IT3205: Fundamentals of Software Engineering (Compulsory)

BIT – 2nd Year

Semester 3

IT3205: Fundamentals of Software Engineering

Software Development Process Models

Duration: 8 hours

Learning Objectives

- Describe different process models used for software development
- Identify the most appropriate software process model for a given problem
- Identify how CASE tools can be used to support software process activities

Detailed Syllabus

2.1 What is a software process?

2.2 What is a software process model?

- 2.2.1 The waterfall model
- 2.2.2 Evolutionary development
- 2.2.3 Component-Based Software Engineering (CBSE)

2.3 Process Iteration

- 2.3.1 Incremental delivery
- 2.3.2 Spiral development

Detailed Syllabus

2.4 Rapid software development

- 2.4.1 Agile methods
- 2.4.2 Extreme programming
- 2.4.3 Rapid application development (RAD)
- 2.4.4 Software prototyping

2.5 Rational Unified Process (RUP)

2.6 Computer Aided Software Engineering (CASE)

- 2.6.1 Overview of CASE approach
- 2.6.2 Classification of CASE tools

2.1 WHAT IS A SOFTWARE PROCESS?

Software Process

- It is important to go through a series of steps to produce high quality software. These steps or the road map followed is called **the software process.**
- Software process is a set of ordered tasks involving activities, constraints and resources that produce a software system
- A process is important because it imposes consistency and structure on a set of activities

Software Process

- It guides our actions by allowing us to examine, understand, control and improve the activities that comprise the process
- The process of building a product is sometime called a lifecycle because it describes the life of that product from conception through to its implementation, delivery, use and maintenance

Generic activities in all software processes

- Specification
  - what the system should do and its development constraints
- Development
  - production of the software system
- Validation
  - checking that the software is what the customer wants
- Evolution
  - changing the software in response to changing demands

2.2 WHAT IS A SOFTWARE PROCESS MODEL?

Software Process Models

you need to model the process because:

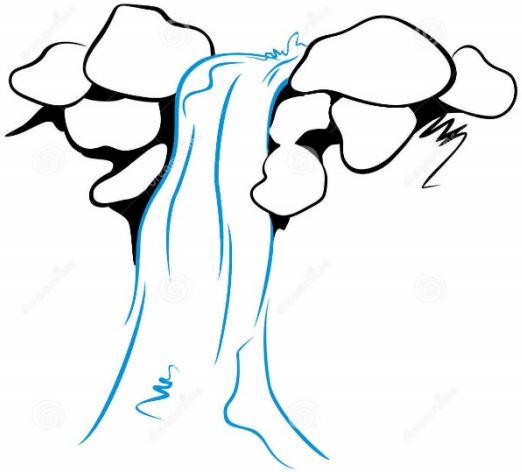
- when a team writes down a description of its development process it forms a common understanding of the activities, resources and constraints involved in software development
- creating a process model helps the team find inconsistencies, redundancies and commissions in the process, as these problems are noted and corrected the process becomes more effective

Software Process Models

you need to model the process because:

- the model reflects the goals of development and shows explicitly how the product characteristics are to be achieved
- each development is different and a process has to be tailored for different situations, the model helps people to understand these differences

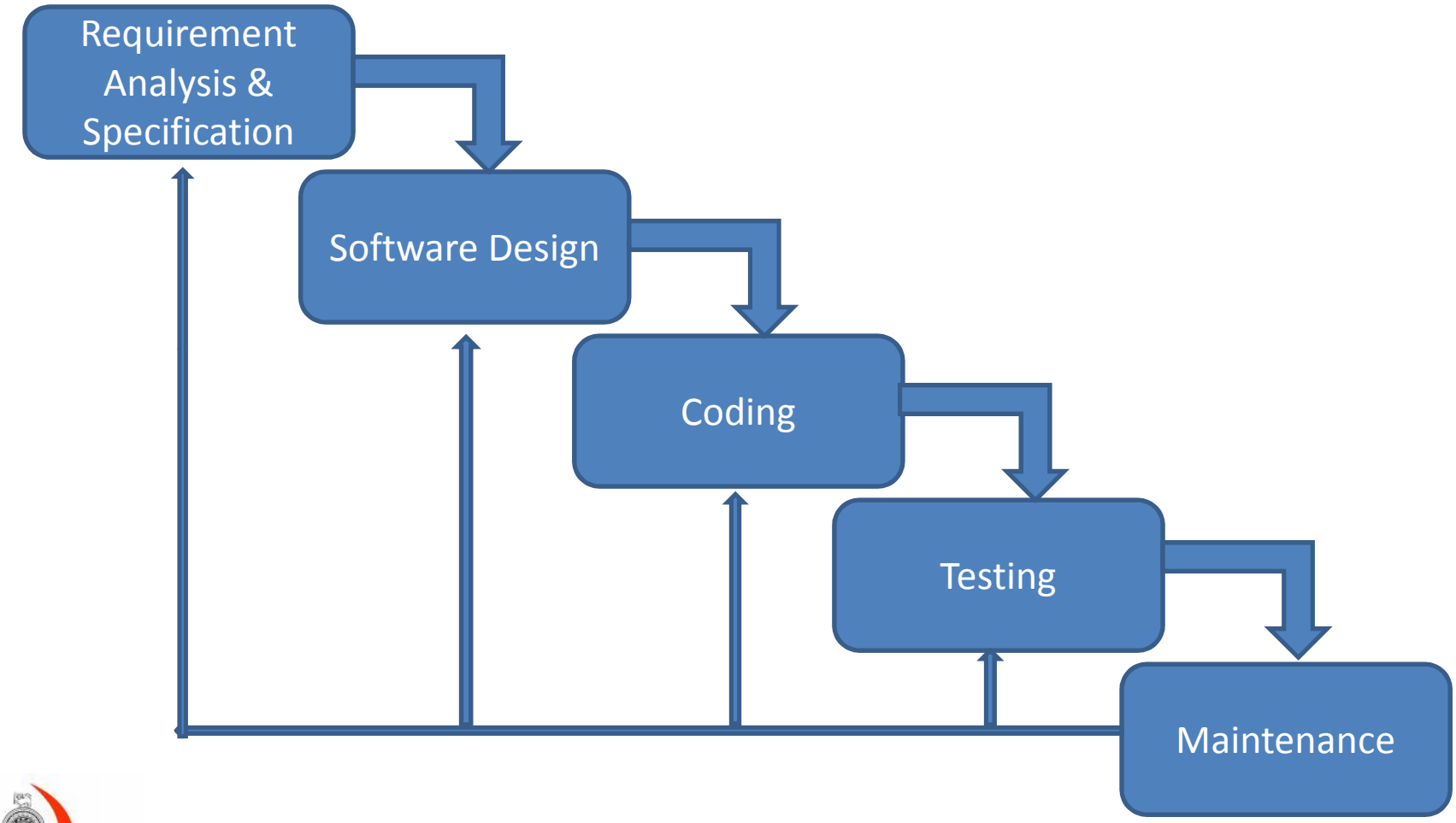
2.2.1 THE WATERFALL MODEL



Waterfall Model

- Separate and distinct phases of specification and development
- A linear sequential model

Waterfall Model Phases



Requirement Analysis & Specification

- The system's services, constraints and goals are established with the consultation with the users.
- This would include the understanding of the information domain for the software, functionality, behavior, performance, interface, security and exceptional requirements.
- This requirements are then specified in a manner which is understandable by both users and the development staff.

Software design

- The design process translates requirements into a representation of the software that can be implemented using software tools.
- The major objectives of the design process are the identification of the software components, the software architecture, interfaces, data structures and algorithms.

Coding (implementation)

- The design must be translated to a machine readable form.
- During this stage the software design is realized as a set of programs or program units.
- Programming languages or CASE tools can be used to develop software.

Testing

- The testing process must ensure that the system works correctly and satisfies the requirements specified.
- After testing, the software system is delivered to the customer.

Maintenance

- Software will undoubtedly undergo changes after it is delivered to the customer.
- Errors in the system should corrected and the system should be modified and updated to suit new user requirements.

Problems with the Waterfall Model

1. Real projects rarely follow the sequential flow that the model proposes. Although the Waterfall model can accommodate iteration, it does so indirectly.
2. It is often very difficult for the customer to state all requirements explicitly. The Waterfall model has the difficulty of accommodating the natural uncertainty that exists at the beginning of many projects.
3. The customers must have patience. A working version of the program(s) will not be available until late in the project time-span. A major blunder, if undetected until the working program is reviewed, can be disastrous.

Problems with the Waterfall Model

4. The difficulty of accommodating change after the process is underway.
5. One phase has to be complete before moving on to the next phase.
6. Few business systems have stable requirements.

Comment : The Waterfall model is suitable for projects which have clear and stable requirements.

2.2.2 EVOLUTIONARY DEVELOPMENT

Evolutionary Development

- Evolutionary development approach is typically used to develop and implement software in a evolutionary manner.
- This approach has been described by Steve McConnell as the "best practice for software development and implementation".

Evolutionary Development

- Early versions of the system are presented to the customer and the system is refined and enhanced based on customer feedback.
- The cycle continues until development time runs out (schedule constraint) or funding for development runs out (resource constraint).

Prototyping

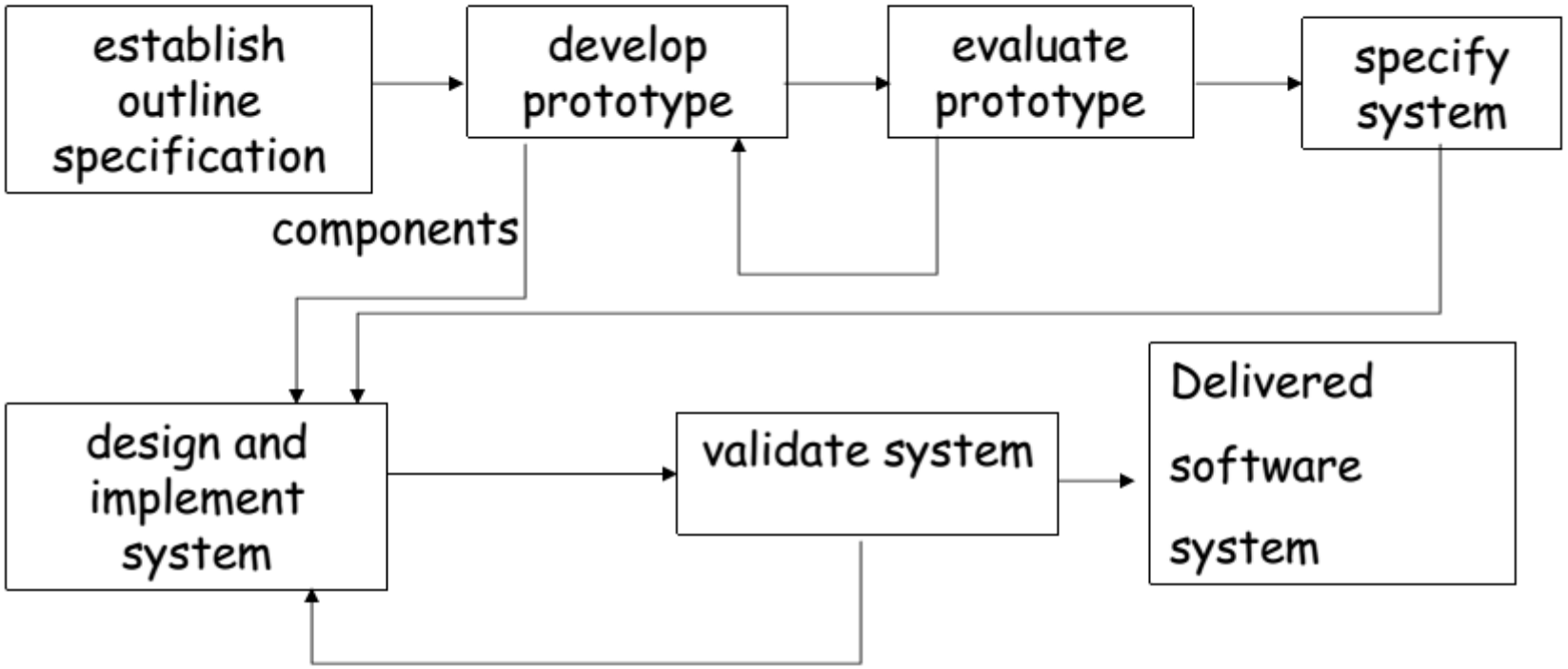
- It is very difficult for end-users to anticipate how they will use new software systems to support their work. If the system is large and complex, it is probably impossible to make this assessment before the system is built and put into use.

- A prototype (a small version of the system) can be used to clear the vague requirements. A prototype should be evaluated with the user participation.

Prototyping

- A prototype is a working model that is functionally equivalent to a component of the product.
- There are two types of Prototyping techniques
  - Throw-away Prototyping
  - Evolutionary Prototyping

Throw-away Prototyping



Throw-away Prototyping

- The objective is to understand the system requirements clearly.
- Starts with poorly understood requirements. Once the requirements are cleared, the system will be developed from the beginning.
- This model is suitable if the requirements are vague but stable.

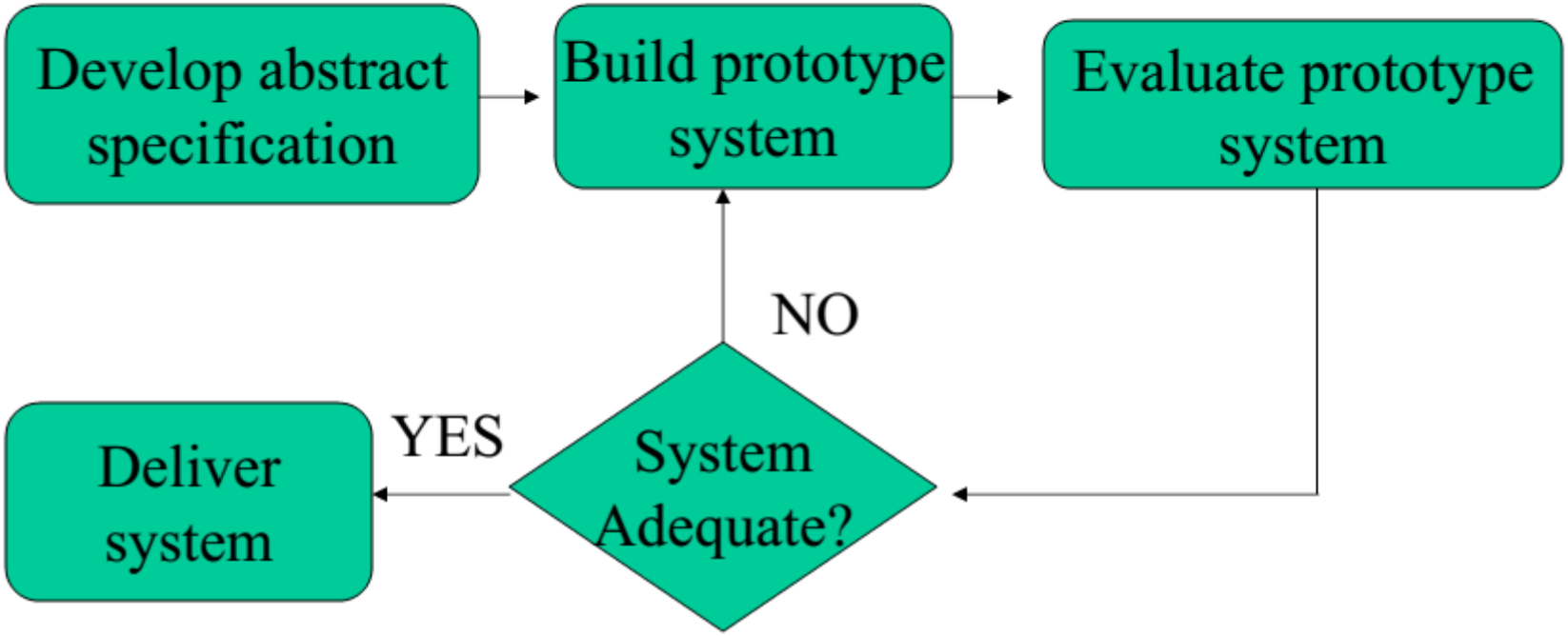
Some problems with Throw-away Prototyping

1. Important features may have been left out of the prototype to simplify rapid implementation. In fact, it may not be possible to prototype some of the most important parts of the system such as safety-critical functions.

2. An implementation has no legal standing as a contract between customer and contractor.

3. Non-functional requirements such as those concerning reliability, robustness and safety cannot be adequately tested in a prototype implementation.

Evolutionary Prototyping



Evolutionary Prototyping

- Advantages
  - Effort of prototype is not wasted
  - Faster than the Waterfall model
  - High level of user involvement from the start
  - Technical or other problems discovered early – risk reduced
  - A working system is available early in the process
  - Misunderstandings between software users and developers are exposed
  - Mainly suitable for projects with vague and unstable requirements

Evolutionary Prototyping

- Disadvantages
  - Prototype usually evolve so quickly that it is not cost-effective to produce great deal of documentation
  - Continual change tends to corrupt the structure of the prototype system. Maintenance is therefore likely to be difficult and costly
  - It is not clear how the range of skills which is normal in software engineering teams can be used effectively for this mode of development
  - Languages which are good for prototyping not always best for final product

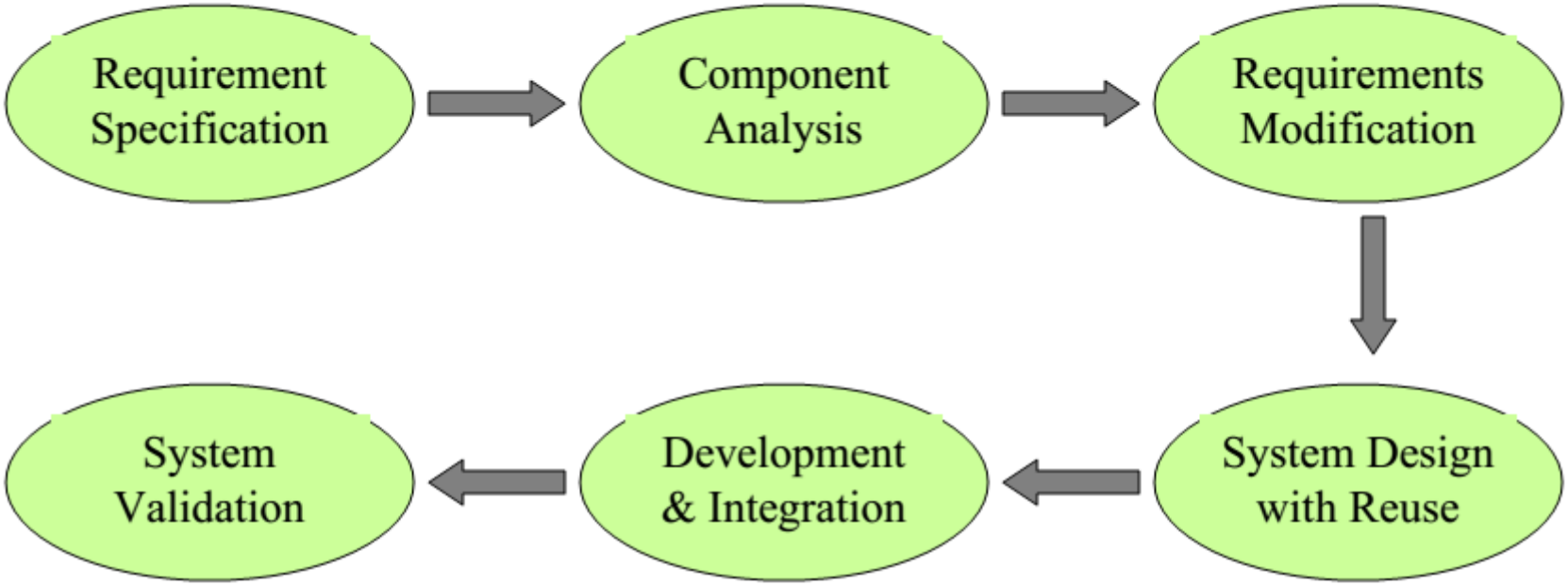
2.2.3 COMPONENT-BASED SOFTWARE ENGINEERING (CBSE)

Component-Based Software Engineering

- Emphasizes the design and construction of computer based systems using software “components”.
- The process relies on reusable software components.
- Similar to the characteristics of the spiral model.

Component-Based Software Engineering

Process



Component-Based Software Engineering

- Requirement specification and validation steps are similar to the other processes.

- **Component Analysis**
  - During this stage try to find the software components need for the implementation once the requirements are specified.
- **Requirements Modification**
  - Analyze the discovered software components to find out whether it is able to achieve the specified requirements.

Component-Based Software Engineering

- **System Design with Reuse**
  - The frame work of the system is designed to get the maximum use of discovered components. New software may have to design if the reusable components are not available.

- **Development and integration**
  - Software that cannot be discovered is developed, and the reusable components are integrated to create the new system. The integration process, may be part of the development process rather than a separate activity.

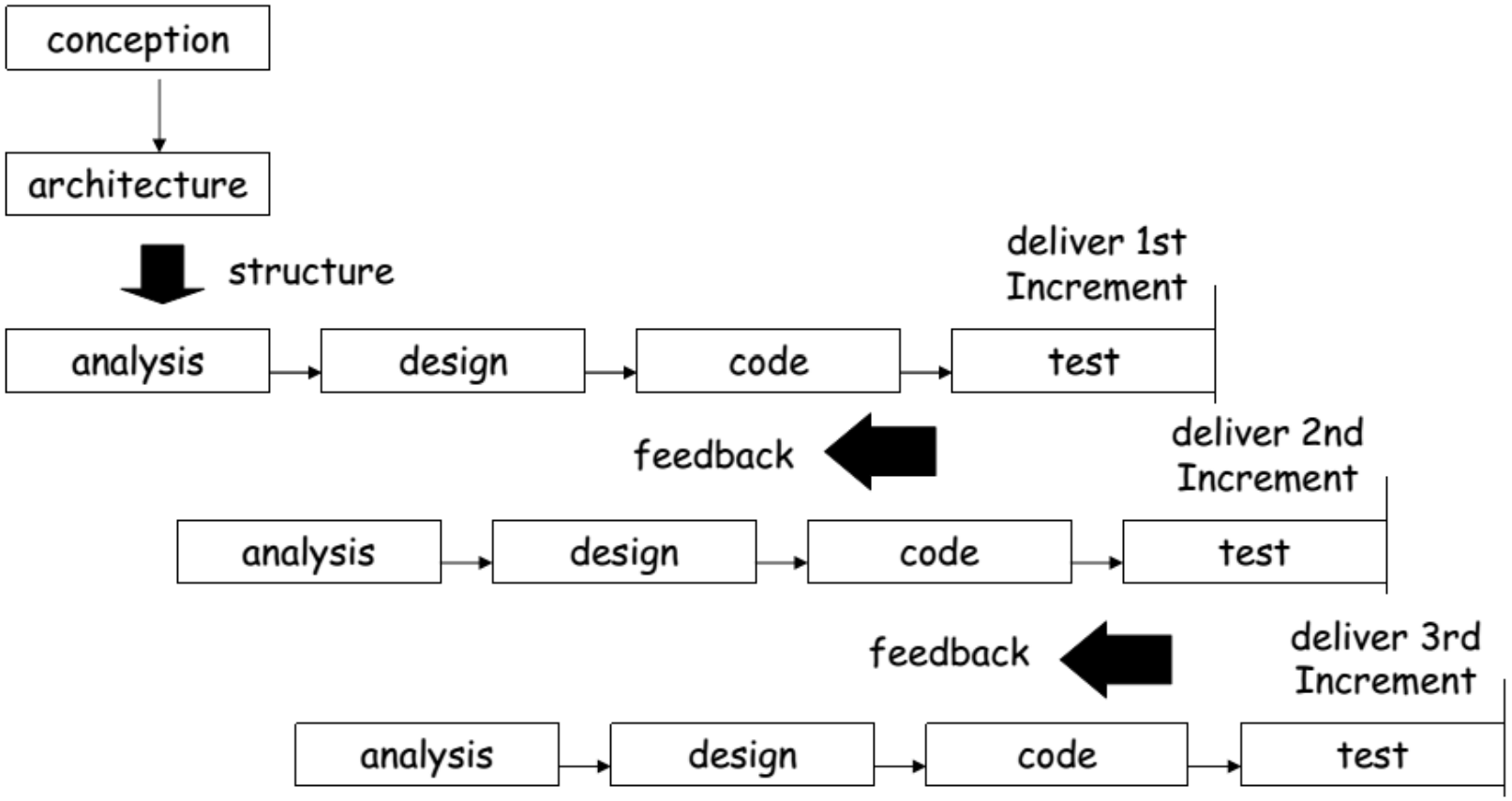
2.3 PROCESS ITERATION

Process Iteration

- A process for arriving at a decision or a desired result by repeating rounds of analysis or a cycle of operations.
- The objective is to bring the desired decision or result closer to discovery with each repetition (iteration).
- The iterative process can be used where the decision is not easily revocable or where the consequences of revocation could be costly.

2.3.1 INCREMENTAL DELIVERY

Incremental Model



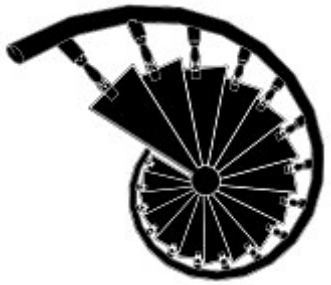
Incremental Development

- The Incremental development model involves developing the system in an incremental fashion.
- The most important part of the system is fist delivered and the other parts of the system are then delivered according to their importance.
- Incremental development avoids the problems of constant change which characterize evolutionary prototyping.

Incremental Development

- An overall system architecture is established early in the process to act as a framework.
- Incremental development is more manageable than evolutionary prototyping as the normal software process standards are followed.
- Plans and documentation must be produced.

2.3.2 SPIRAL DEVELOPMENT



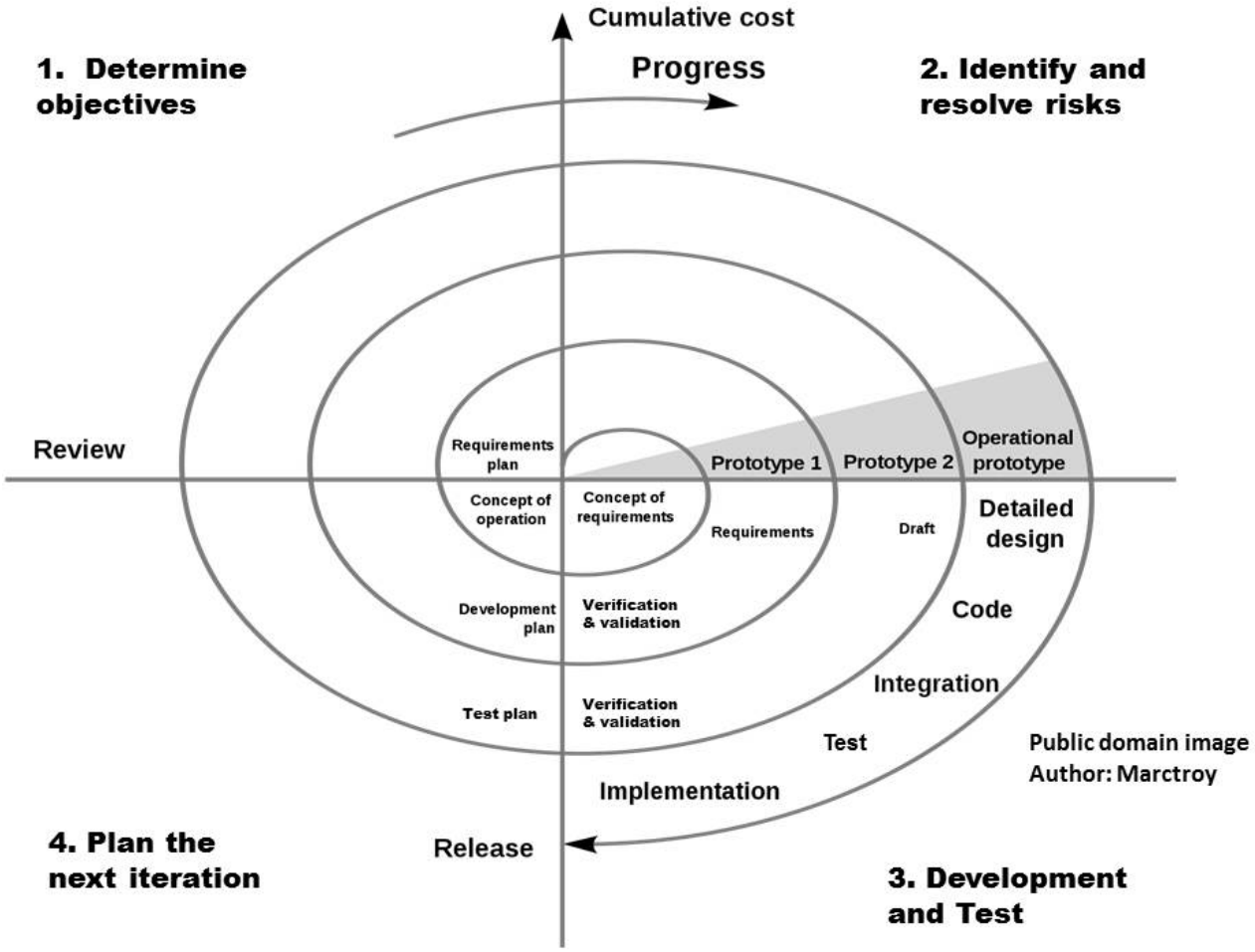
The Spiral Model

- This model is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the linear sequential model.
- Using the spiral model software is developed in a series of incremental releases. During early iterations, the incremental release might be a paper model or prototype.

The Spiral Model

- The spiral model is divided into four main task regions
  1. Determine goals, alternatives, constraints
  2. Evaluate alternatives and risks
  3. Develop and test
  4. Plan

Spiral Model



2.4 RAPID SOFTWARE DEVELOPMENT

Rapid Software Development

- Because of rapidly changing business environments, businesses have to respond to new opportunities and competition.
- This requires software and rapid development and delivery is not often the most critical requirement for software systems.
- Businesses may be willing to accept lower quality software if rapid delivery of essential functionality is possible.

Rapid Software Development

- Requirements
  - Because of the changing environment, it is often impossible to arrive at a stable, consistent set of system requirements.
  - Therefore a waterfall model of development is impractical and an approach to development based on iterative specification and delivery is the only way to deliver software quickly.

2.4.1 AGILE METHODS

Agile Process

- Agile software engineering combines a philosophy and a set of development guidelines.
- The philosophy encourages the customer satisfaction and early incremental delivery of software.
- small and highly motivated software teams, informal methods, minimal software engineering work products, and overall development simplicity.
- The development guidelines stress delivery and active and continuous communication between developers and customers.

Agile Process

- An agile process adapt incrementally. To accomplish incremental adaptation, an agile team requires customer feedback.
- An effective tool to get customer feedback is an operational prototype or a portion of an operational system.
- Software increments must be delivered in short time periods so that the adaptation keep pace with the change This iterative approach enables the customer to evaluate the software increment regularly and provide necessary feedback to the software team.

2.4.2 EXTREME PROGRAMMING

Extreme programming

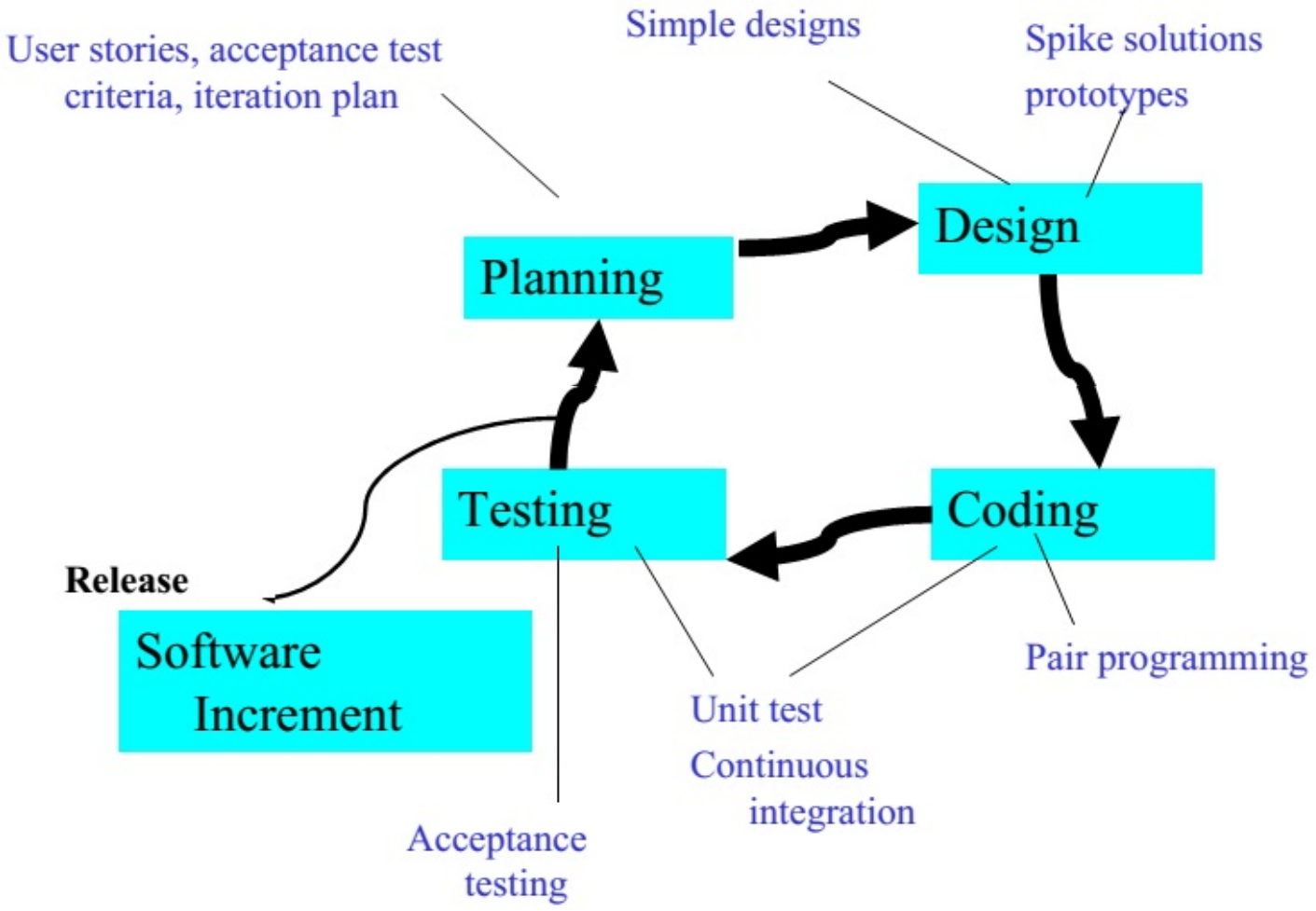
- Extreme Programming (XP) is the most widely used Agile Process model.
- XP uses an object oriented approach as its development paradigm.
- XP encompasses a set of rules and practices that occur within the context of four framework activities;

  planning, design , coding and testing.

Extreme programming

- "Extreme Programming is a discipline of software development based on values of simplicity, communication, feedback and courage"
  - Ron Jeffries

Extreme programming Process



Extreme programming Process

1. **Planning**
   - Begins with a set of stories (scenarios).
   - Each story written by the customer is assigned a value depending on its priority.
   - The members of the XP team assess each story and assigned a cost measured in development weeks.
   - If a story has more that three weeks to develop the customer is asked to split it.
   - New stories can add any time.
   - Customers and XP team work together to decide how to group stories for next increment.
   - AS development work proceeds, the customers can add stories, split stories and eliminate them.
   - The XP team then reconsiders all remaining releases and modify its plan accordingly.

Extreme programming Process

2. Design

- A simple design is preferred
- Design only consider the given stories
- Extra functionality discouraged
- Identify the object oriented classes that are relavant to the current system.
- The output of the design process is a set of CRC (Class Responsibility Collaborator) cards.

3. Coding

- XP recommends developing a series of unit tests for each of the story
- Once the code is complete, units should be unit tested.
- Pair programming – two people work together at one computer.

Extreme programming Process

4. **Testing**
   - The unit tests that has been created in the coding stage should be implemented using a framework that can be implemented.
   - This enables regression testing
   - Integration and validation can occur on a daily basis
   - This provides the XP team with a continual indication of the progress and also raise flags early if things are going wrong.
   - Acceptance tests are derived from user stories that have been implemented as parts of the software release.

Rapid Application Development

- Rapid Application Development (RAD) is both a general term used to refer to alternatives to the conventional waterfall model of software development as well as the name for James Martin's approach to rapid development.
- RAD is an incremental software development process model that emphasizes an extremely short development cycle.

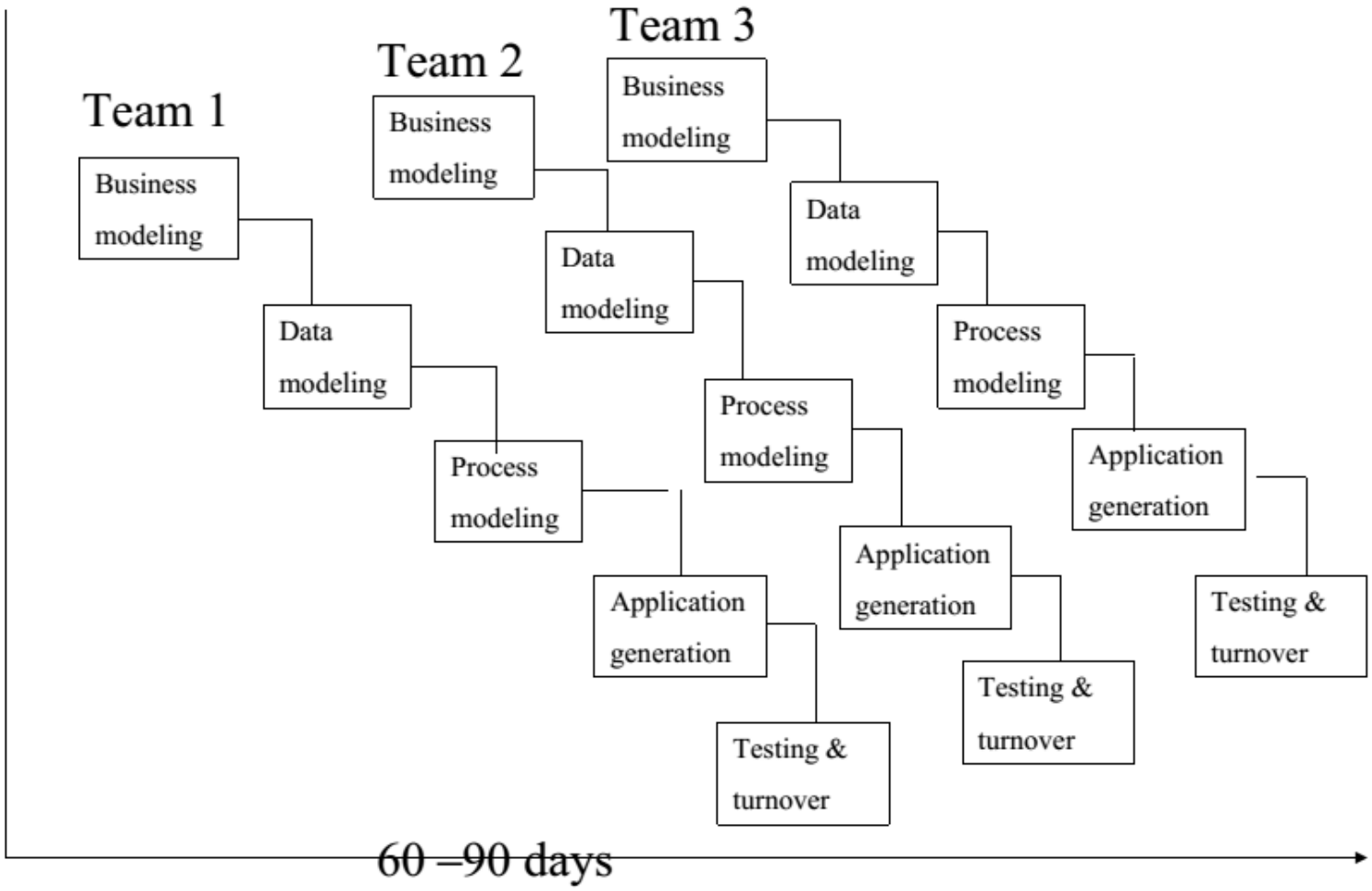
Rapid Application Development

- In general, RAD approaches to software development put less emphasis on planning tasks and more emphasis on development.
- If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a ‘fully functional system’ within very short time periods (eg. 60 to 90 days)

Rapid Application Development

- In contrast to the waterfall model, which emphasizes rigorous specification and planning, RAD approaches emphasize the necessity of adjusting requirements in reaction to knowledge gained as the project progresses.
- This causes RAD to use prototypes in addition to or even sometimes in place of design specifications. RAD approaches also emphasize a flexible process that can adapt as the project evolves rather than rigorously defining specifications and plans correctly from the start.

The RAD Model



Processes in the RAD Model

- **Business modeling**
  - The information flow in a business system considering its functionality.
- **Data Modeling**
  - The information flow defined as part of the business modeling phase is refined into a set of data objects that are needed to support the business
- **Process Modeling**
  - The data objects defined in the data modeling phase are transformed to achieve the information flow necessary to implement business functions.

Processes in the RAD Model

- **Application generation**
  - RAD assumes the use of 4GL or visual tools to generate the system using reusable components.
- **Testing and turnover**
  - New components must be tested and all interfaces must be fully exercised

Problems with the RAD model

- RAD requires sufficient human resources to create right number of RAD teams.
- RAD requires developers and customers who are committed to the rapid-fire activities necessary to get a system completed in a much abbreviated time frame.
- If a system cannot be properly modularized, building the components necessary for RAD will be problematic.
- RAD is not applicable when technical risks are high. This occurs when a new application makes heavy use of new technology or when the new software requires a high degree of interoperability with existing computer programs.

2.4.4 SOFTWARE PROTOTYPING

Software Prototyping

- The first RAD alternative was developed by Barry Boehm and was known as the spiral model.
- Boehm and other subsequent RAD approaches emphasized developing prototypes as well as or instead of rigorous design specifications.
- Prototypes had several advantages over traditional specifications:

Software Prototyping - Advantages

- Risk reduction: A prototype could test some of the most difficult potential parts of the system early on in the life-cycle. This can provide valuable information as to the feasibility of a design and can prevent the team from pursuing solutions that turn out to be too complex or time consuming to implement. This benefit of finding problems earlier in the life-cycle rather than later was a key benefit of the RAD approach. The earlier a problem can be found the cheaper it is to address.

Software Prototyping - Advantages

- Users are better at using and reacting than at creating specifications. In the waterfall model it was common for a user to sign off on a set of requirements but then when presented with an implemented system to suddenly realize that a given design lacked some critical features or was too complex. In general most users give much more useful feedback when they can experience a prototype of the running system rather than abstractly define what that system should be.

Software Prototyping - Advantages

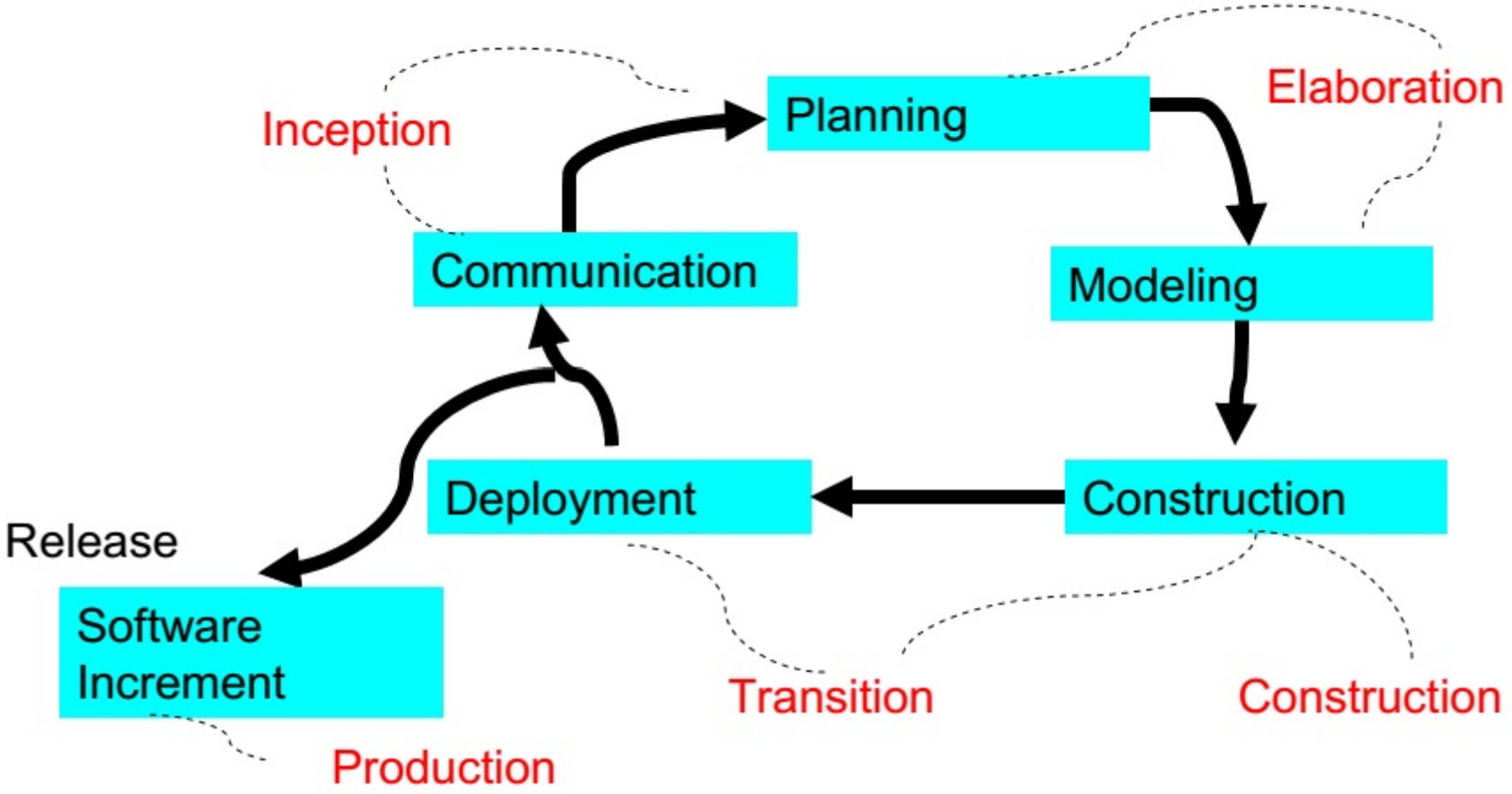
- Prototypes can be usable and can evolve into the completed product. One approach used in some RAD methodologies was to build the system as a series of prototypes that evolve from minimal functionality to moderately useful to the final completed system. The advantage of this besides the two advantages above was that the users could get useful business functionality much earlier in the process.

2.5 RATIONAL UNIFIED PROCESS (RUP)

Unified Process

- The Unified Process or Rational Unified Process (RUP) is a framework for object oriented software engineering using UML.
- This is a use-case driven, architecture centric, iterative and incremental software development model.

Unified Process



Unified Process

- **Inception Phase:**
  - The Inception Phase of UP includes both customer communication and planning activities. By collaborating with the customer and end-users, business requirements for the software are identified, a rough architecture for the system is proposed, and a plan for the iterative, incremental nature of the project is developed.

Unified Process

- **Elaboration Phase:**
  - The Elaboration Phase encompasses the planning and modeling activities of the generic process model. Elaboration refines and expands the primary use-cases that were developed as part of the inception phase and expands the architectural representation to include five different views of the software – the use-case model, the analysis model, the design model, the implementation model and the deployment model.

Unified Process

- **Construction Phase:**
  - The construction phase of the UP is identical to the construction activity defined in the generic software process. Using the architectural model as input, the construction phase develops or acquires the software components that will make each usecase operational for end-users. As components are developed unit tests are designed and executed for each component. Integration testing and acceptance testing are carried out using use-cases to derive required test cases.

Unified Process

- **Transition Phase:**
  - The Transition Phase of the UP encompasses the later stages of the generic construction activity and the first part of the generic deployment activity. Software is given to end-users for beta testing. The software team creates necessary support information (user manual, installation manual etc.). At the end of transition phase, the software increment becomes a usable software release.

Unified Process

- **Production Phase:**
  - The production phase of the UP coincides with the deployment activity of the generic process. During this phase, the on going use of the software is monitored, support for operating environment is provided, and defect reports and requests for change are submitted and evaluated.

Unified Process

- It is likely that at the same time the construction, transition and production phases are being conducted, work may have already begun on the next software increment. This means that the unified process do not occur in a sequence, but rather with staggered concurrency.

Major work products produced for each UP phases

Inception Phase	Elaboration Phase
Vision document	Use-case model
Initial use-case model	Requirements functional, non-functional
Initial risk assessment	Analysis model
Project Plan	Software architecture
Business model	Preliminary design model
Prototypes	Revised risk list, revised prototypes
Construction Phase	**Transition Phase**
Design model	Delivered software increment
Software components	Beta test reports
Integrated software increment	General user feedback
Test cases	
Test Plan and Procedures	
Support documentation	

2.6 COMPUTER AIDED SOFTWARE ENGINEERING (CASE)

2.6.1 OVERVIEW OF CASE APPROACH

Overview of CASE Approach

- Computer Aided Software Engineering (CASE) is the use of computer-based support in the software development process.
- Embedding CASE technology in software process led improvements in software quality and productivity.

CASE Automated Activities

- The development of graphical system models
- Understanding a design using a data dictionary
- Generation of user interfaces
- Program debugging
- The automated translation of programs from an old version

CASE Limited Factors

- Providing artificial intelligence technology to support design activities will be unsuccessful.

- Software engineering is a team based activity that require interactivity among team members. CASE technology does not support for this.

2.6.2 CLASSIFICATION OF CASE TOOLS

CASE Tools

- Tools used to assist the all aspects of the software development lifecycle are known as CASE Tools.

- Some typical CASE tools are:
  - Code generation tools
  - Data modeling tools
  - UML
  - Refactoring tools
  - QVT or Model transformation Tools
  - Configuration management tools including revision control

CASE Classification

- Help to understand the types of CASE tools and their role in supporting software process activities.

- CASE tools can be classified in three perspectives.
  1. ***Functional perspective:*** Classify according to the specific function.
  2. ***Process perspective:*** Classify according to the process activities that they support.
  3. ***Integration perspective:*** Classify according to how they are organized into integrated units that provide support for one or more process activities.

Functional Classification of CASE Tools

Tool Type	Examples
Planning Tools	PERT tools, ESTIMATION tools, Spreadsheets
Editing Tools	Text editors, diagram editors, word processors
Change Management Tools	Requirements traceability tools, change control systems
Configuration Management Tools	Version management systems, system building tools
Prototyping Tools	Very high level languages, User interface generators.
Method Support Tools	Design editors, data dictionaries, Code generators
Language-Processing Tools	Compilers, Interpreters
Program analysis Tools	Cross reference generators, Static analyzers, Dynamic analyzers
Testing Tools	Test data generators, File comparators
Debugging Tools	Interactive debugging systems
Documentation Tools	Page layout programs, Image editors
Re-engineering Tools	Cross reference systems, Program restructuring systems

Another Classification Dimension

- CASE tools can be classified according to the support offered to software process.
  - ***Tools :*** Support individual process tasks such as checking consistency of a design etc.
  - ***Workbenches :*** Support process phases or activities such as specification etc.
  - ***Environments :*** Support all or substantial part of the software process

Tools Workbenches and Environments