required, desirable, optional*
- Visible software process: intermediate deliverables
- Good communication within team and with  
Teaching Assistant

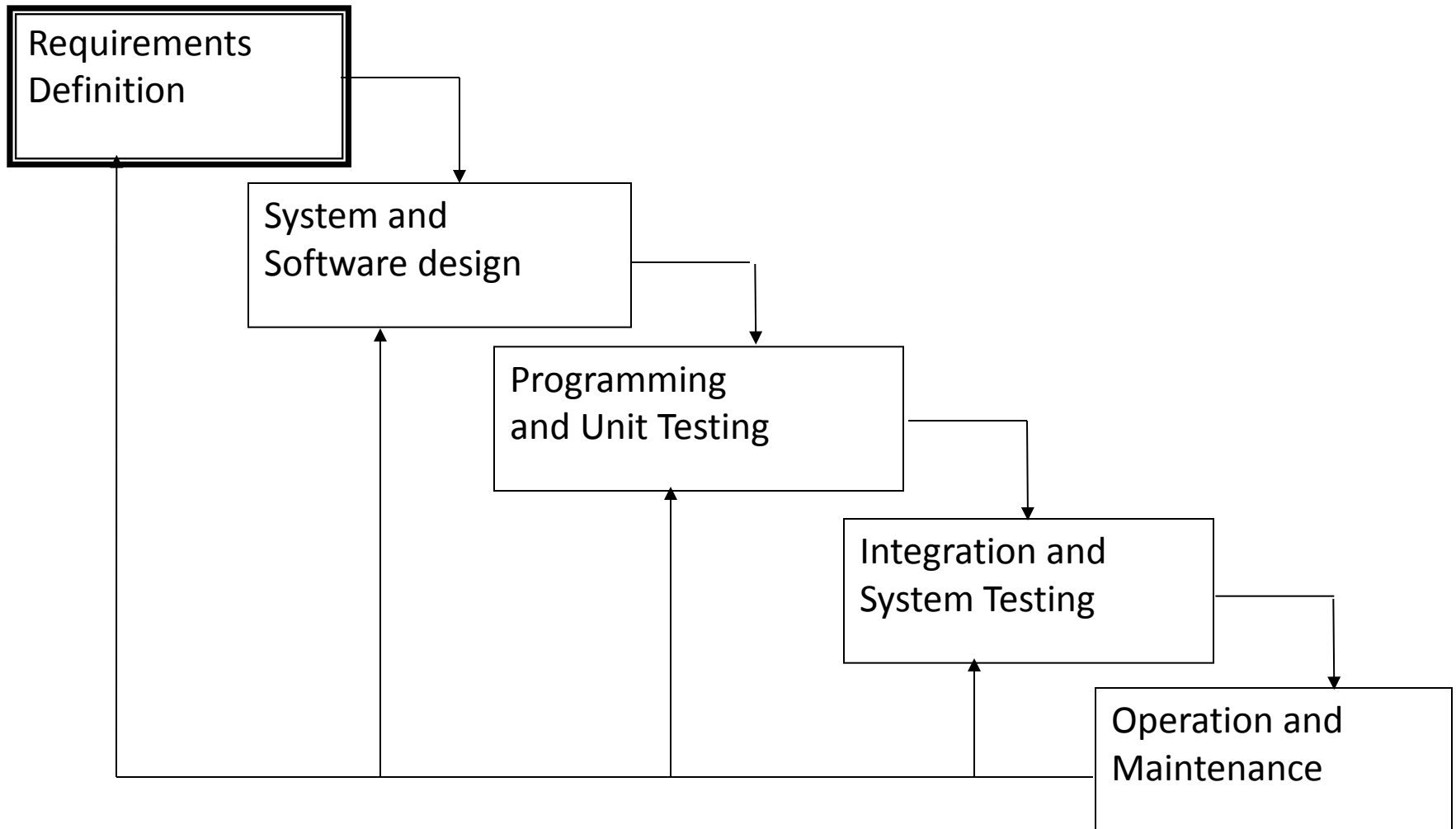


**Good processes lead to good software**

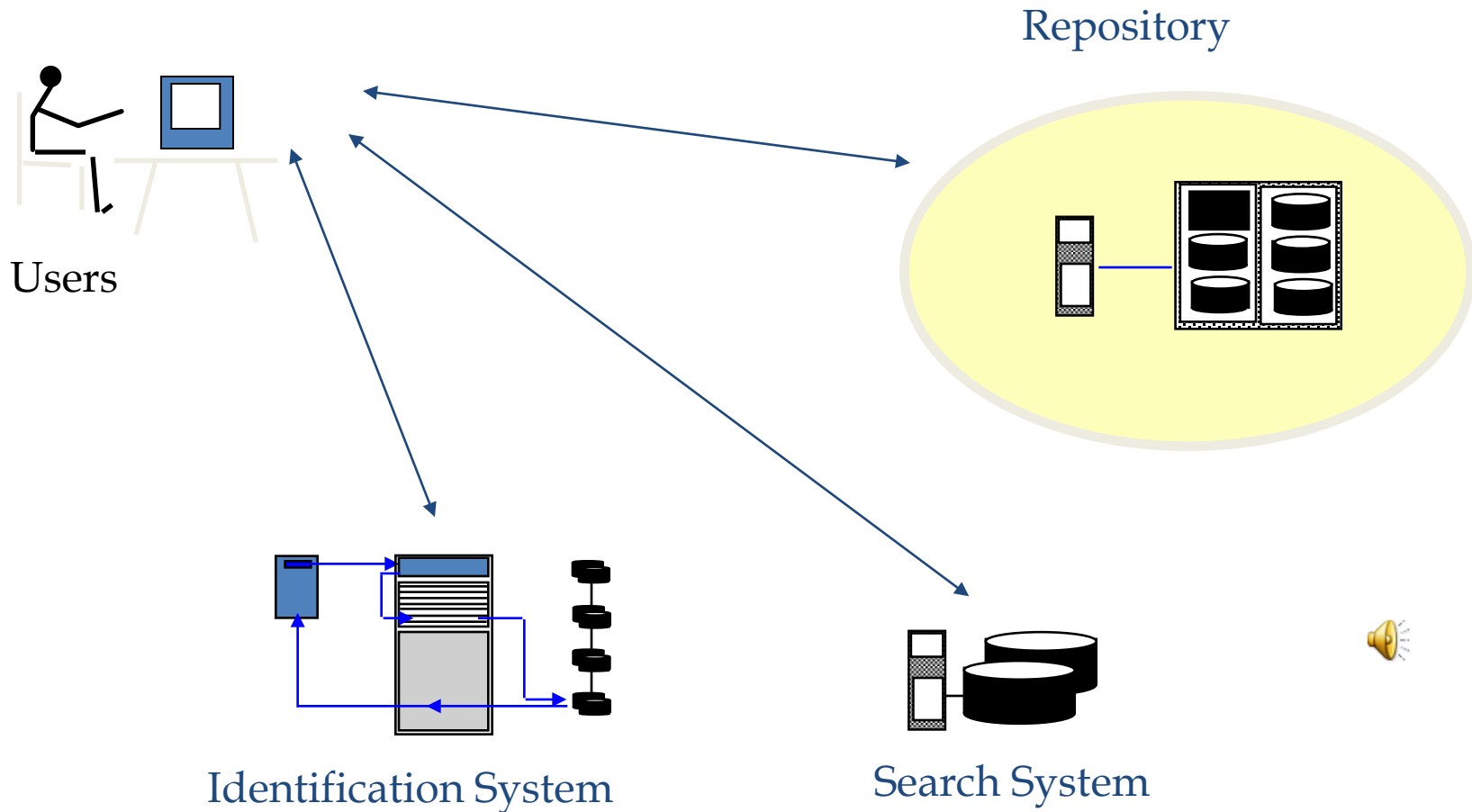
**Good processes reduce risk**



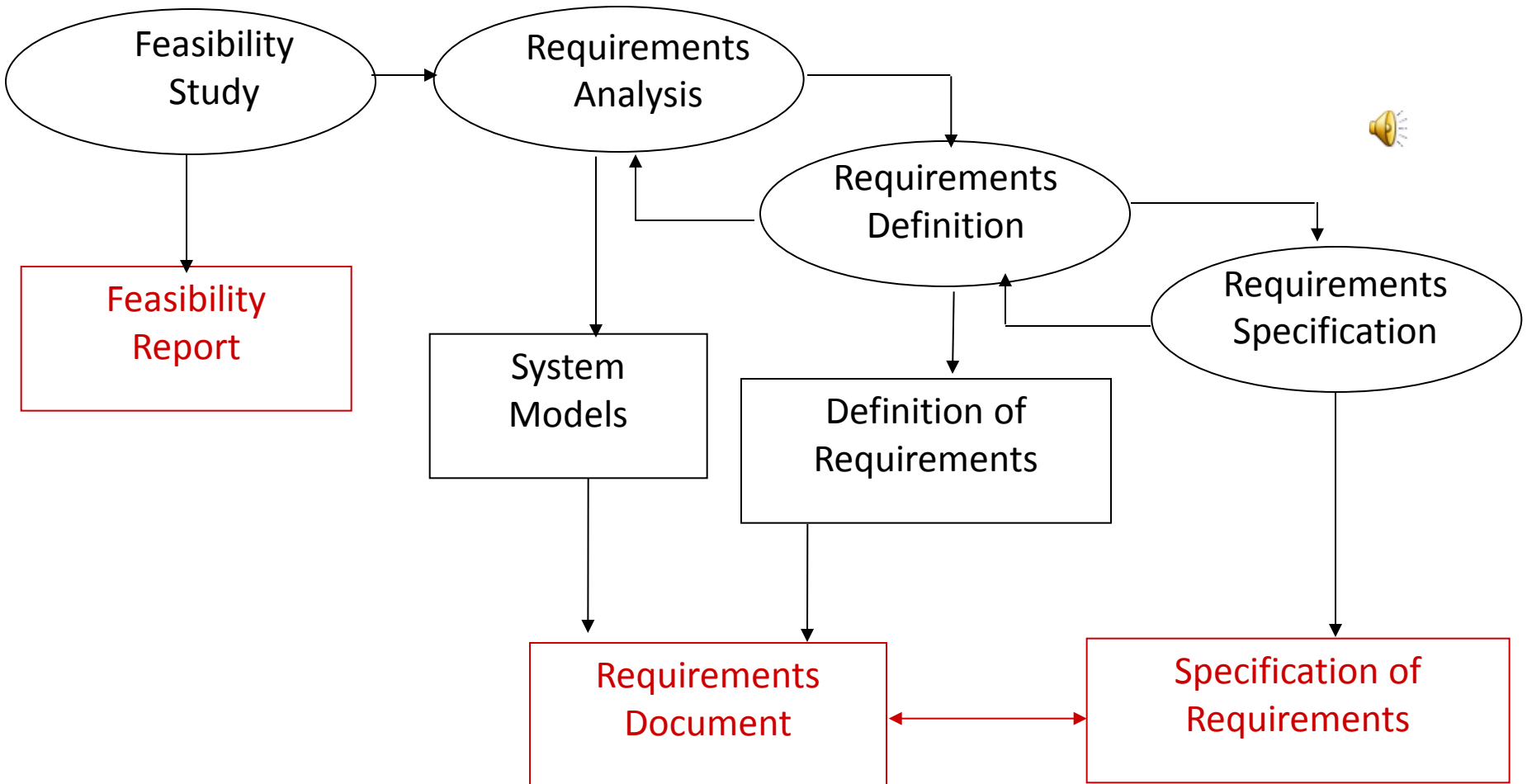
# Requirements Definition and Analysis



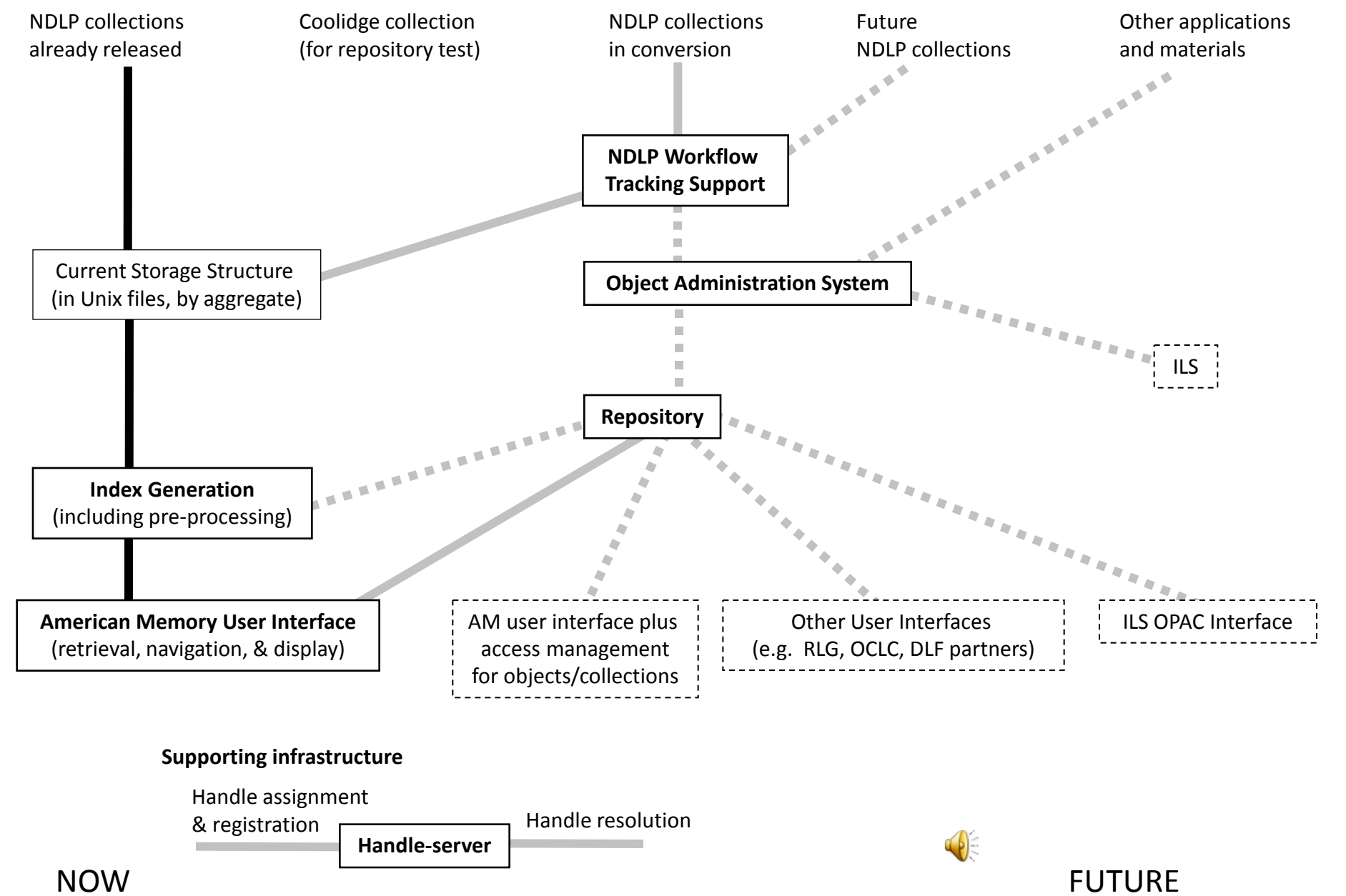
# The Repository



# The Requirements Process



**DRAFT OVERVIEW OF ITS SUPPORT**  
**FOR NDLP PRODUCTION AND DELIVERY OF AMERICAN MEMORY**



# Project Planning Methods

The Critical Path Method, Gantt charts, Activity bar charts, etc. are roughly equivalent.

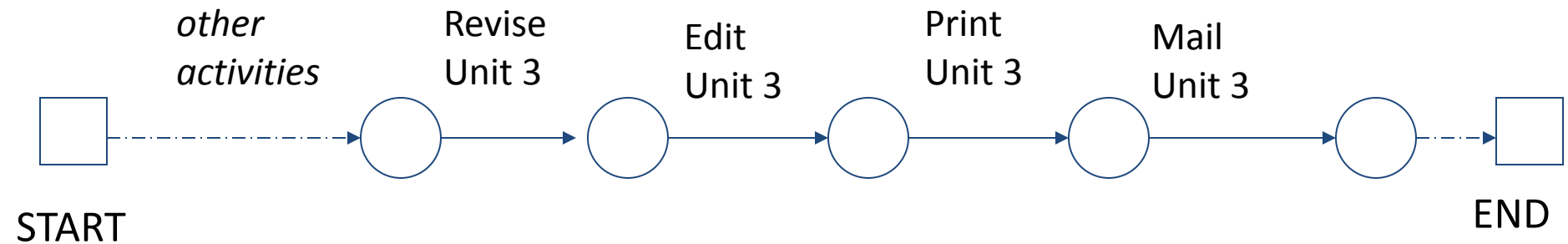
These methods are best when:

- Model is updated regularly (e.g., monthly)
- The structure of the project is well understood
- The time estimates are reliable
- Activities do not share resources

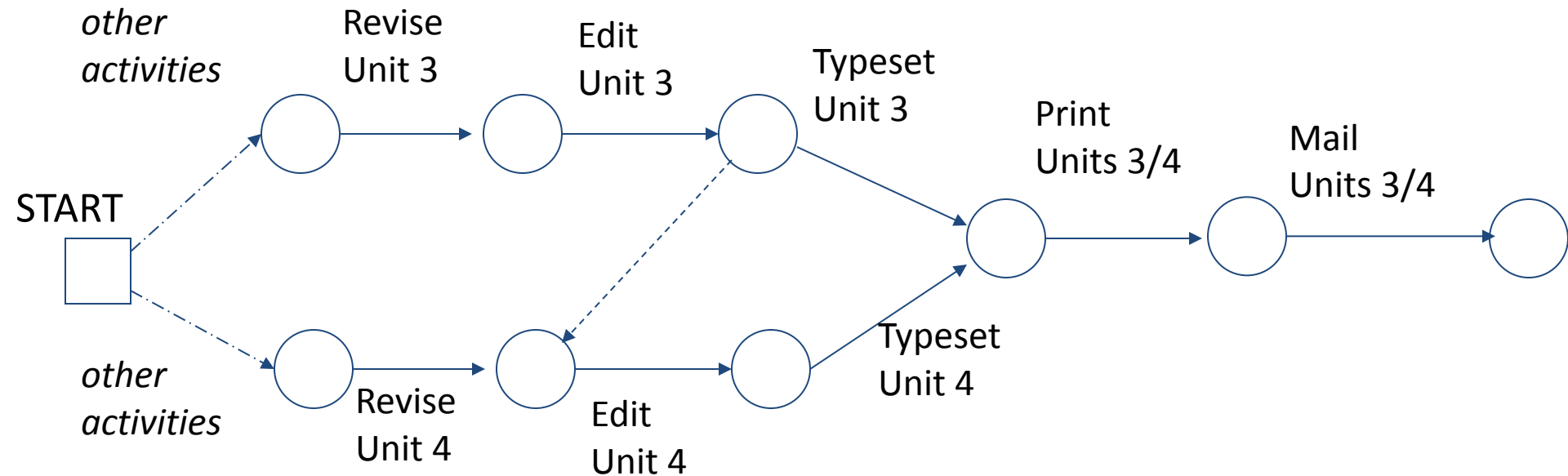
*[Critical Path Method is excellent for large construction projects.]*



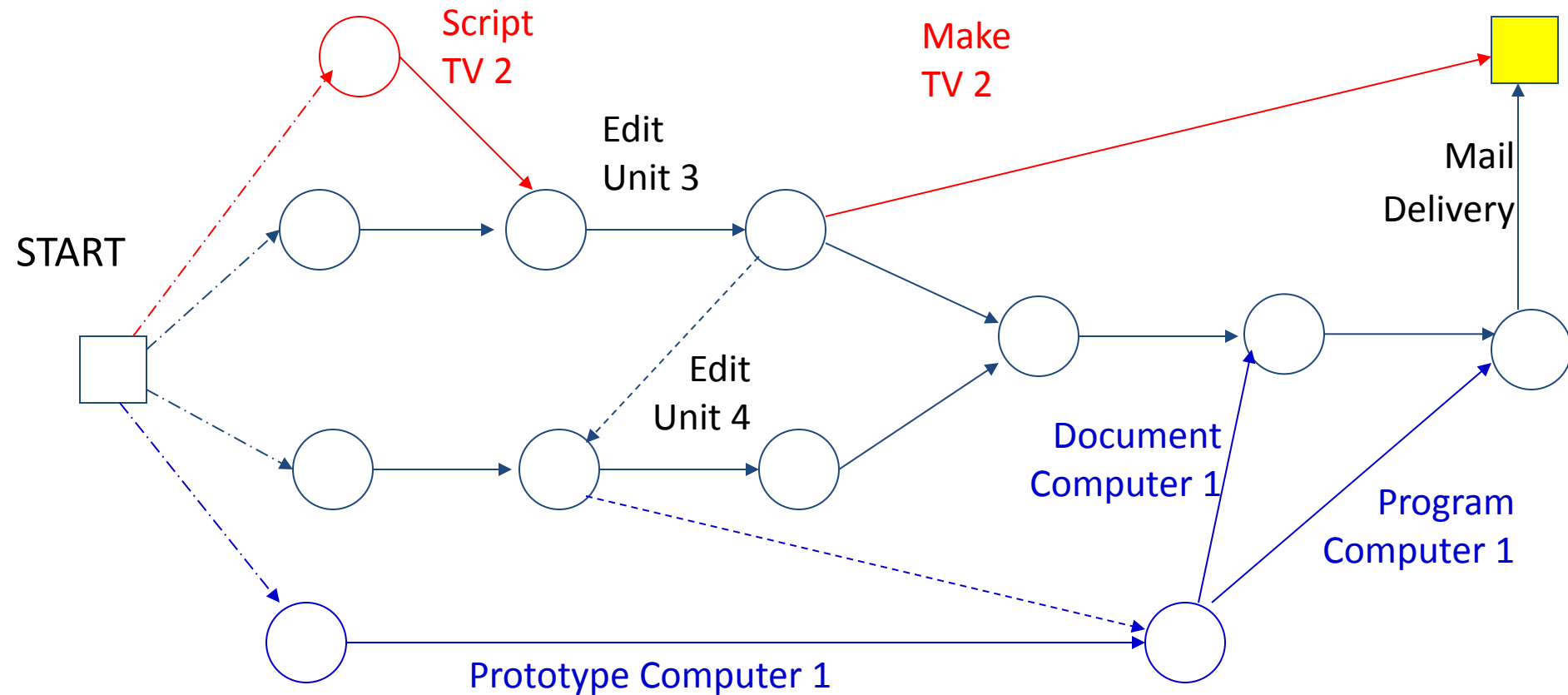
# Critical Path Method



# Critical Path Method



# Critical Path Method



# Key Personnel

*In computing, not all people are equal:*

- The best are at least 5 times more productive
- Some tasks are too difficult for everybody

*Adding more people adds communications complexity*

- Some activities need a single mind
- Sometimes, the elapsed time for an activity can not be shortened.

*What happens to the project if a key person is sick or quits?*



# Start-up Time

*On a big project, the start-up time is typically three to six months:*

- Personnel have to complete previous projects (fatigue) or recruited.
- Hardware and software has to be acquired and installed.
- Staff have to learn new domain areas and software (slow while learning)
- Clients may not be ready.

# Documentation

- Reasons for documentation:
  - visibility (e.g., project plan, interim report)
  - user support (e.g., user manual)
  - team communication (e.g., interface specifications)
  - maintenance and evolution (e.g., requirements)
- Characteristics of documentation:
  - accurate and kept current
  - appropriate for audience
  - maintained online (usually)
  - simple but professional in style and appearance



*Documentation is expensive --> Quality not volume*

# Form of Documentation

## *External*

- Printed
- Web site

## *Internal*

- Program documentation
- Program context (e.g., copyright notices)

# Characteristics of Software Products

General characteristics

Usability

Maintainability

Dependability

Efficiency

Good software products require good programming,

**but ...**

Programming quality is the means to the end, **not the end** itself.

Example: DEC's optical scanner



# Variety of Software Products

Software products are very varied

--> Client requirements are very different

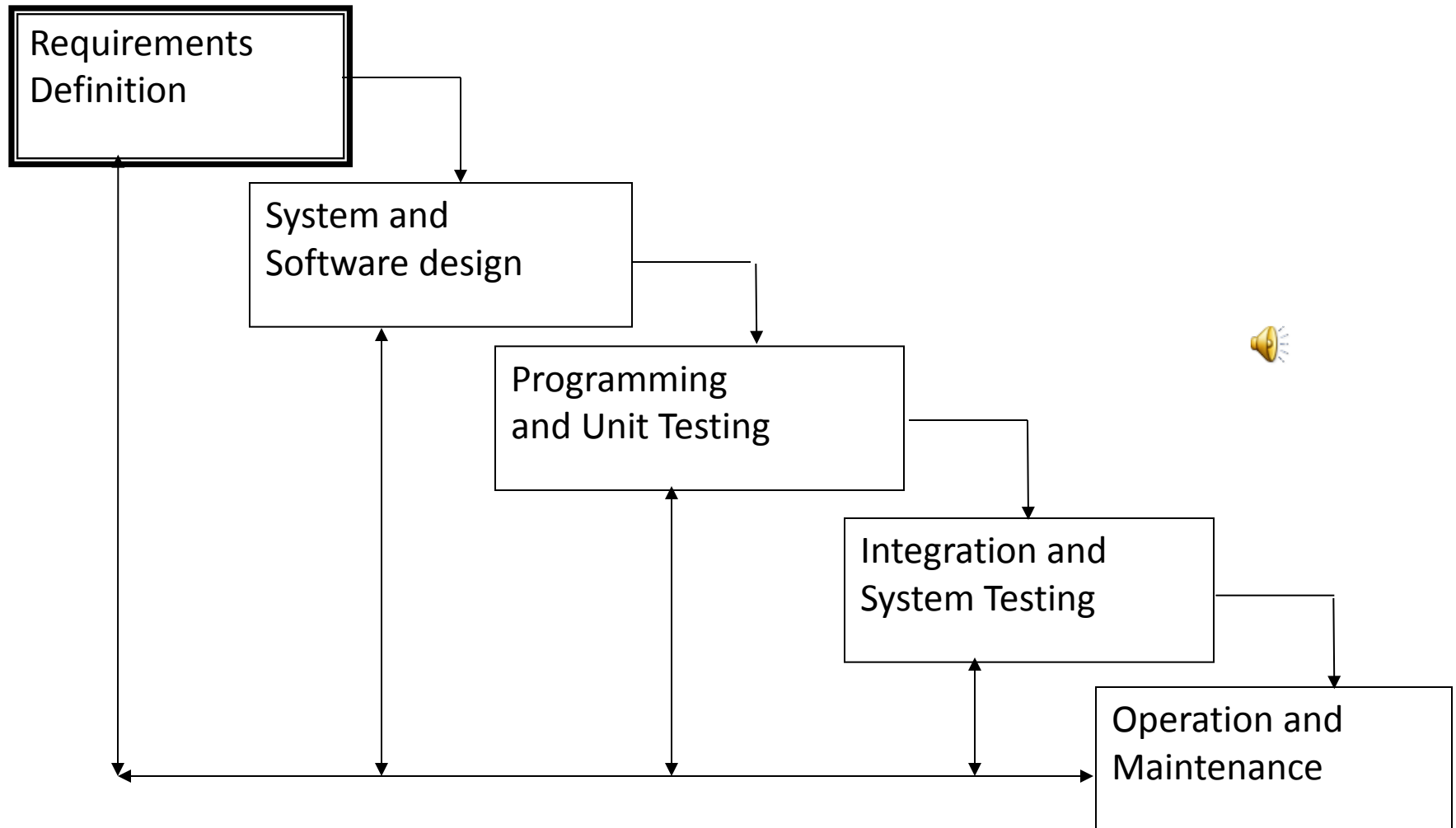
--> There is no standard process for software engineering

--> There is no best language, operating system, platform, database system, development environment, etc.

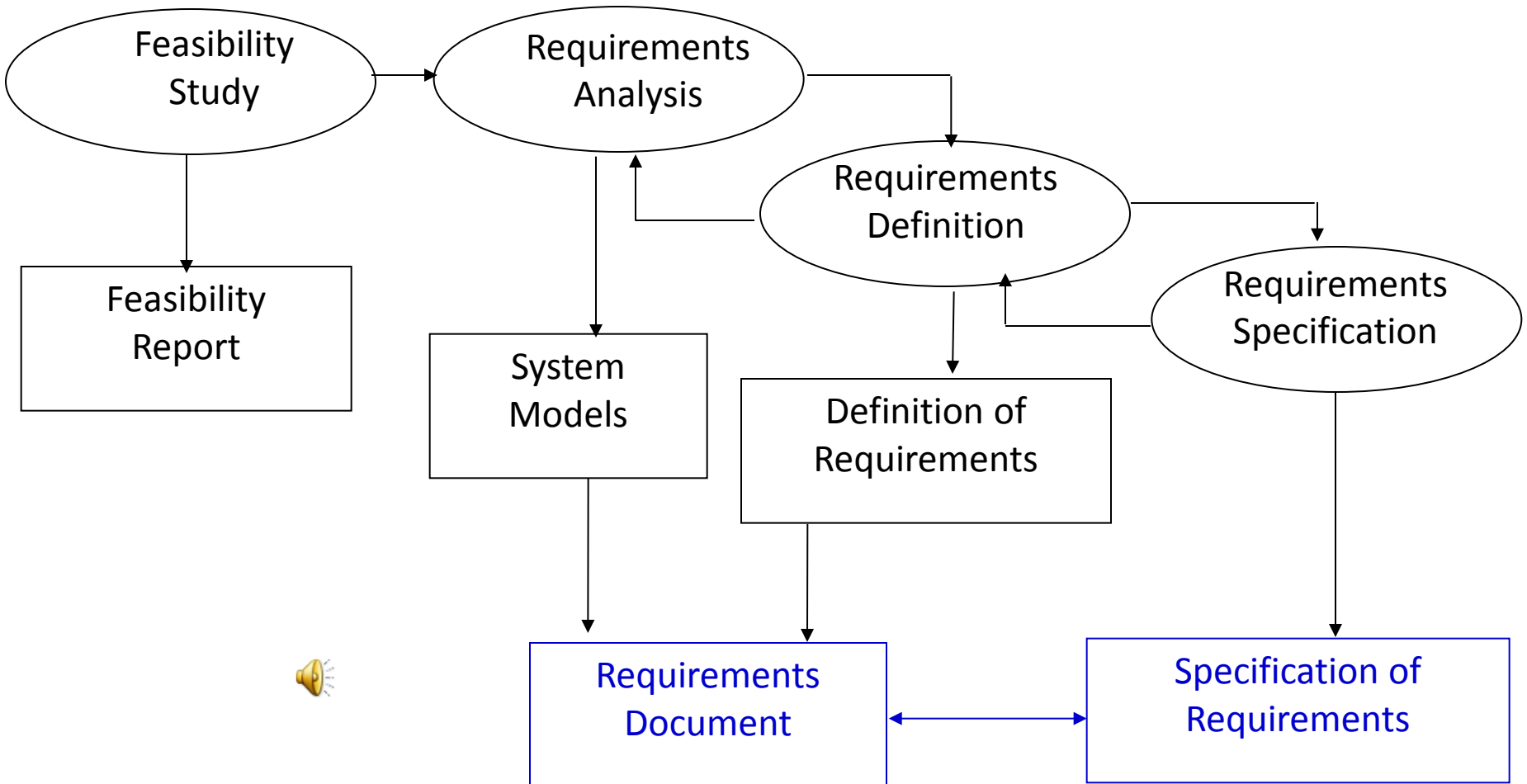
A skilled software developer knows about a wide variety of approaches, methods, tools. The **craft** of software engineering is to select appropriate methods for each project and apply them effectively.



# Requirements Definition and Analysis



# The Requirements Process



# Interviews with Clients

***Clients may have only a vague concept of requirements.***

- Prepare before you meet with them
- Keep full notes
- If you don't understand, delve further
- Small group meetings are often most effective

***Clients often confuse the current system with the underlying requirement.***

# Requirements Analysis v. System Design

## Dilemma.

- Requirements analysis should make minimal assumptions about the system design.
- But the requirements definition must be consistent with computing technology and the resources available.

In practice, analysis and design are interwoven. However, ***do not to allow the analysis tools to prejudge the system design.***



# Procedural Models: Pseudo-code

*Example: Check project project plan*

**check\_plan** (report)

**if** report (date\_time) > due\_date\_time **then** error (too\_late)

**if** report (client) = none **then** error (no\_client)

**if** report (team) < min\_team **or** > max\_team

**then** error (bad\_team)

**if** error() = none

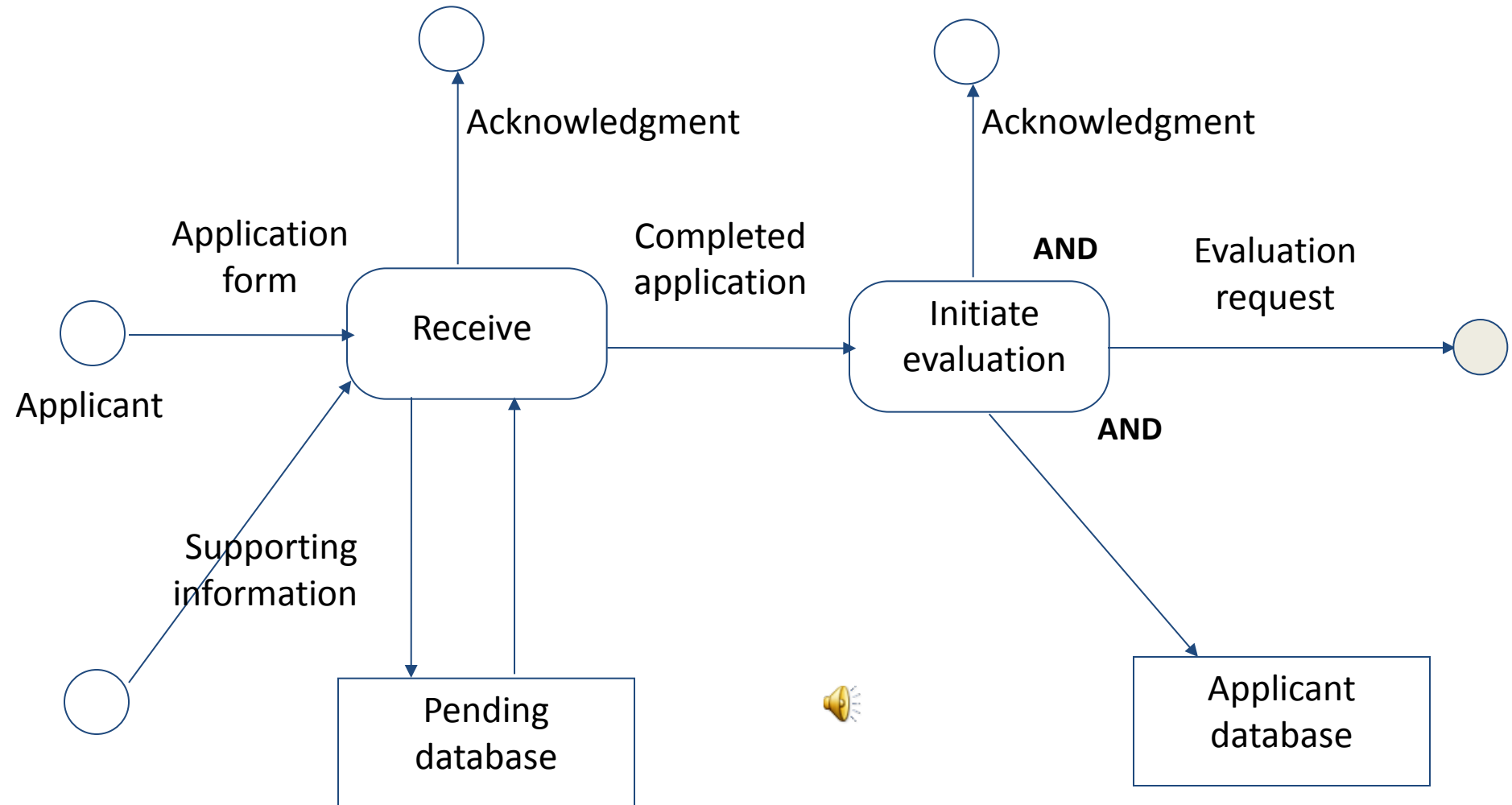
**then** comments = read\_report (report)

**return** (comments (text), comments (grade))

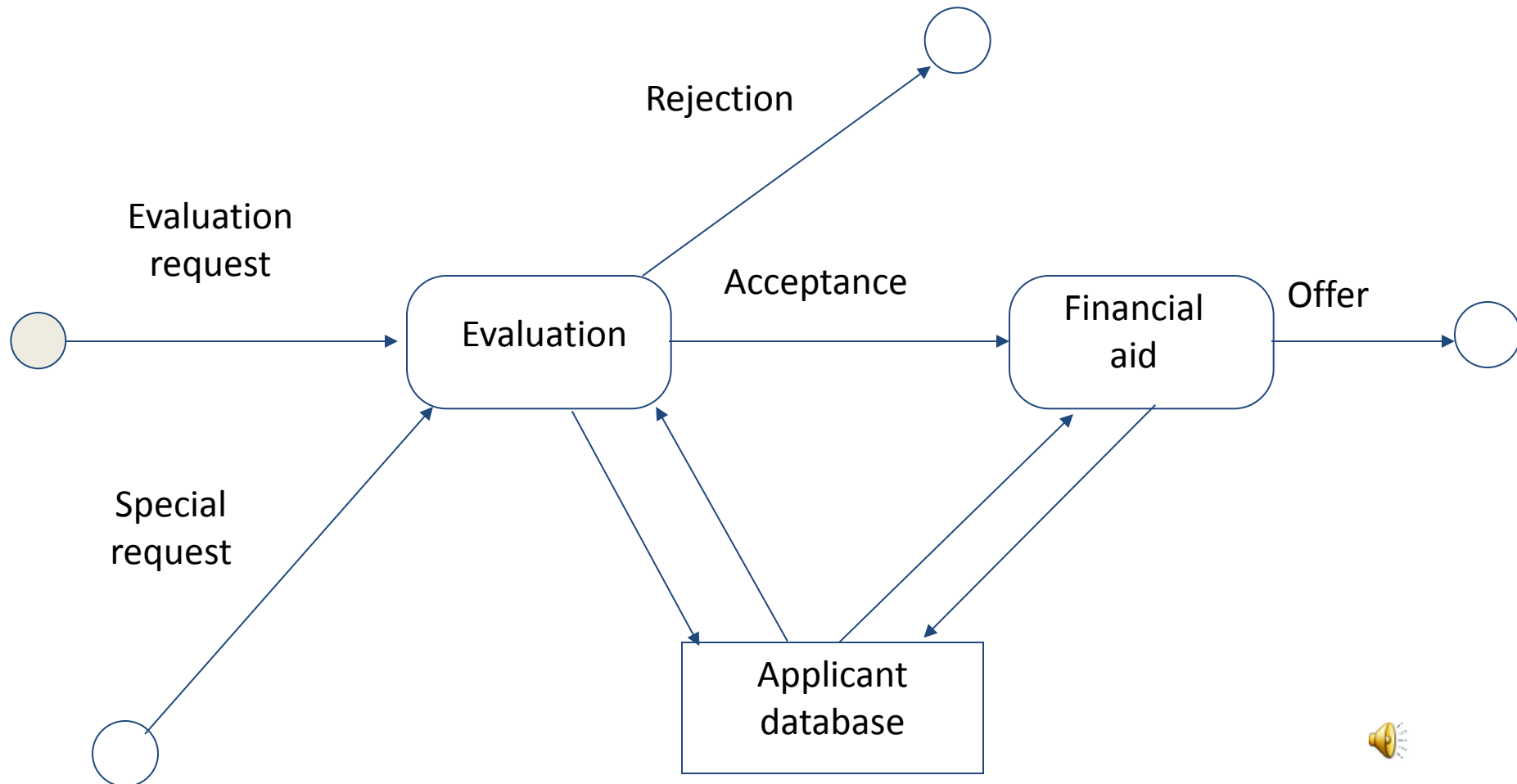
**else return** error()



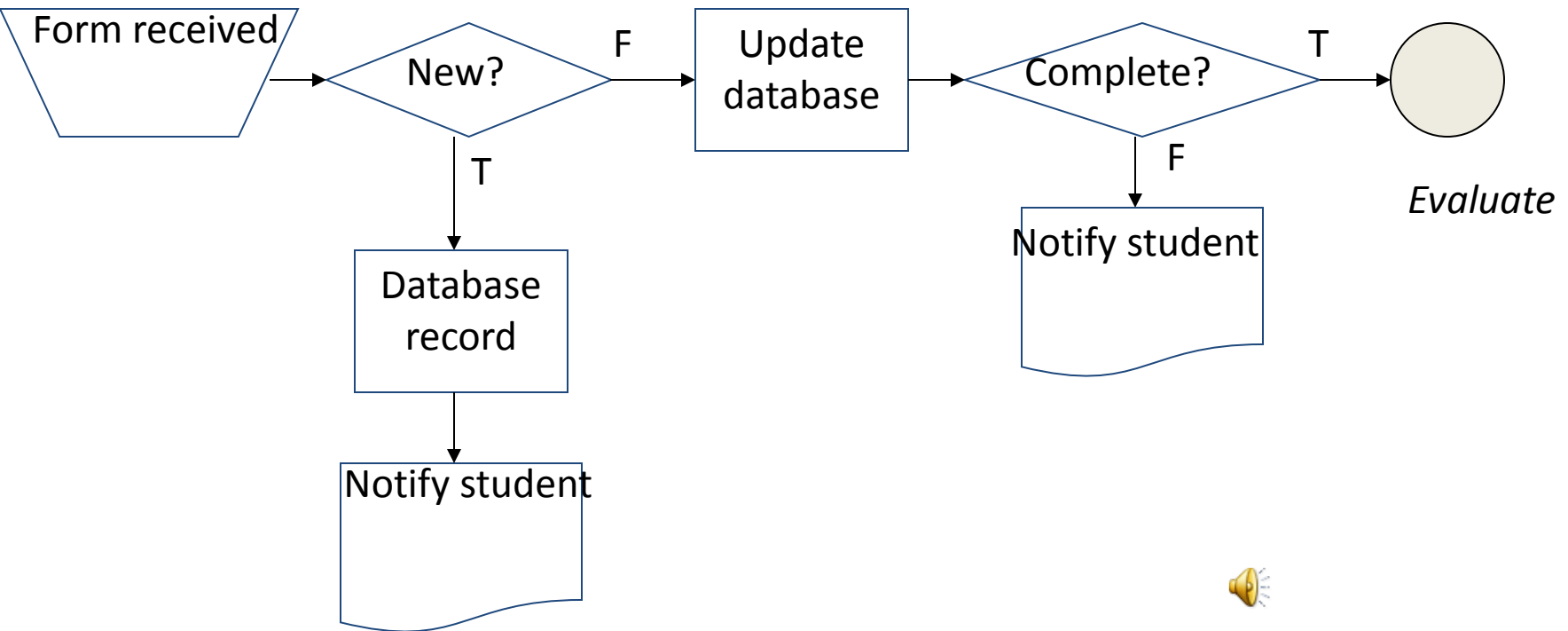
# Example: University Admissions Assemble Application Stage



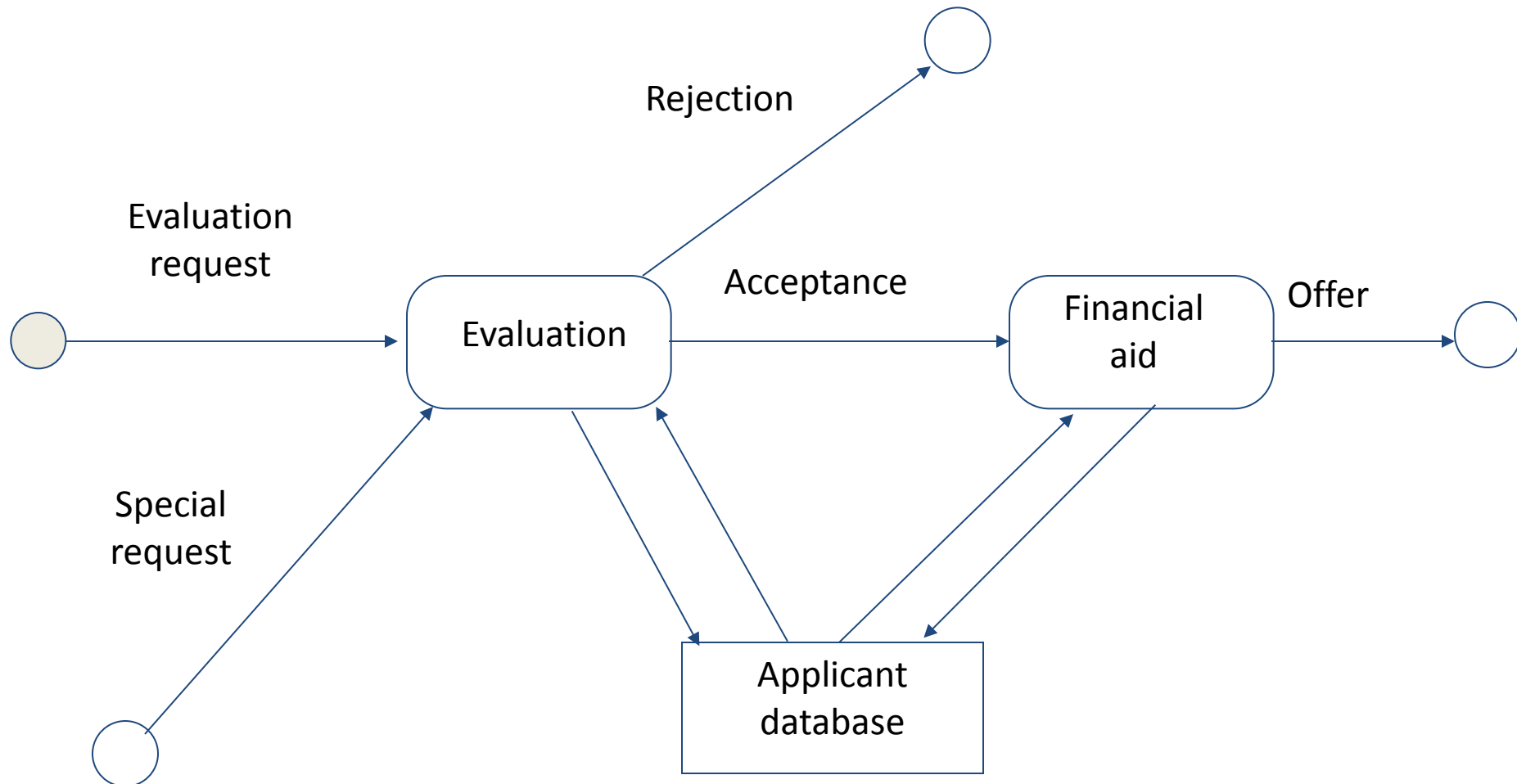
# Example: University Admissions Process Completed Application Stage



# Flowchart: University Admissions



# Example: University Admissions Process Completed Application Stage



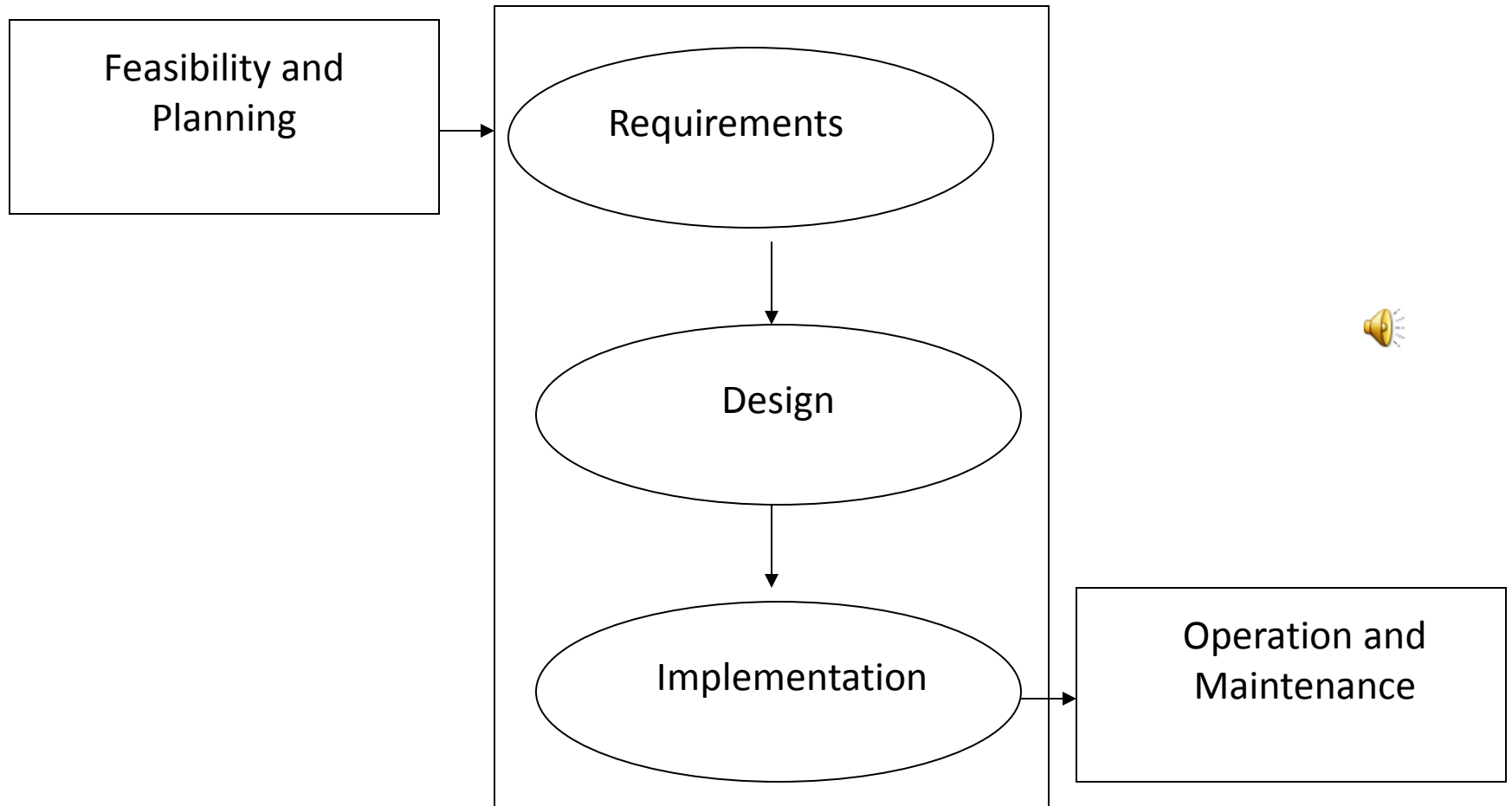
# Professional Responsibility

## Organizations put trust in software developers:

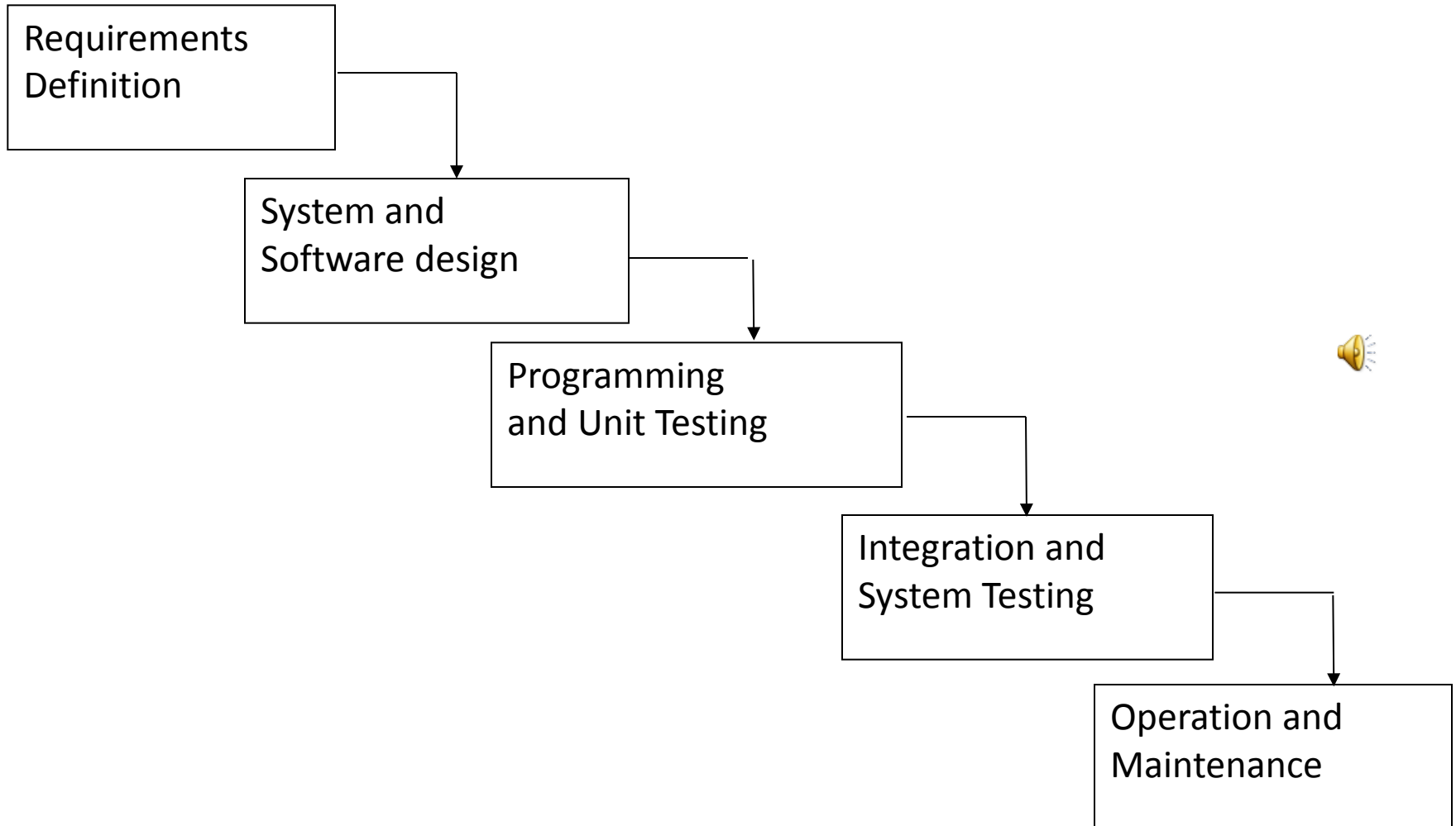
- ◆ Competence: Software that does not work effectively can destroy an organization.
- ◆ Confidentiality: Software developers and systems administrators may have access to highly confidential information (e.g., trade secrets, personal data).
- ◆ Legal environment: Software exists in a complex legal environment (e.g., intellectual property, obscenity).
- ◆ Acceptable use and misuse: Computer abuse can paralyze an organization (e.g., the Internet worm).



# The Software Process (Simplified)



# The Waterfall Model



# System and Software Design

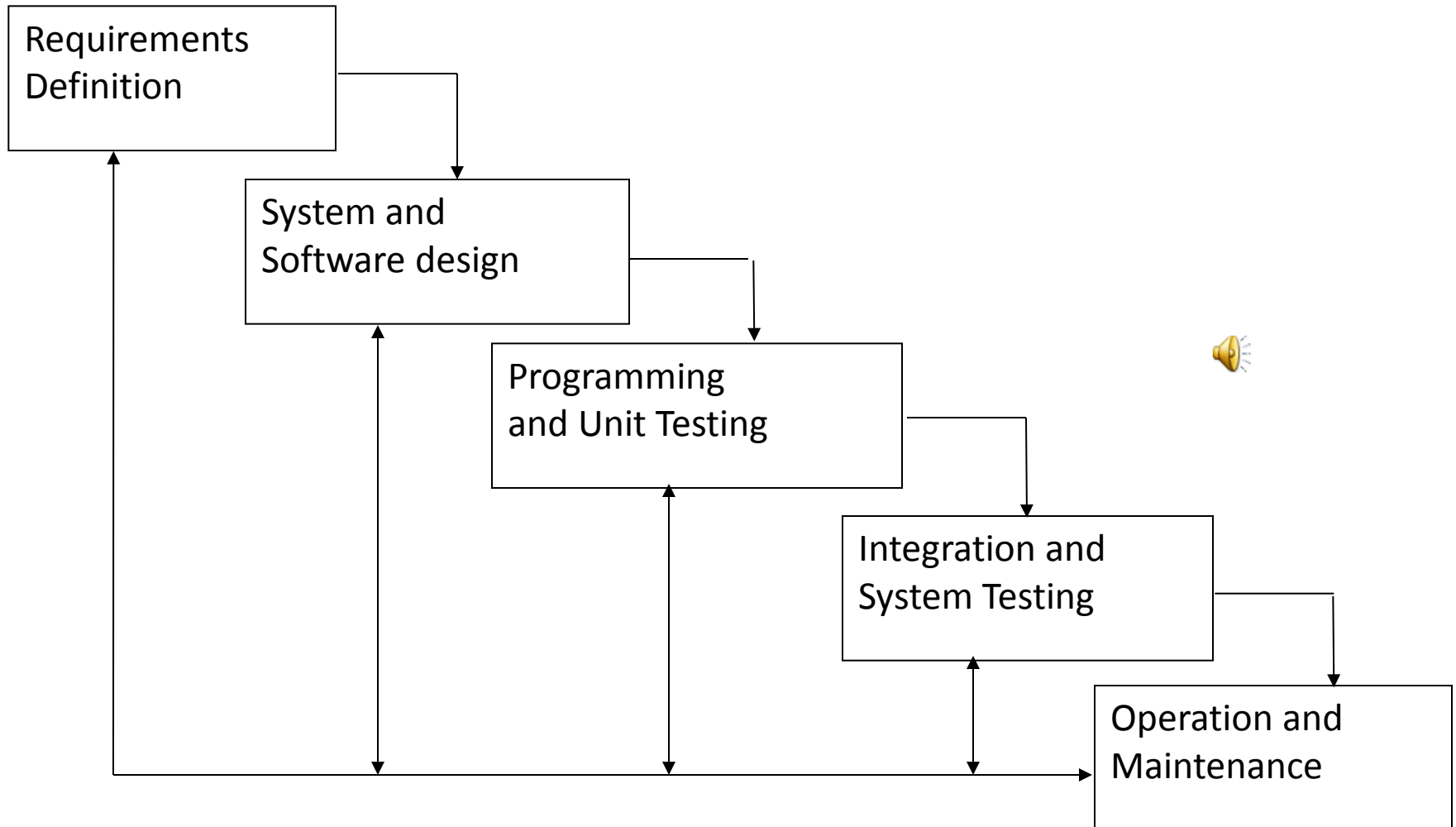
System design: Partition the requirements to hardware or software systems. Establishes an overall system architecture

Software design: Represent the software system functions in a form that can be transformed into one or more executable programs

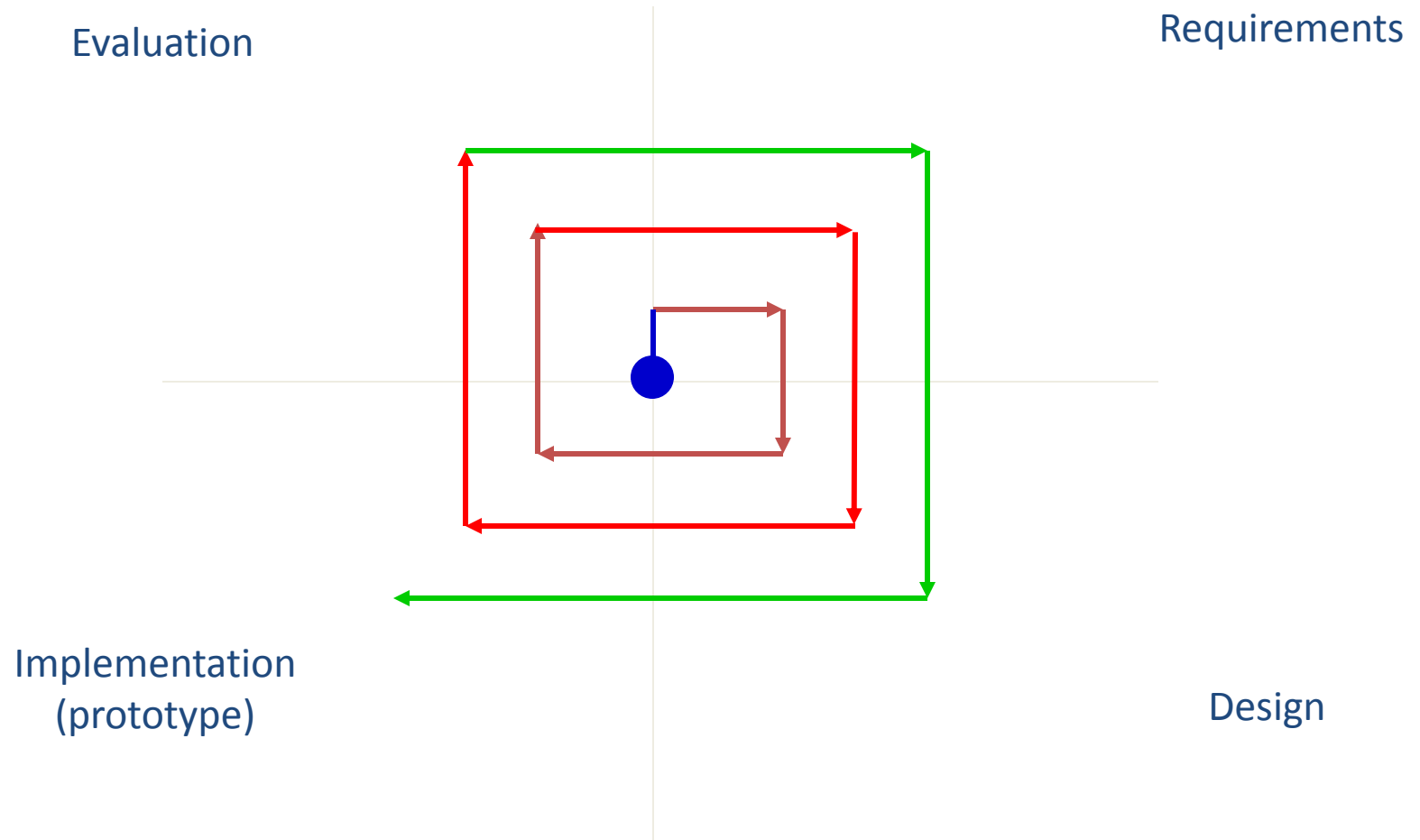
◆ Unified Modeling Language (UML)



# Feedback in the Waterfall Model

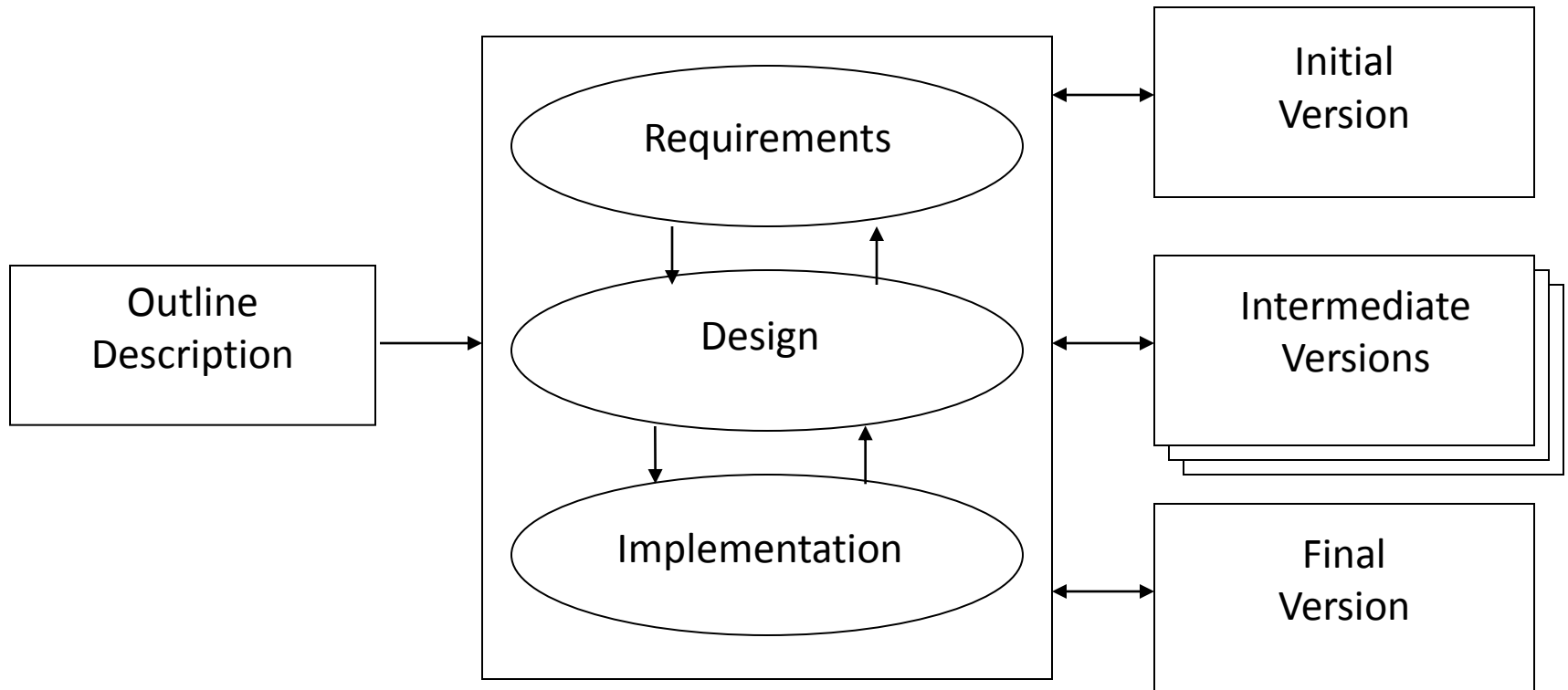


# Iterative Refinement



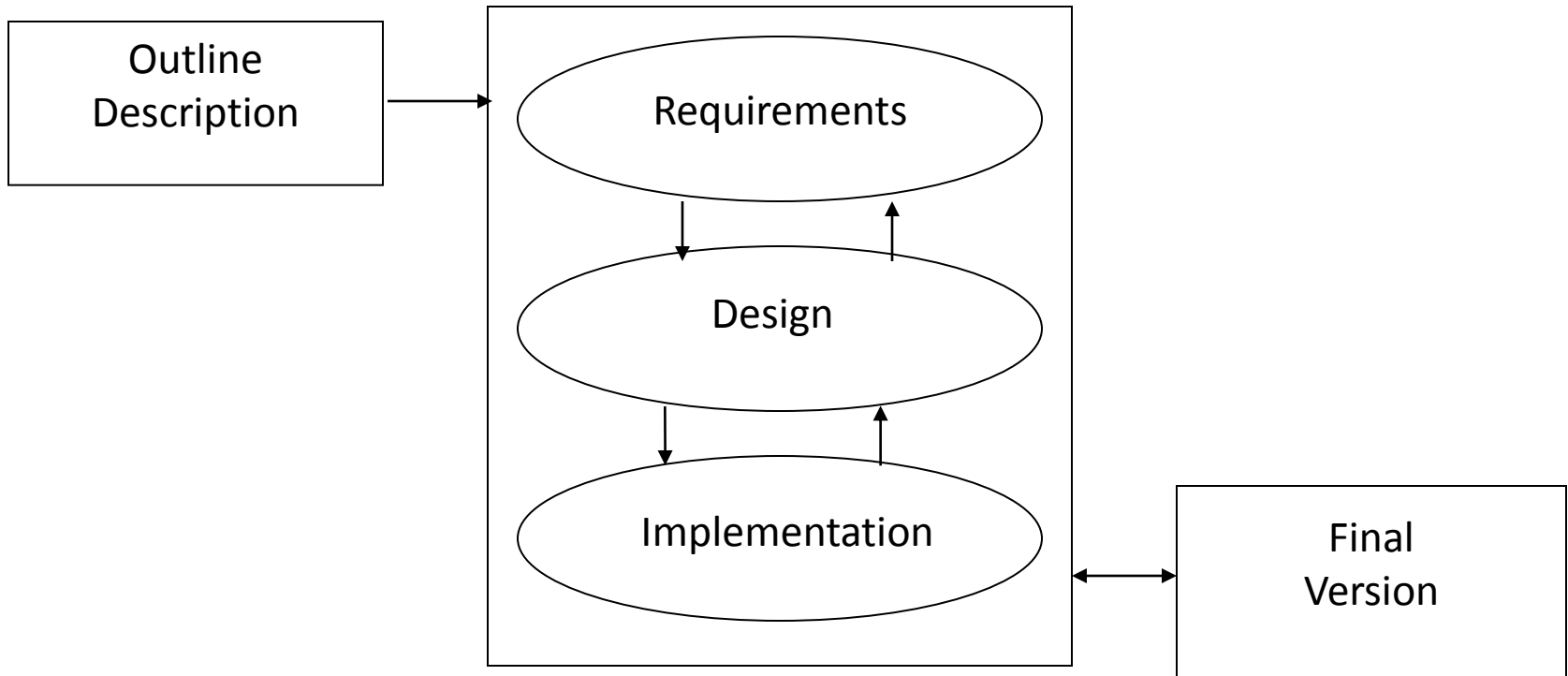
# Iterative Refinement

*Concurrent  
Activities*

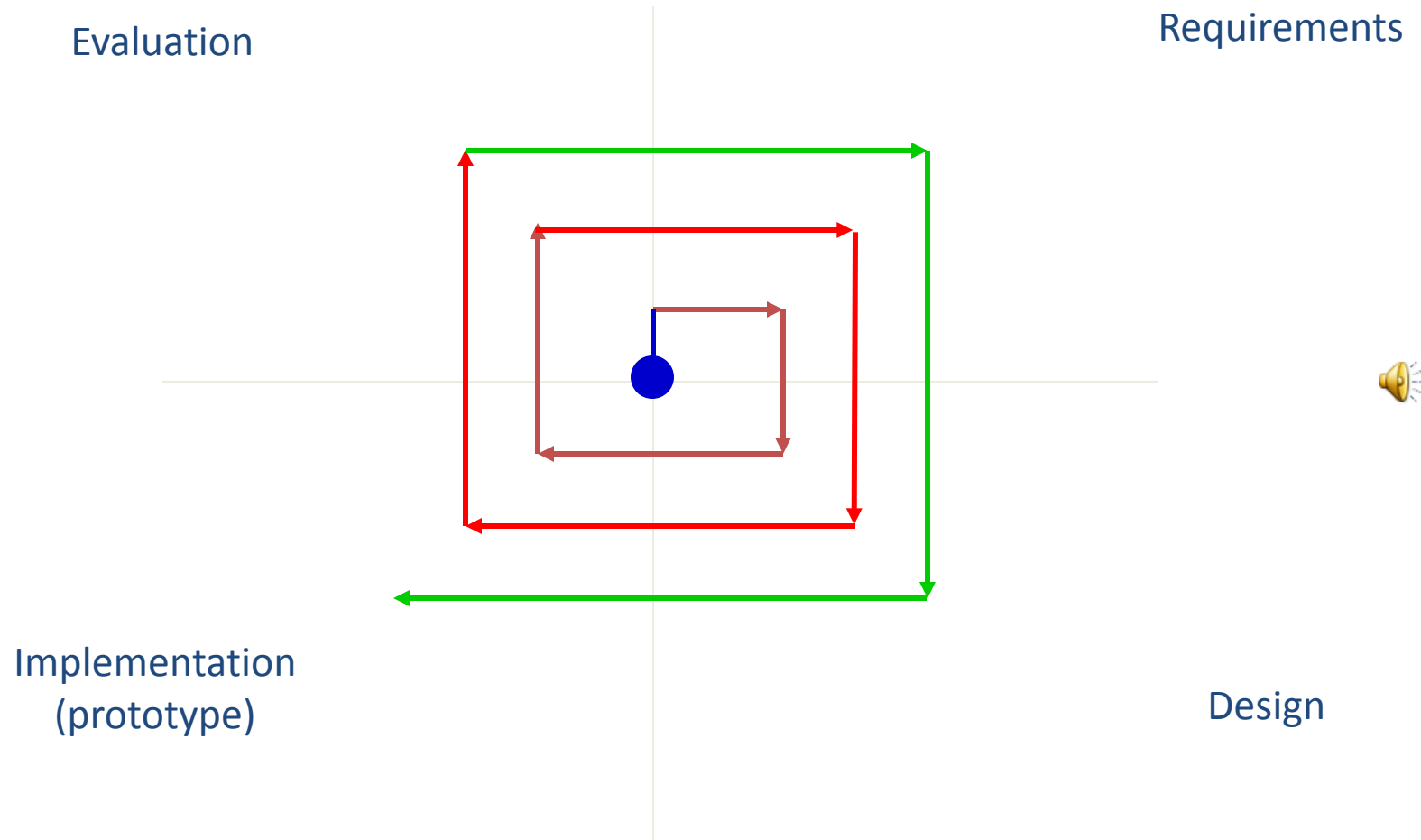


# Iterative Refinement & Software Process

*Concurrent  
Activities*

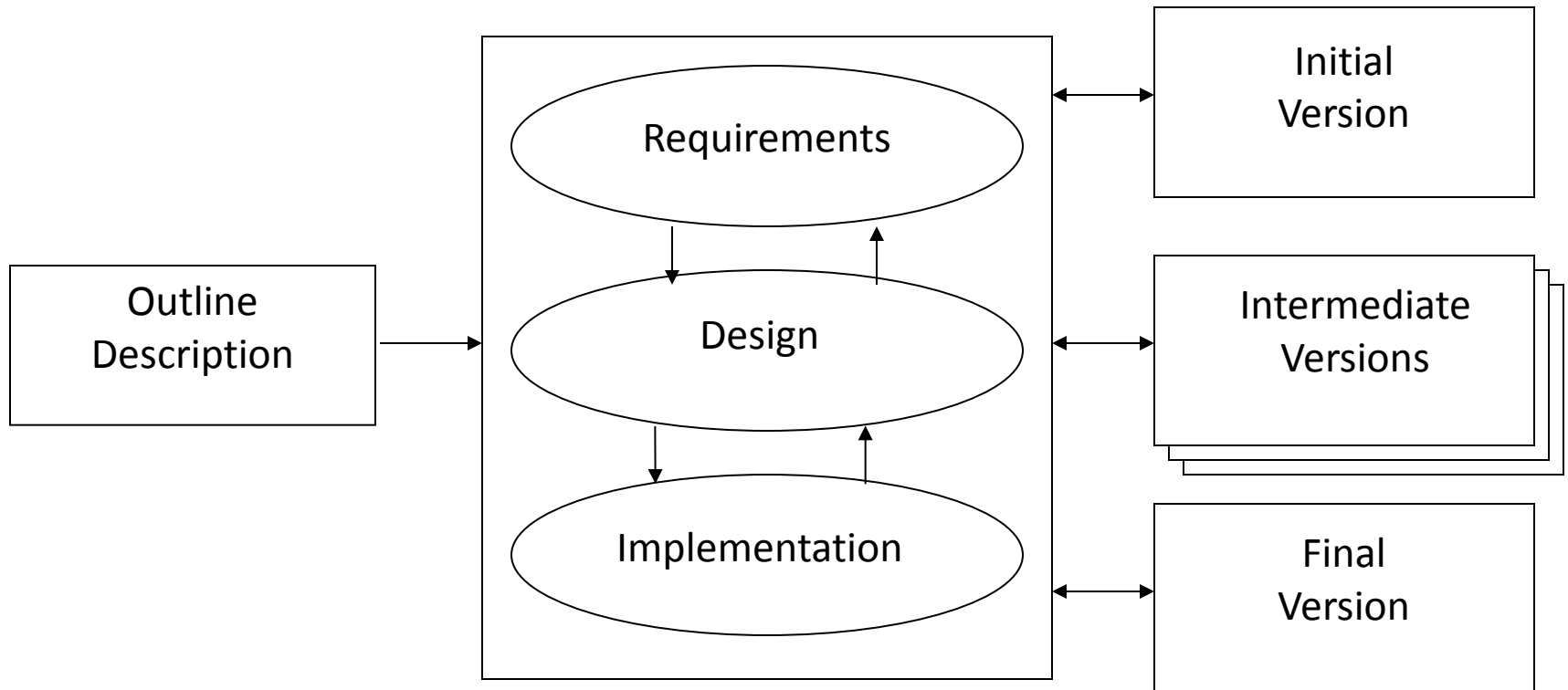


# Iterative Refinement



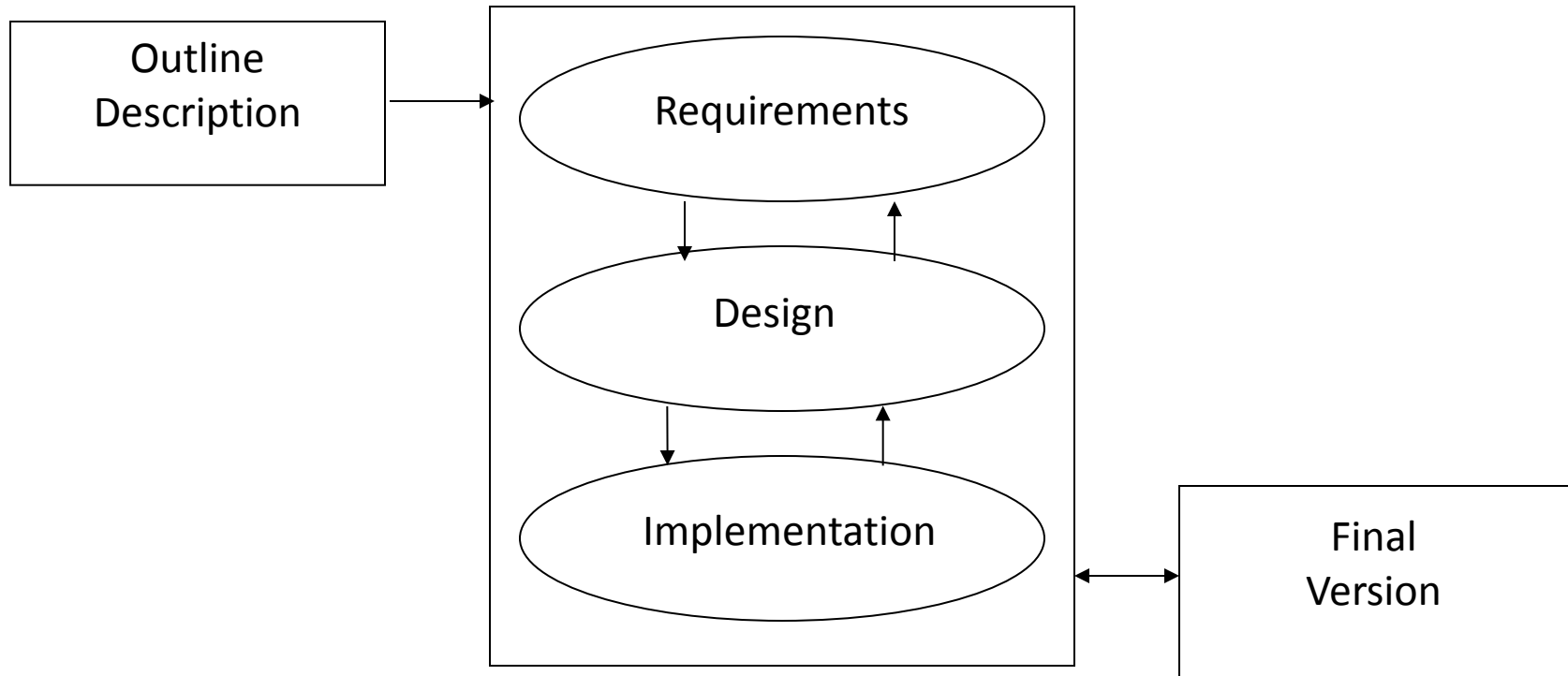
# Iterative Refinement

*Concurrent  
Activities*



# Iterative Refinement & Software Process

*Concurrent  
Activities*



# Observations about Software Processes

Completed projects should look like the Waterfall Model  
*but ...* the development process is always partly evolutionary.

Risk is lowered by:

- ◆ **Prototyping** key components
- ◆ Dividing into **phases**
- ◆ Following a **visible** software process
- ◆ Making use of reusable **components**

