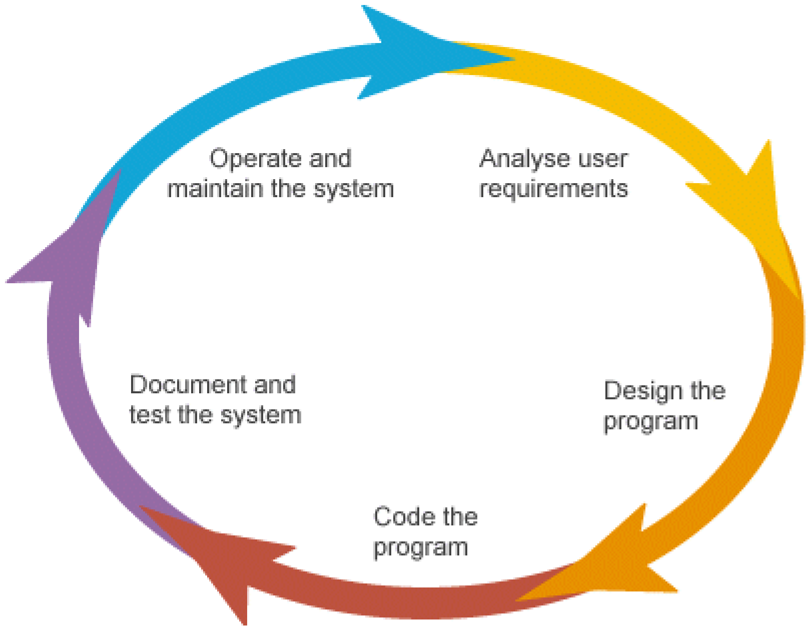
Software Development Life Cycle Models

Software Development Life Cycle Models

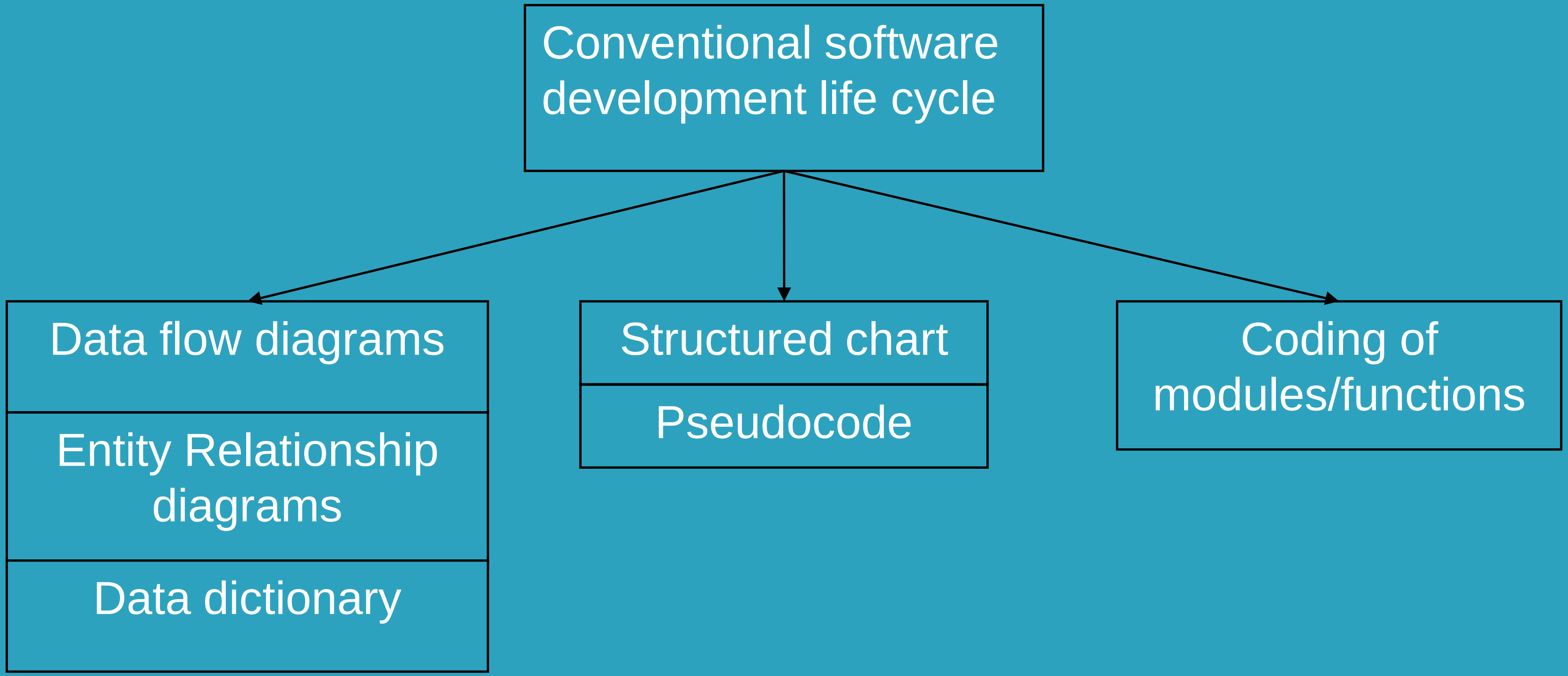
- ✓ When we decide to develop a software, many initial questions focusing on the need of the software come into our mind, such as
  - o “Would it be better to electronically generate mark sheets?,” or
  - o “Is there any market for the word processing system?,” or
  - o Is it economically viable if we electronically generate payroll of an employee?”

Software Development Life Cycle Models

- ✓ Once the needs are determined, the software goes through a series of phases of development.
- ✓ These phases include analysis, design, implementation, testing, maintenance, and retirement.
- ✓ The software remains useful between analysis and maintenance phases and ends with the retirement phase.
- ✓ The period from which a software starts till the retirement is known as life cycle of a software.


Operate and maintain the system
Analyse user requirements
Design the program
Code the program
Document and test the system

Conventional Software Life Cycle Models

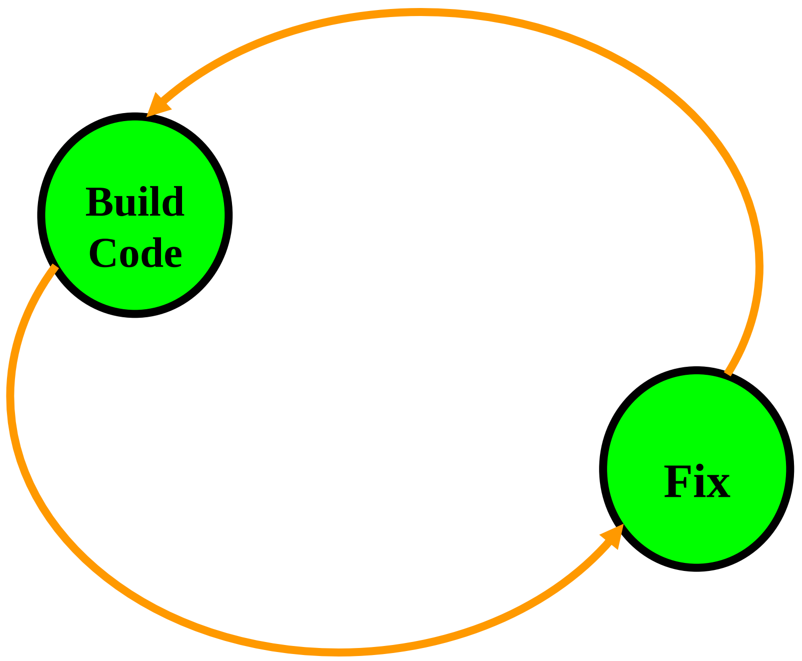

Conventional software development life cycle
Data flow diagrams
Entity Relationship diagrams
Data dictionary
Structured chart
Pseudocode
Coding of modules/functions

Build and Fix Model

- ❖ Product is constructed without specifications or any attempt at design

- ❖ Adhoc approach and not well defined

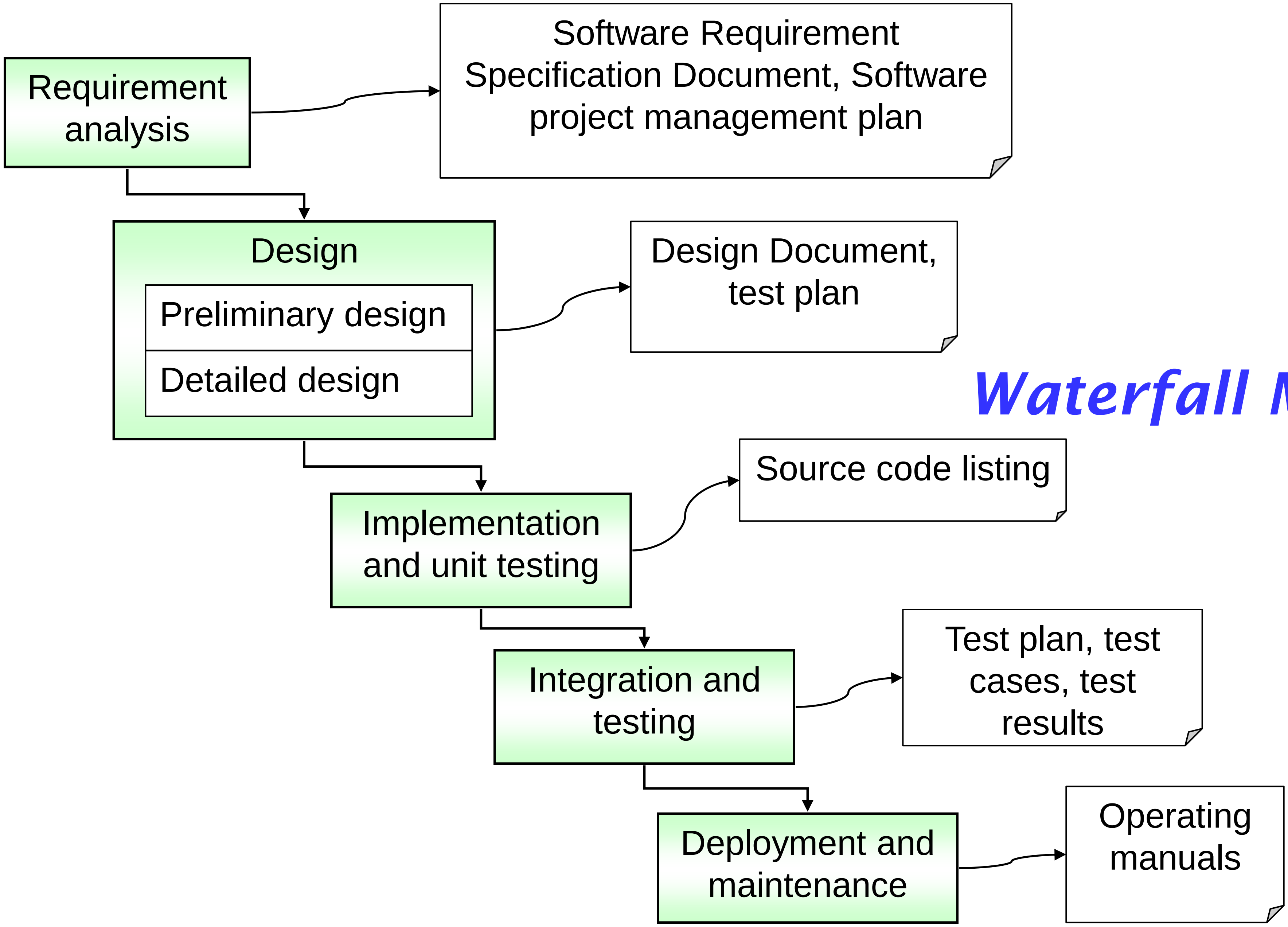
- ❖ Simple two phase model



Build and Fix Model

- Suitable for small programming exercises of 100 or 200 lines
- Unsatisfactory for software for any reasonable size
- Code soon becomes unfixable & unenhanceable
- No room for structured design
- Maintenance is practically not possible

Waterfall Model


Requirement analysis
Software Requirement Specification Document, Software project management plan
Design
Preliminary design
Detailed design
Design Document, test plan
Implementation and unit testing
Source code listing
Integration and testing
Test plan, test cases, test results
Deployment and maintenance
Operating manuals

Waterfall Model

- ✓ The phases are completed in sequential order and do not overlap with each other.
- ✓ Each phase defines its own set of functions that are distinct from those of the other phases.
- ✓ Each phase should be completed before the commencement of the next phase.

Waterfall Model

- ✓ The advantages of waterfall model include its simplicity, ease of understandability, and distinct phases with their own set of functions.
- ✓ It is highly suitable for the projects where requirements are completely known in the beginning of the project.

Waterfall Model

Disadvantages

- ✓ Large number of documents are produced. The customer may not understand the formal terminology and notations used in these documents. The software is built on the basis of written document that the customer may have partially understood.
- ✓ It freezes the requirement at the end of requirement analysis phase. The requirements may not be well understood by the customer and the developer at the beginning. Thus, it is unrealistic to determine all the requirements at the start of the project itself.

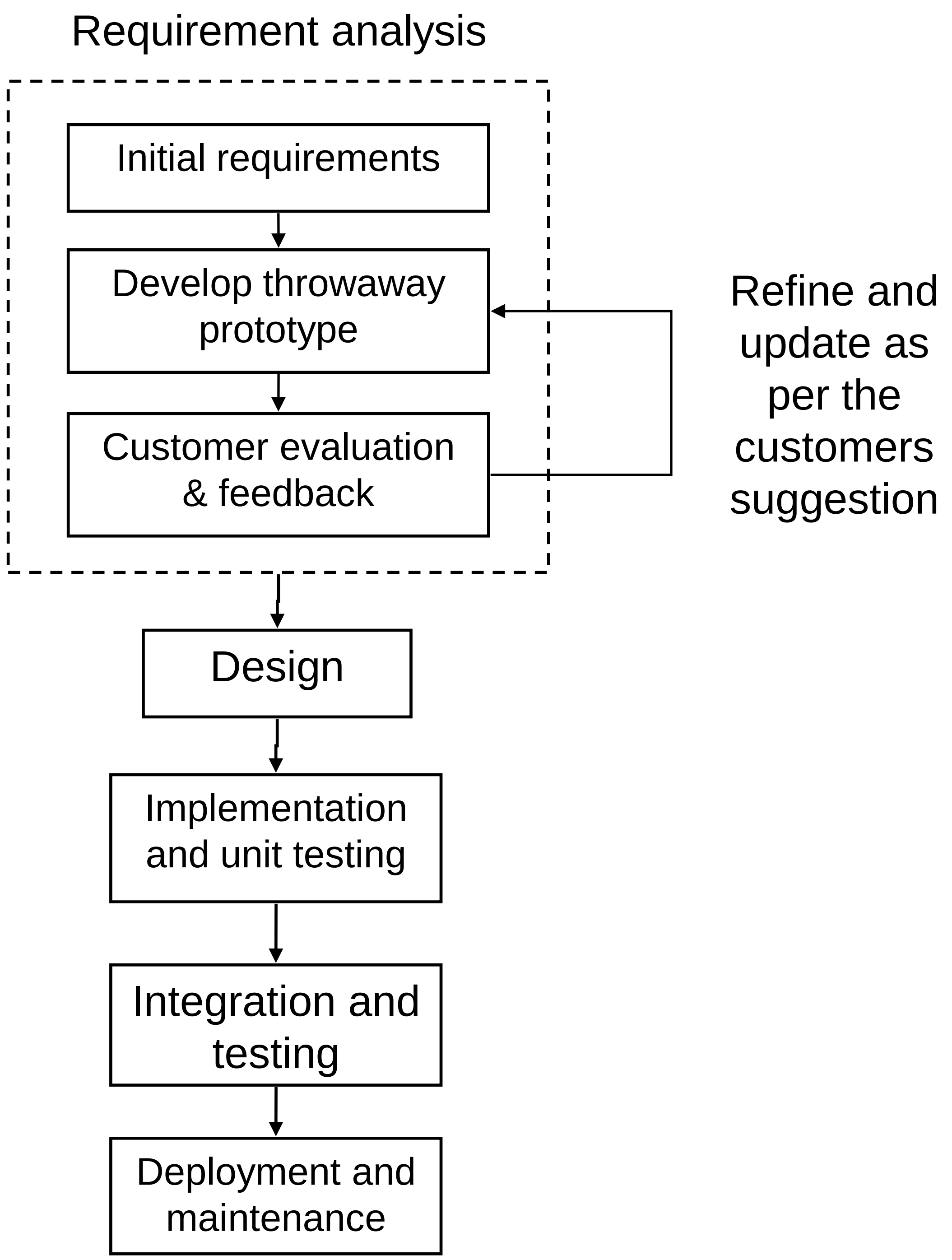
Waterfall Model

- ✓ Customer does not get the opportunity to see the working product until late after the implementation and testing is completed. The customer may find the software different than what they actually wanted
- ✓ It may take years to complete the project and at that point technology may become old and obsolete or customer’s expectations may change.
- ✓ Testing the system as a whole becomes difficult.
- ✓ Real-life projects are rarely sequential.
- ✓ This approach is particularly not useful in the case of interactive user application.

Prototyping Model

- ✓ In prototyping model, a working prototype is constructed in order to determine and understand the customer needs.
- ✓ The customer evaluates this prototype and then the final SRS document is prepared on the basis of refined requirements.

Prototyping Model


Requirement analysis
Initial requirements
Develop throwaway prototype
Customer evaluation & feedback
Refine and update as per the customers suggestion
Design
Implementation and unit testing
Integration and testing
Deployment and maintenance

Prototyping Model

- ✓ A throwaway prototype is developed which undergoes design, coding and testing but these phases are not carried out thoroughly and rigorously.
- ✓ The throwaway prototype is discarded once the requirements are determined and the final system is developed from the scratch.
- ✓ This prototype is generally built quickly to give the customer the feel of the system.

Prototyping Model

The advantages of the prototyping model are:

Stable requirements: Requirements are stabilized by refining the prototype again and again based on the customer's feedback. Thus, the probability of change in the requirements after developing the final SRS will be minimized.

High quality system: The quality of the system will be high as the developers and customers have gained lot of experience while constructing the prototype.

Low cost: The cost of the actual system will be reduced.

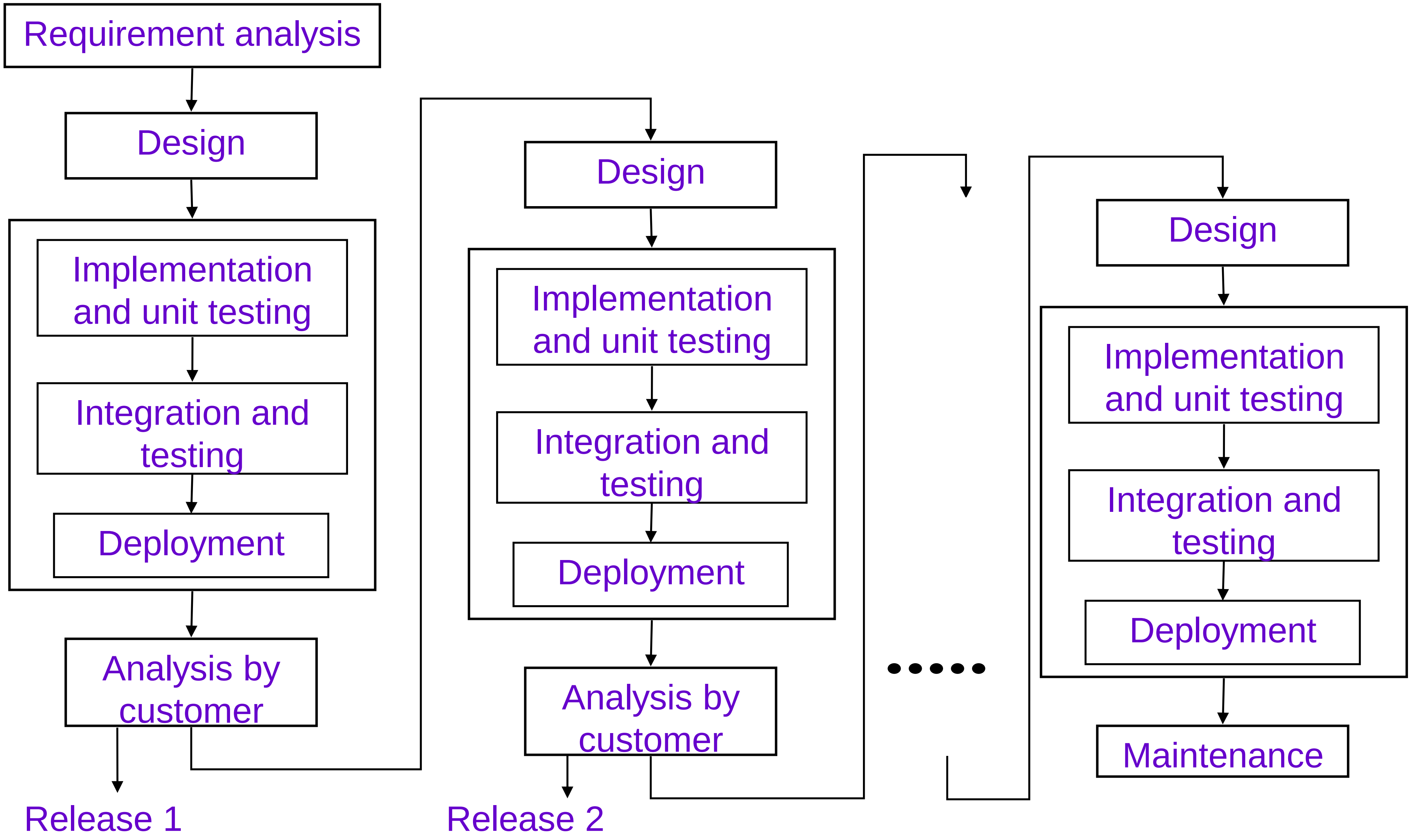
Iterative Enhancement Model

- ✓ In waterfall model the product is delivered after the completion of implementation and testing phases.
- ✓ This problem of waterfall model is overcome in the iterative enhancement model.
- ✓ Iterative enhancement model combines the advantages of both waterfall model and prototyping model.

Iterative Enhancement Model

- ✓ The phases in iterative enhancement model are same as in waterfall model but they are carried out in many cycles.
- ✓ A usable product is delivered at the end of each cycle. This allows obtaining the customer’s feedback after each cycle.
- ✓ As the product is developed in increments, the testing of the product also becomes easy.

Iterative Enhancement Model



Iterative Enhancement Model

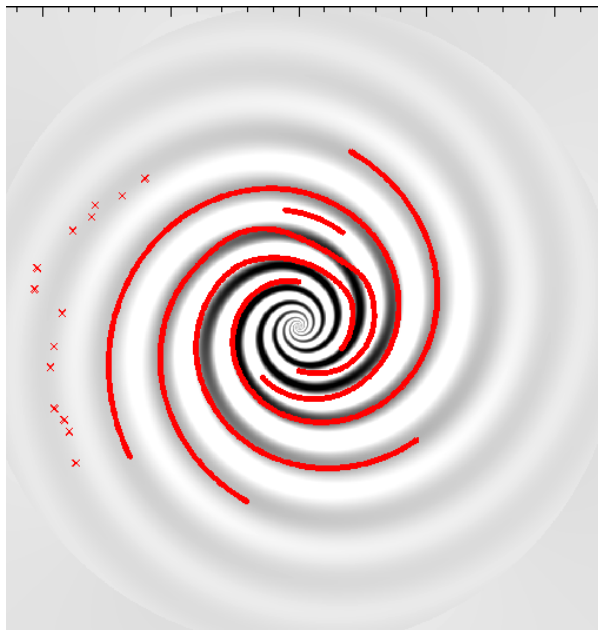
- The iterative enhancement model broadly consists of three iterative phases:
  - design,
  - implementation and
  - analysis by customer.
- After requirement analysis, these phases are carried out repeatedly until complete functionality is delivered to the customer.

Iterative Enhancement Model

- The iterative enhancement model allows to accommodate changing requirements and adds new features to the software during each phase.
- Unlike waterfall model, the iterative process carried through various cycles permits the customer to see the progress of the software continuously.
- In iterative enhancement model, later software increments may require modifications in earlier ones which may increase costs.

Spiral Model

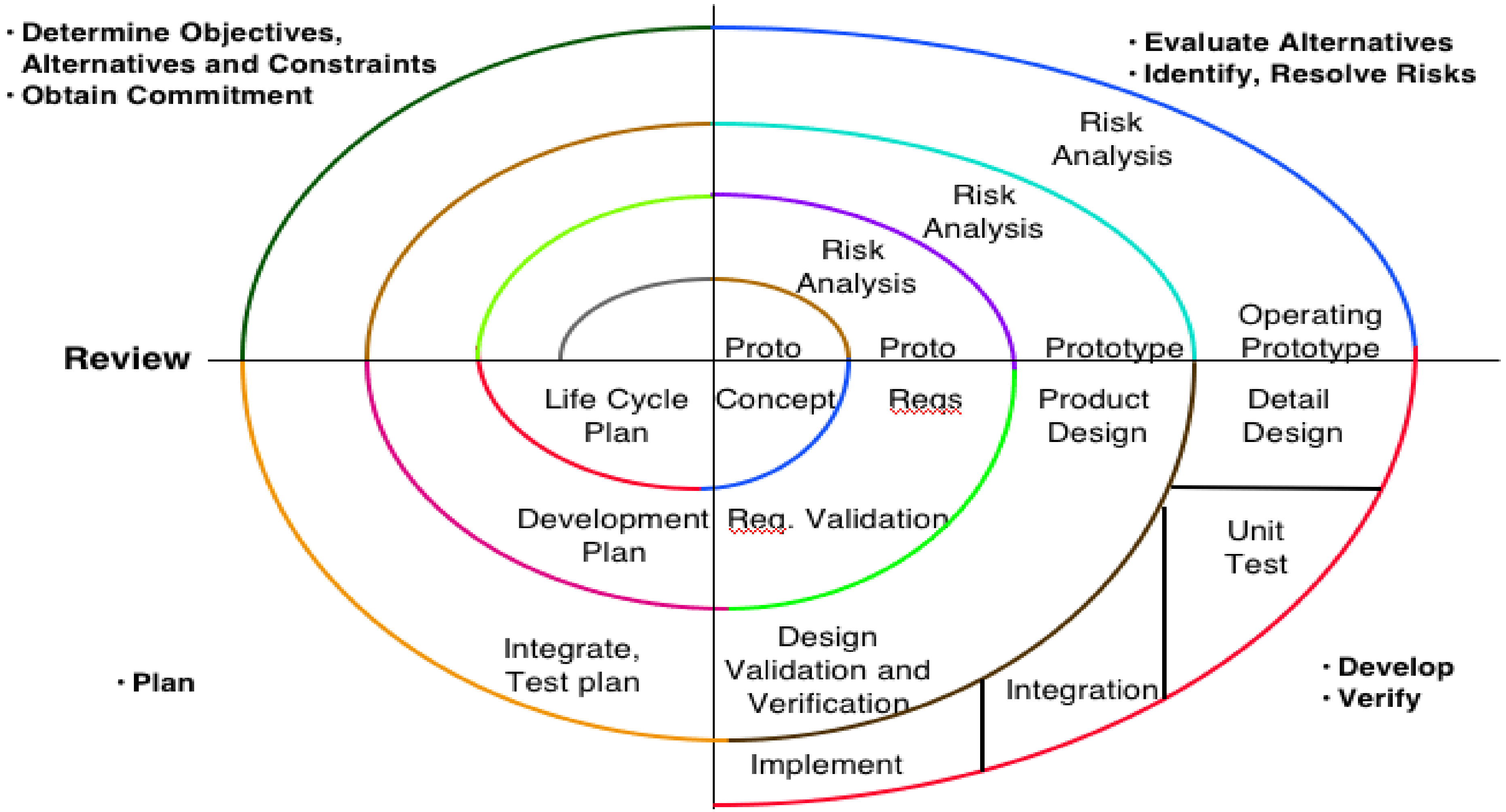
- Spiral model is developed by Barry W. Boehm in 1986.
- The spiral model is a risk driven model that deals with uncertainty of the outcome of an activity.
- Risk measures the combined effect of probability of occurrence of failure and impact of that failure on the software operation.



Spiral Model

- The radial dimension shows the cumulative cost of the project and the angular dimension shows the progress made in the completion of the final software.
- The spiral model is divided into four phases:
  - Determine objectives, alternatives, constraints,
  - evaluate alternatives, identify, resolve risks, develop,
  - verify next-level product, and
  - plan next phases.

Spiral Model



Spiral Model

- Each cycle of the spiral model begins with
  - identification of objectives (functional and non functional requirements etc),
  - determination of alternative means of implementing the specified part of the product (design 1, design 2, reuse, buy, component etc) and
  - identification of constraints imposed on the completion of the portion of the product (resources, cost, schedule etc).

Spiral Model

Round 0: Feasibility study

As recommended by Boehm, initial round 0 must be carried out in order to determine the feasibility of the system in terms of resources, cost, schedule and productivity.

Round 1: Concept of operation

The output of this round is a basic requirement and life cycle plan for implementation of concepts emerged till now.

Round 2: Top-level requirements analysis

In this round the requirements are specified and validated following risk-driven approach. The output produced is the development plan.

Spiral Model

Round 3: Software design

A preliminary design is created, verified and validated. Integration and test plan is produced after the completion of round 3.

Round 4: Design, implementation and testing

The round 4 constructs detailed deign of the product. This includes coding, unit testing, integration testing and acceptance testing.

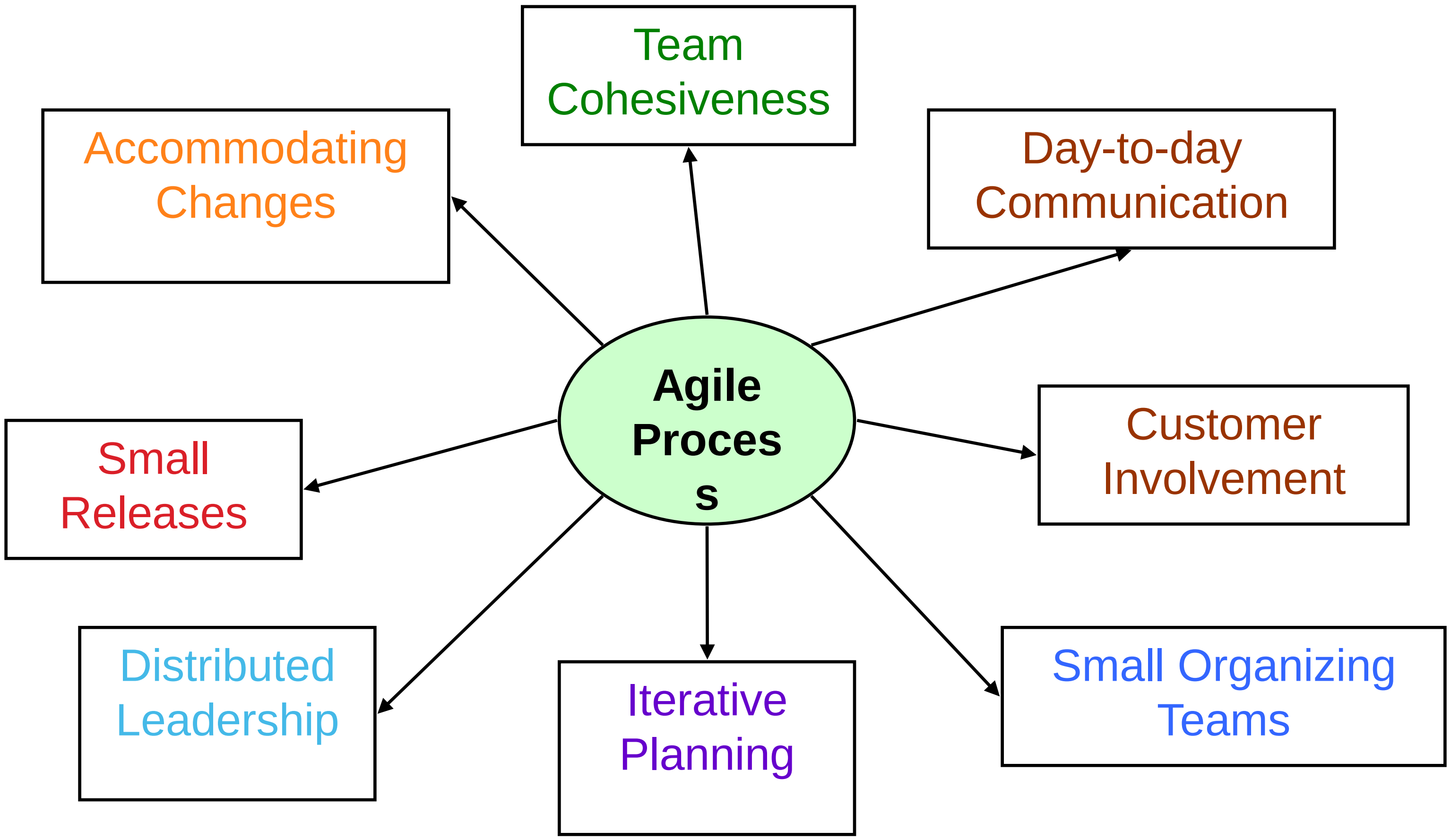
Spiral Model

- The spiral model incorporates validation and verification steps in order to reduce risks and defects in early phases of software development.
- Thus, it embeds software quality in the life cycle.
- The spiral model has some disadvantages that include
  - lack of expertise to determine and resolve risks and it is applicable to only large sized projects.
  - Sometimes cost of performing risk analysis may overweigh the returns of the spiral model.

Extreme Programming

- Extreme programming (XP) project started in middle of 1990s.
- XP is based on the principles of agile processes.
- Agile means the ability to move **quickly** and **easily**.

Extreme Programming



Extreme Programming

The rules followed in the agile processes are given below:

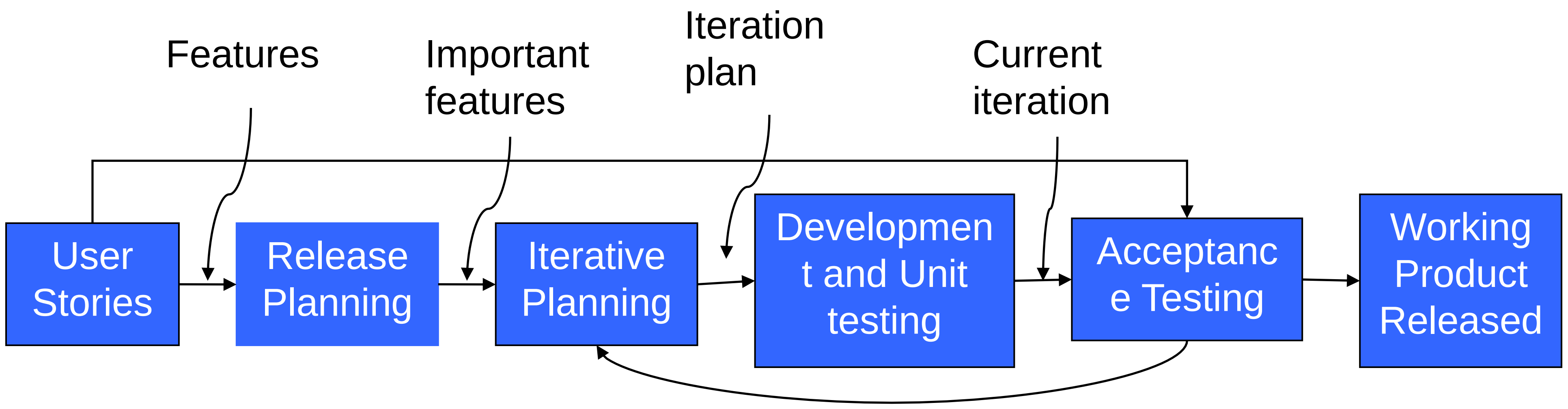
- Team cohesiveness and the way in which a team works is given more importance than any process.
- Customer (a domain expert) is always a part of the team and makes decisions related to deadlines.
- Requirement changes are accepted unlike the traditional approaches where the requirement specification was assumed to be rigid and complete.
- A working software is produced very early and shown to the customer within few weeks of development.

Extreme Programming

The rules followed in the agile processes are given below:

- The progress is measured by how much working software has been built rather by documents.
- Iterative planning is done instead of iterative development followed in traditional models. Plans are changed based on the learned processes.
- Distributed leadership is a key feature of agile processes. Decisions are not taken by only the project manager as done by traditional strategies of team build up.

Extreme Programming

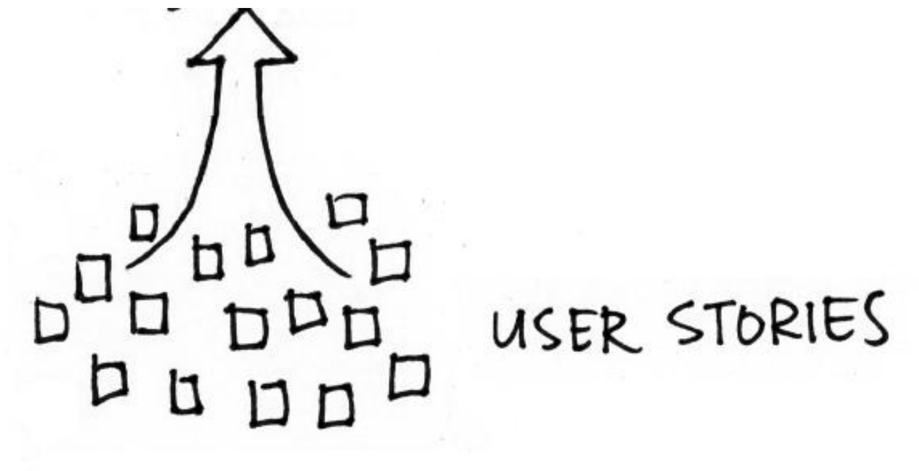


A typical life cycle of XP

Extreme Programming

User stories:

- XP begins with collection of requirements termed as "user stories" from the customer.
- These stories are short and written by the customer(s).
- User stories differ with traditional requirement specification in the sense that they only provide details for making an estimate of how much time will it take to implement a user story.



Extreme Programming

Release planning:

- In this phase the developers estimate the time to implement the user stories and customer selects the order in which the stories will be implemented.
- The developer may do a quick exploration (spike) of a user story in order to estimate its cost.

Extreme Programming

$$\begin{tikzcd}
 & \text{User stories} \arrow[d] & \\
 & \text{Estimate} \arrow[dl] \arrow[dr] & \\
\text{Divide a story} \arrow[dr] & & \text{Spike a story} \arrow[dl] \\
 & \text{Extract important stories} &
\end{tikzcd}$$

Extreme Programming

Iteration planning:

- ❑ In iteration planning user stories to be implemented in current iteration are broken into tasks and these tasks are assigned to the developers (selected by their importance).
- ❑ A working product is produced after each iteration, rather than producing a software at the end of life cycle as done in traditional approaches.
- ❑ After the iteration planning, the acceptance test cases are generated from the user stories.

Extreme Programming

Development and unit tests:

- ❑ In this phase the most important tasks chosen by the customers are implemented. In order to increase software quality, pair programming is used in which two people work together on the same computer.
- ❑ Refactoring is carried out throughout the development cycle by removing obsolete designs, unused functionality and redundancy. Automated unit tests are created before the source code is developed.
- ❑ Unit tests also enable to identify the need of refactoring.
- ❑ After each change the developer verifies through the unit tests any change in functionality.
- ❑ Unit testing enables effective functionality and regression testing so that refactoring process is carried out efficiently.

Extreme Programming

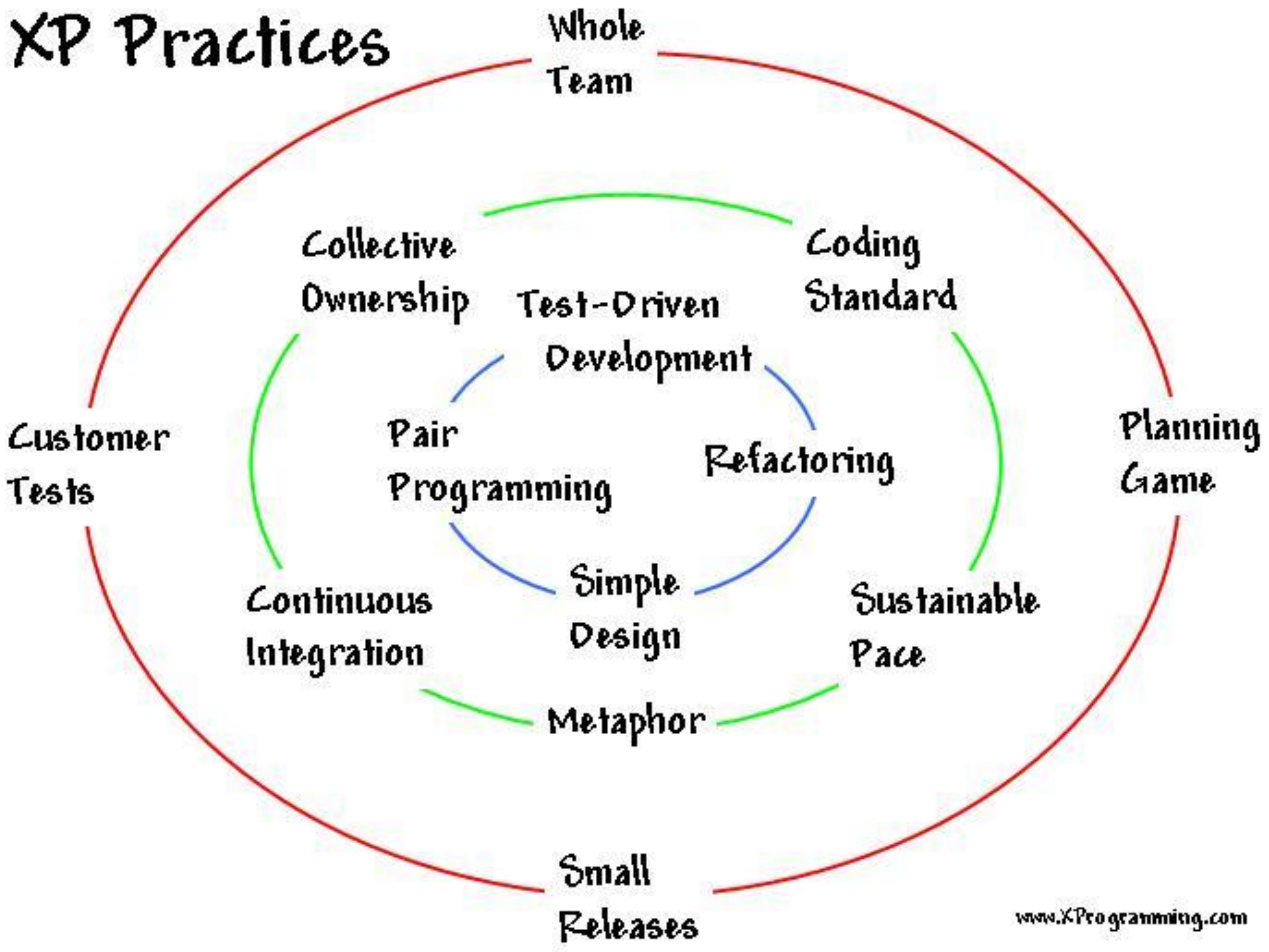
Acceptance testing:

- Acceptance test plans and test cases are created from user stories.
- These tests are carried out to ensure that requirements given by the customer are met and the software developed in current iteration is acceptable.
- During each development iteration, the acceptance test plans and test cases are prepared from the user stories. Acceptance test cases are created using black box testing techniques.

Extreme Programming

- **Working product released:** After acceptance testing the decision about the release of the product is left to the customer. There are frequent releases in which a working product is delivered to the customer in each release.

Extreme Programming



Extreme Programming

XP rules (www.extremeprogramming.org)

- It follows 80-20 rules which states that 80% of the profits come from 20% of the development.
- Computers must be placed in the middle of the big room.
- Tasks that are longer than 3 days are further broken down and which are shorter than one day are grouped together.
- Pair programming should not be misinterpreted as a student-teacher relationship.
- Each iteration must not last more than a few weeks.
- Customer must always be present at the developer’s site.

Extreme Programming

XP rules (www.extremeprogramming.org)

- Spikes are used for elaborating user stories.
- Continuous refactoring is followed.
- Class Responsibility Collaboration (CRC) cards are used to design the system. These are used to determine classes and interactions between them.
- Entire team is responsible for the system’s architecture.
- Integration is done within every few hours.
- The developer cannot overtime for the second consecutive week.

Extreme Programming

XP rules (www.extremeprogramming.org)

- The main drawback of XP is that it is “hard to do simple things”.
- XP is not suitable for projects where high documentation is required by the customer.
- It is also not suitable in cases where customer cannot be involved continuously.
- XP may not be used where technology is complex and/or time to obtain feedback as a consequence of integration becomes difficult.

Object Oriented Software Life Cycle Models

- The bases of object oriented methodology are objects incorporating attributes and functions that operate on these attributes.
- The focus of object oriented software life cycle is on objects rather than on processes as in the case of traditional waterfall and other models.
- The object oriented software life cycle models enables the designers and coders to properly design the objects and the interrelationships between these objects.
- This allows to produce a simple to understand, modular, flexible and reusable design of the object oriented software under development.

Object Oriented Software Life Cycle Models

	Traditional approach	Object oriented approach
Methodology	Functional and process driven	Object driven
Requirement analysis phase	Data flow diagrams, data dictionary, ER diagrams.	Use case approach for requirement capturing
Analysis phase		Object identification and description, attributes and functions that operate on those attributes are determined.
Design phase	Structured chart, flowchart, pseudo code	Class diagram, object diagrams, sequence diagrams, collaboration diagrams
Implementation and testing phase	Implementing processes functions, unit, integration and system testing.	Implementing objects and interactions amongst the objects. Unit, integration and system testing.
Documentation	Many documents are produced at end of each phase	A document may or may not be produced at end of each phase.

Object Oriented Software Life Cycle Models

Object oriented requirement analysis: In this phase use case approach is used for capturing requirements from the customer. The SRS document containing the use case description may be produce during this phase.

Object oriented analysis: In this phase objects and the inter relationships between these objects are identified. The functions of the objects are determined. An ideal structure is created in order to build high quality maintainable software.

Object Oriented Software Life Cycle Models

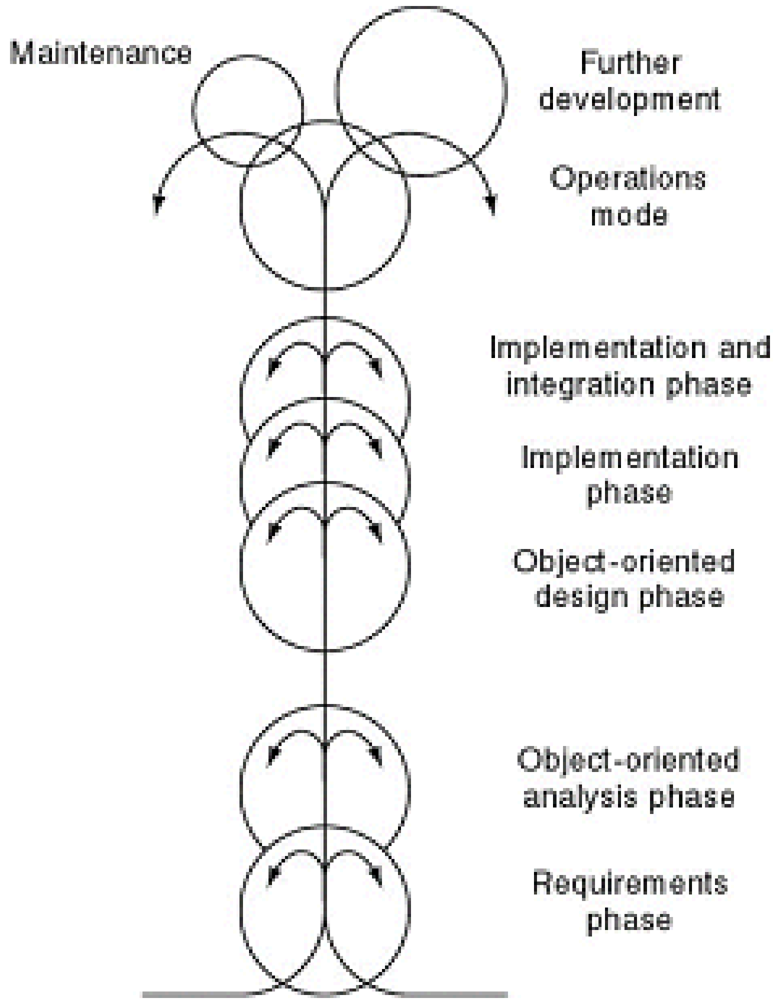
Object oriented design: In this phase the objects are defined in detail keeping in mind the implementation environment. All the objects identified during object oriented analysis phase must be traced to the design phase.

Object oriented programming and testing: The objects, functions encapsulated in these objects and interaction amongst the objects are implemented in terms of source code. The existing developed and well tested components may be used and new components are created. This feature supports reusability of source code. The developed components are tested using object oriented testing techniques.

Fountain model

- The fountain model provides a clear representation of iterations and overlapping phases.
- The model emphasizes on reusability of source code.
- The circles depict the overlapping phases and arrows within circles depict iterations within the phases.
- The smallest circle represents the maintenance phase showing that the maintenance phase is smallest in fountain model.

Fountain model


Maintenance
Further development
Operations mode
Implementation and integration phase
Implementation phase
Object-oriented design phase
Object-oriented analysis phase
Requirements phase

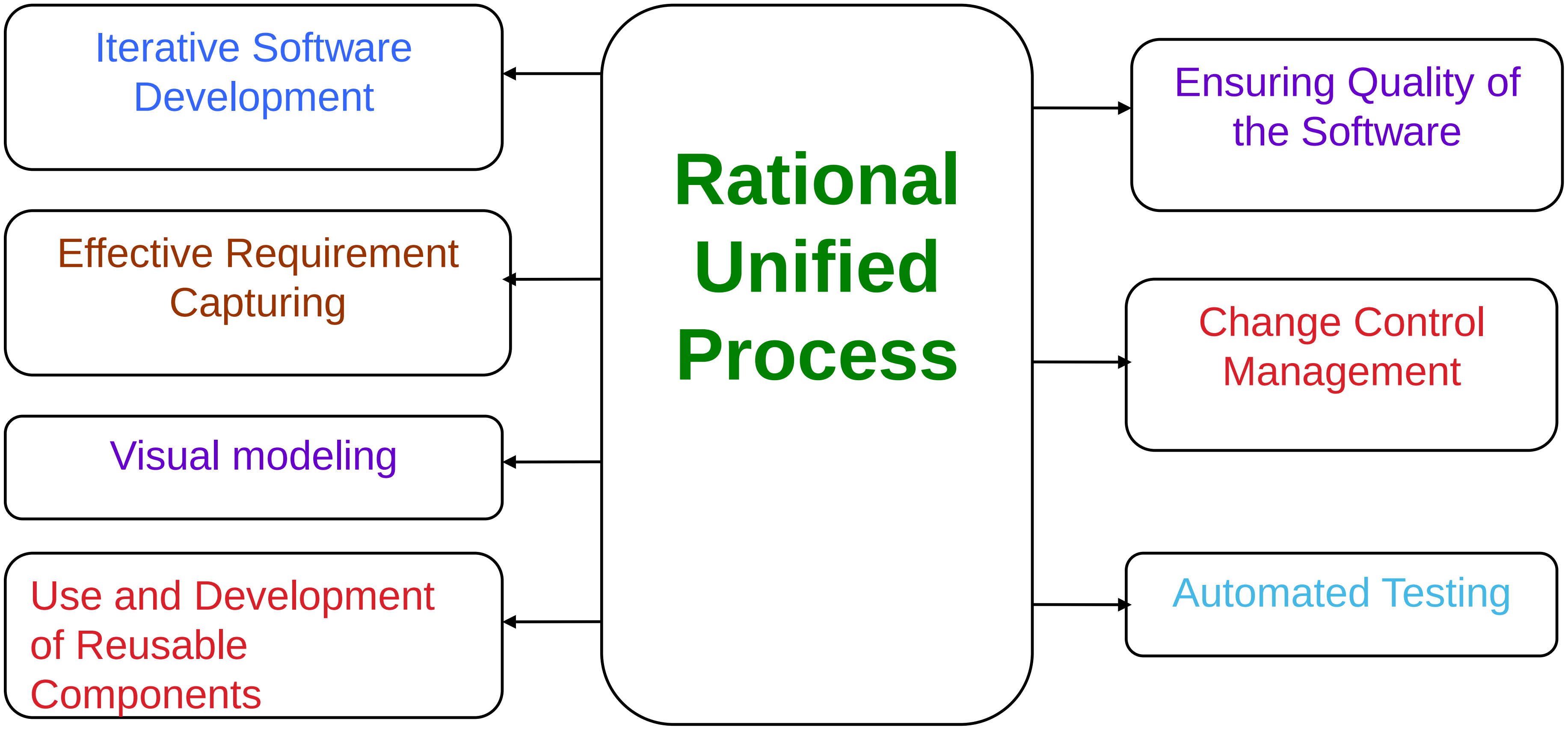
Rational Unified Process

- Rational Unified process (RUP) is maintained by Rational Software.
- It provides a process framework that may be adapted by organizations according to their needs.
- RUP supports iterative software development where the software goes through a series of short length mini projects called iterations.
- RUP forms the basis for providing guidelines on the use of Unified Modeling Language (UML).

Rational Unified Process

- UML is a popular standard for visually modeling elements of the system and is governed by Object Management Group (OMG).
- UML is a language for creating, visualizing and documenting various diagrams that depict different aspects of the software.

Rational Unified Process

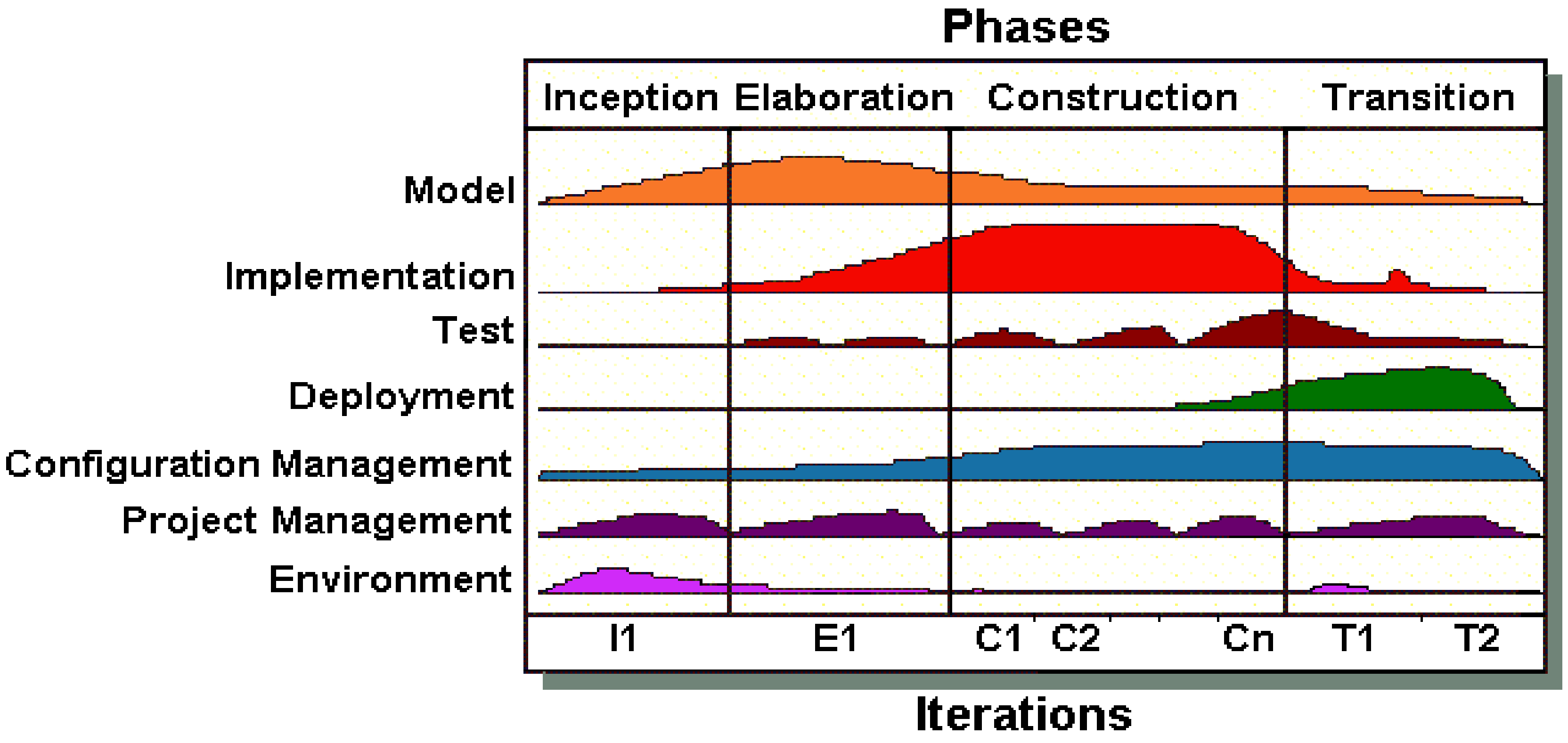

Rational Unified Process
Iterative Software Development
Effective Requirement Capturing
Visual modeling
Use and Development of Reusable Components
Ensuring Quality of the Software
Change Control Management
Automated Testing

Rational Unified Process

RUP overview

- **Static structure:** It provides the process description in terms of roles, activities, artifacts, disciplines and workflows.
- **Dynamic structure:** the dynamic aspect of the process can be viewed in terms of iterative development.

Iterations & Workflow of Unified Process

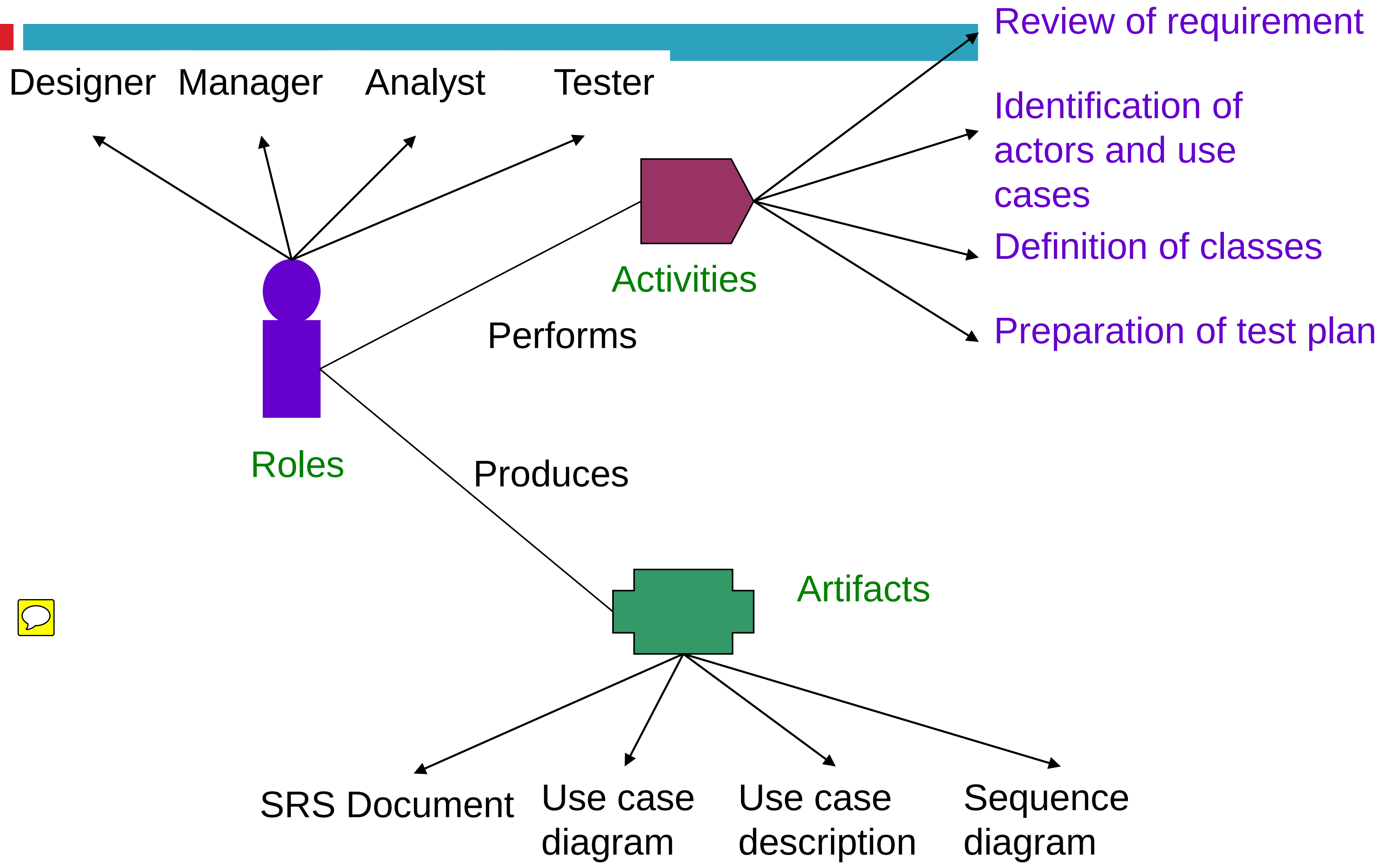


Rational Unified Process

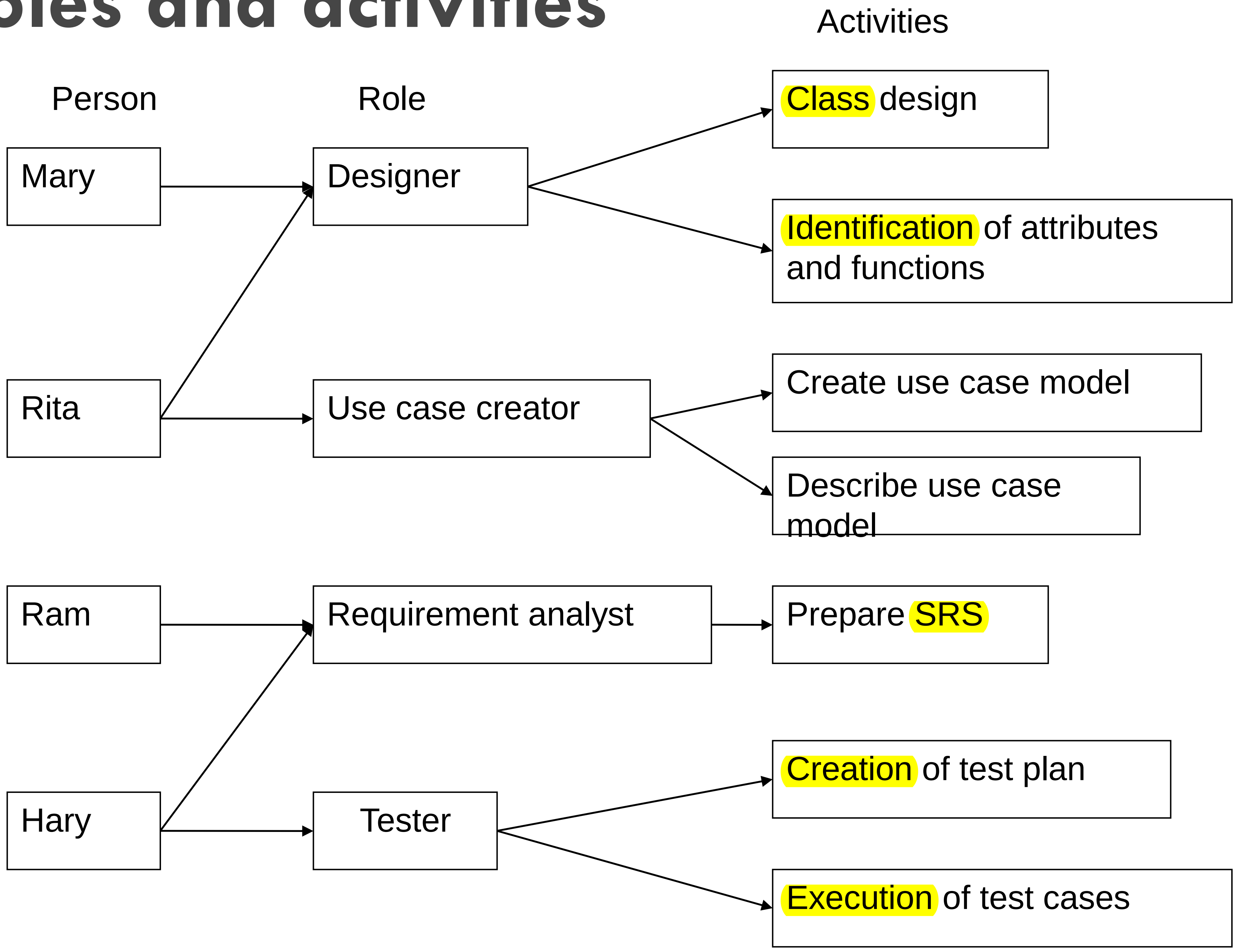
Static structure of RUP

- The static process of RUP describes who (Roles) does how (activities), what (artifacts) and when (workflows).
- Five major elements form the static structure of RUP:
  - role,
  - activities,
  - artifacts,
  - disciplines and
  - workflows.

Rational Unified Process



Roles and activities



Rational Unified Process

- **Activities are the work performed by a person in a specific role to produce the required result from the system. The following are examples of activities performed by persons in different roles:**
  - **Estimation of effort and schedule**
  - **Design of classes**
  - **Review of SRS document**
  - **Verify quality of system design**
  - **Creation of test plan and test cases**

Rational Unified Process

Artifacts

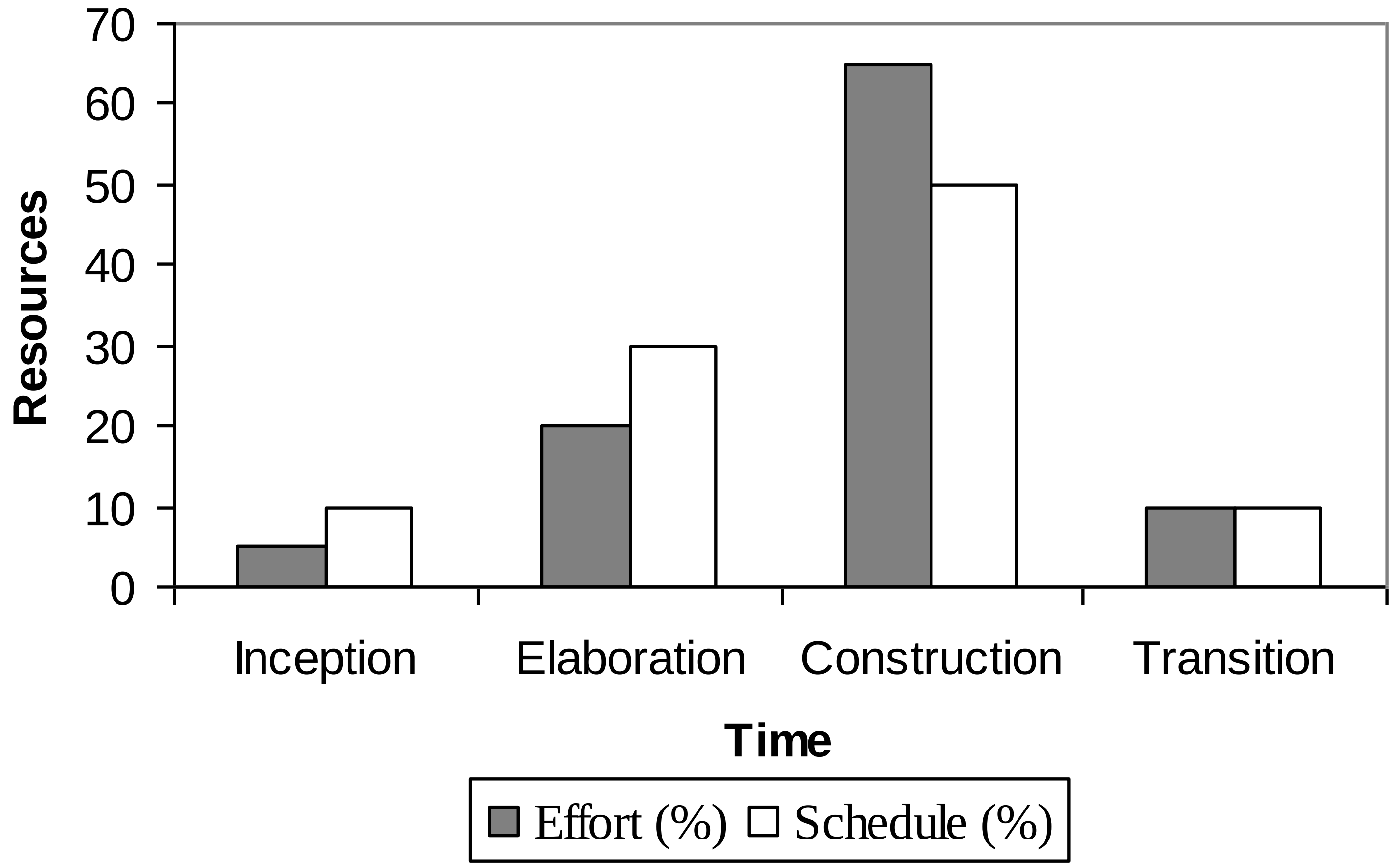
- Artifacts are the outputs produced during development life cycle of the software.
- An activity performed by a role to produce an artifact. The following are the examples of artifacts:
  - Software requirement specification
  - Use case model
  - Class model
  - Test plan
  - Source code
  - User manual

Rational Unified Process

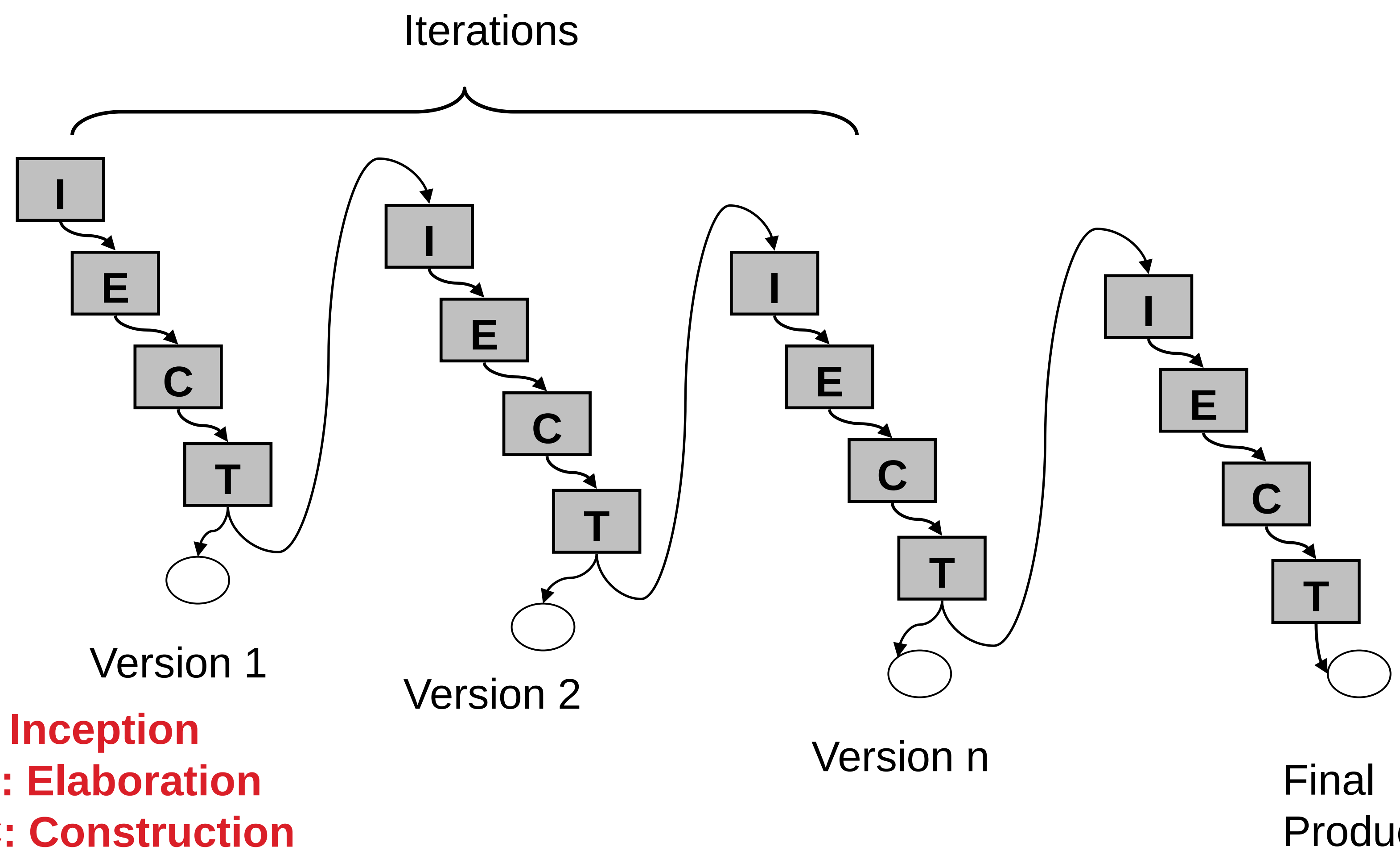
Dynamic structure of RUP

- The dynamic structure of RUP is organized along time. It consists of four phases :
  - inception,
  - elaboration,
  - construction, and
  - transition.

Rational Unified Process



Rational Unified Process

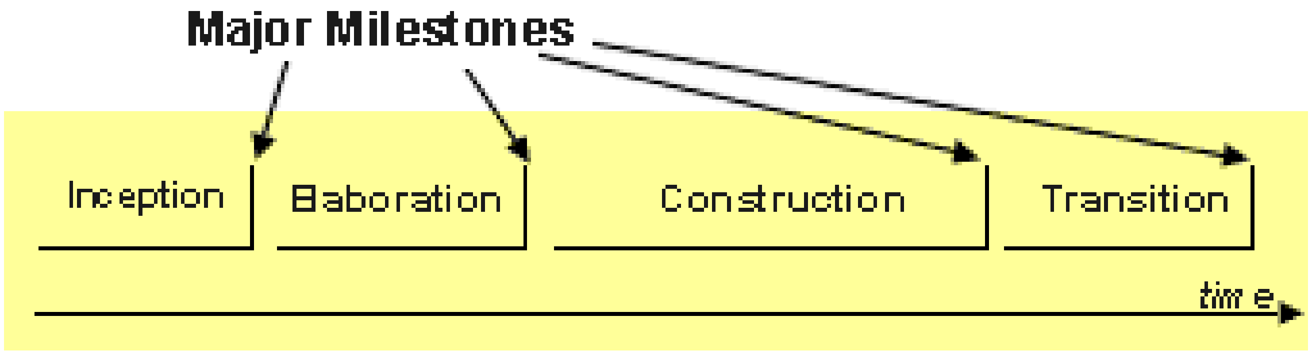


I: Inception
E: Elaboration
C: Construction
D: Deployment

Rational Unified Process

The essential activities of inception phase are:

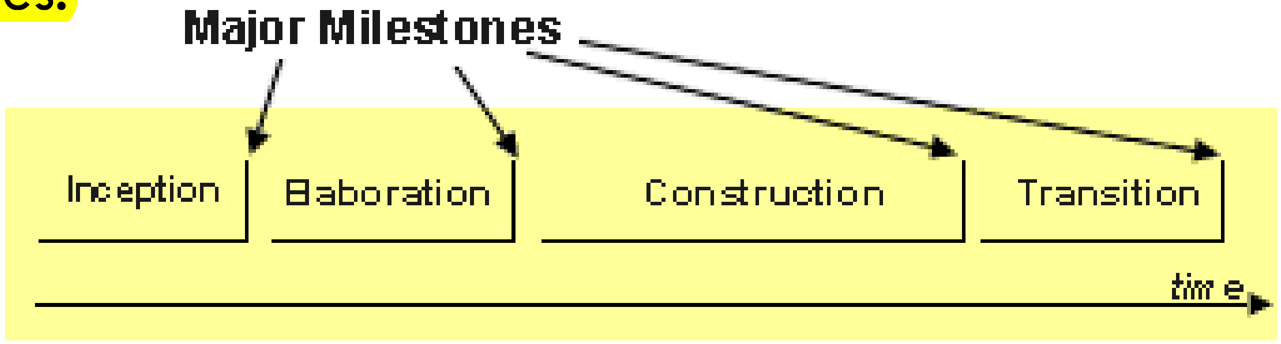
- Establishment of scope and boundary of the project.
- Determination of cost and schedule of the project.
- Identification of actors and use cases.
- Development of initial iteration plan.
- Determination of high level risks.



Rational Unified Process

The essential activities of elaboration phase are:

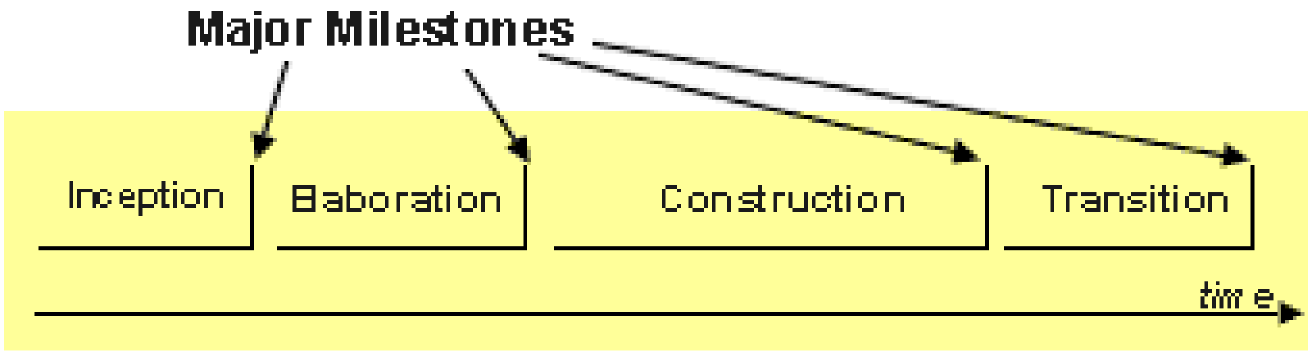
- Establishment and validation of architectural baselines.
- To address significant risks.
- Design of use case model.
- Selection of components and formation of policies for their purchase and usage.
- Creation of detailed iteration plans.
- Development of prototypes.



Rational Unified Process

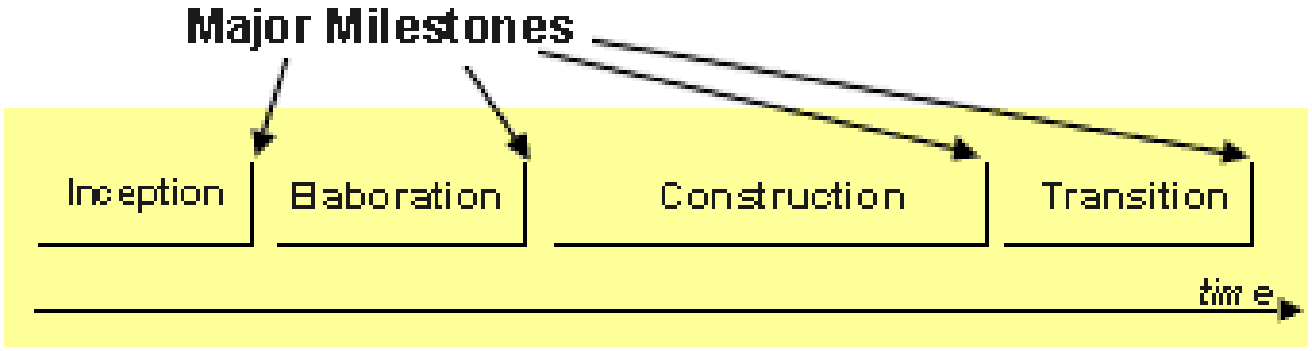
The essential activities of construction phase are:

- Optimization of resources by avoiding rework and unnecessary coding.
- Assessment and verification of quality.
- Testing all the functionalities of the product. Testing activities include unit, integration, and system testing.

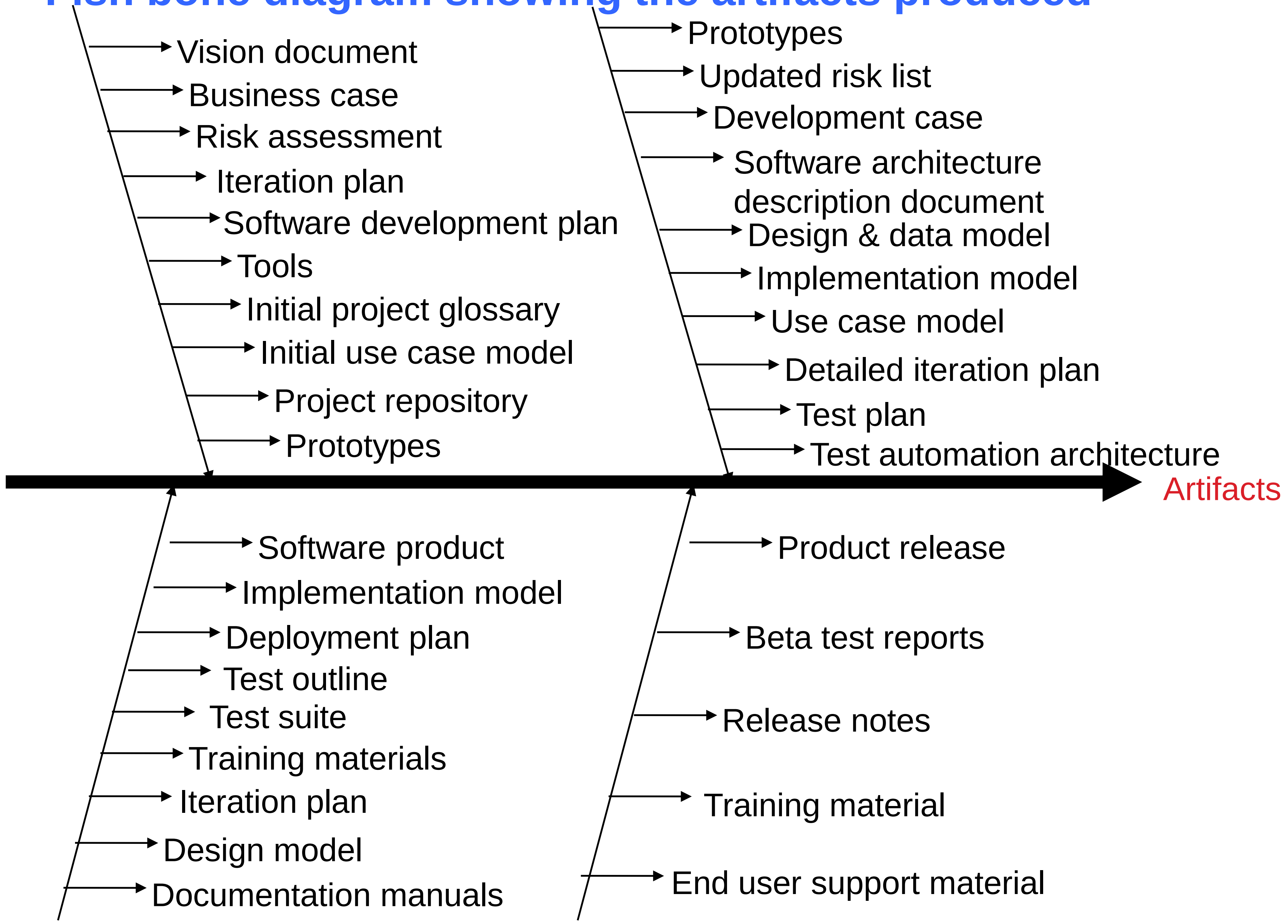


Rational Unified Process

- Transition phase includes delivering, training users and maintaining the software. Elaboration involves beta releases, bug fixes and enhancement releases.



Fish bone diagram showing the artifacts produced


Vision document
Business case
Risk assessment
Iteration plan
Software development plan
Tools
Initial project glossary
Initial use case model
Project repository
Prototypes
Prototypes
Updated risk list
Development case
Software architecture description document
Design & data model
Implementation model
Use case model
Detailed iteration plan
Test plan
Test automation architecture
Artifacts
Software product
Implementation model
Deployment plan
Test outline
Test suite
Training materials
Iteration plan
Design model
Documentation manuals
Product release
Beta test reports
Release notes
Training material
End user support material