Software Engineering

Lecture 2

Software Process

Learning Outcomes

- Appreciate the need for a defined software process
- Be able to describe in detail the main software process models
- Be able to compare software process models and choose between them

What is a Software Process?

- 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
- 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

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 (continued)

- 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

Software process models

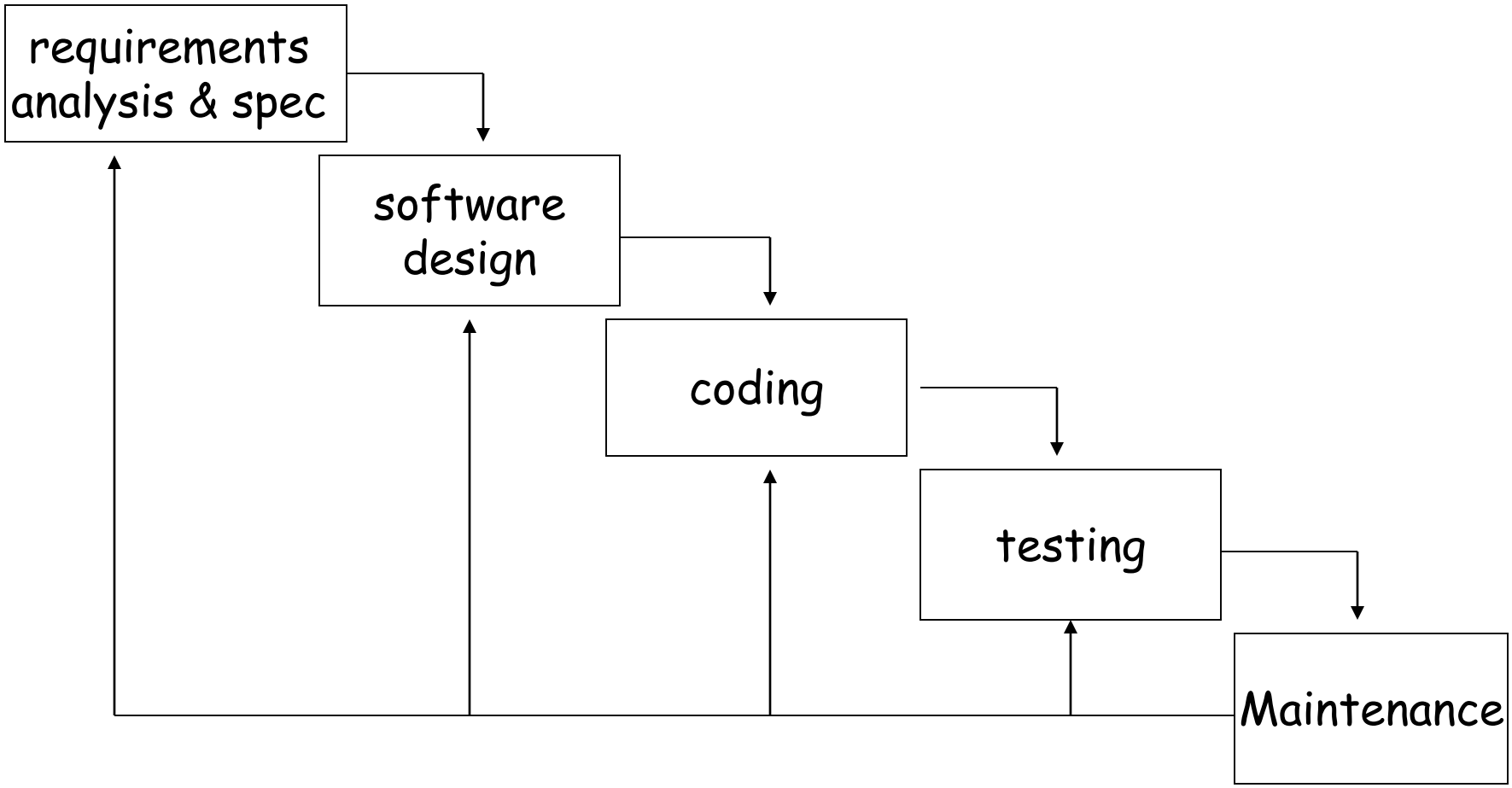
- Waterfall model
- Prototyping models
- Evolutionary models
- The spiral model
- Formal development
- Incremental development
- Rapid Application Development
- Unified Process
- Agile Process
- Extreme Programming (XP)

The Waterfall model

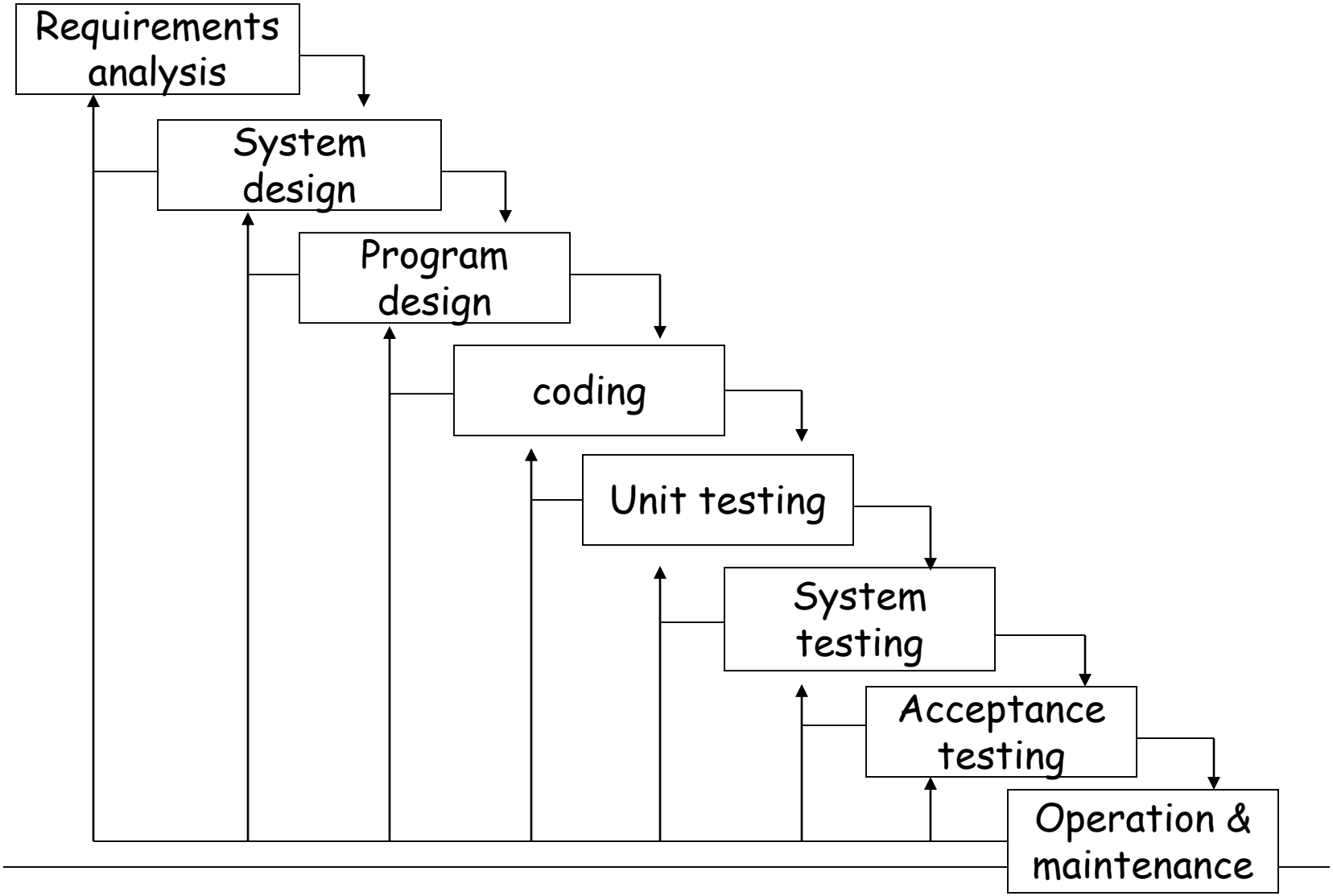
Separate and distinct phases of specification and development

A linear sequential model

Waterfall model



Waterfall Model II



The Waterfall Model

Software Requirement Analysis and 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, behaviour, 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.

Some 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.

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

Prototyping

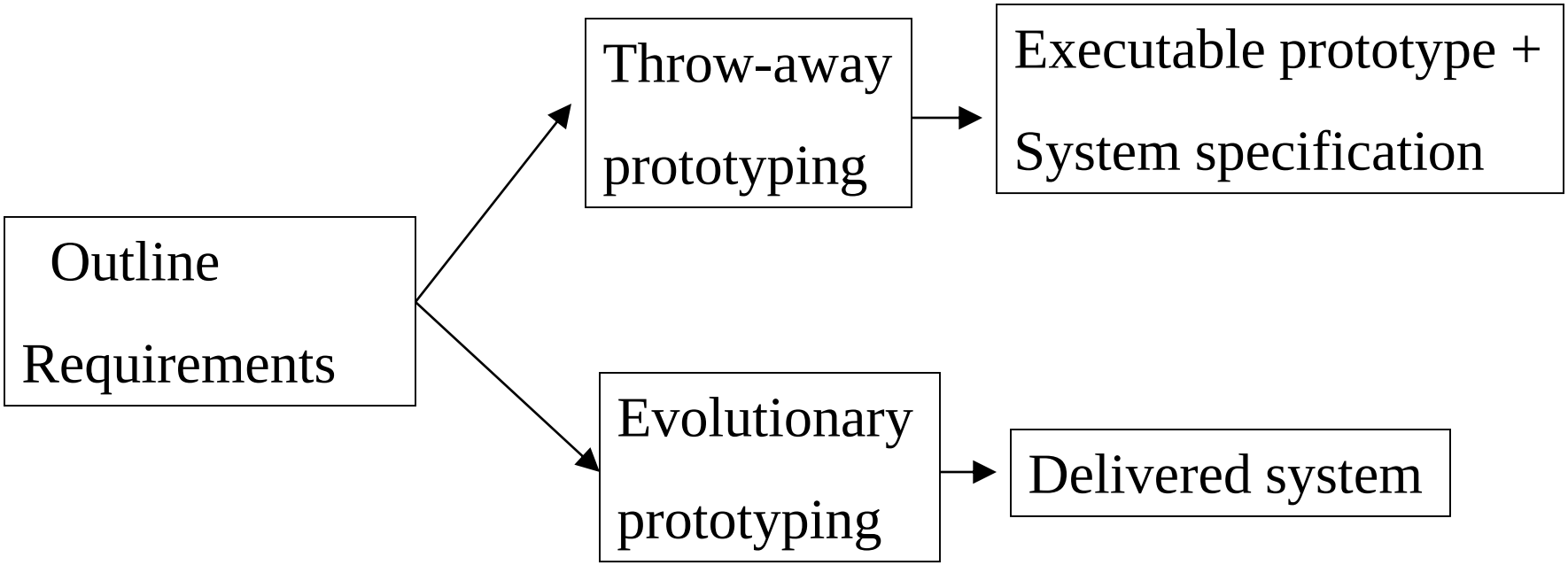
It is very difficult for end-users to anticipate how they will use new software systems to support their work. If the system is large and complex, it is probably impossible to make this assessment before the system is built and put into use.

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.

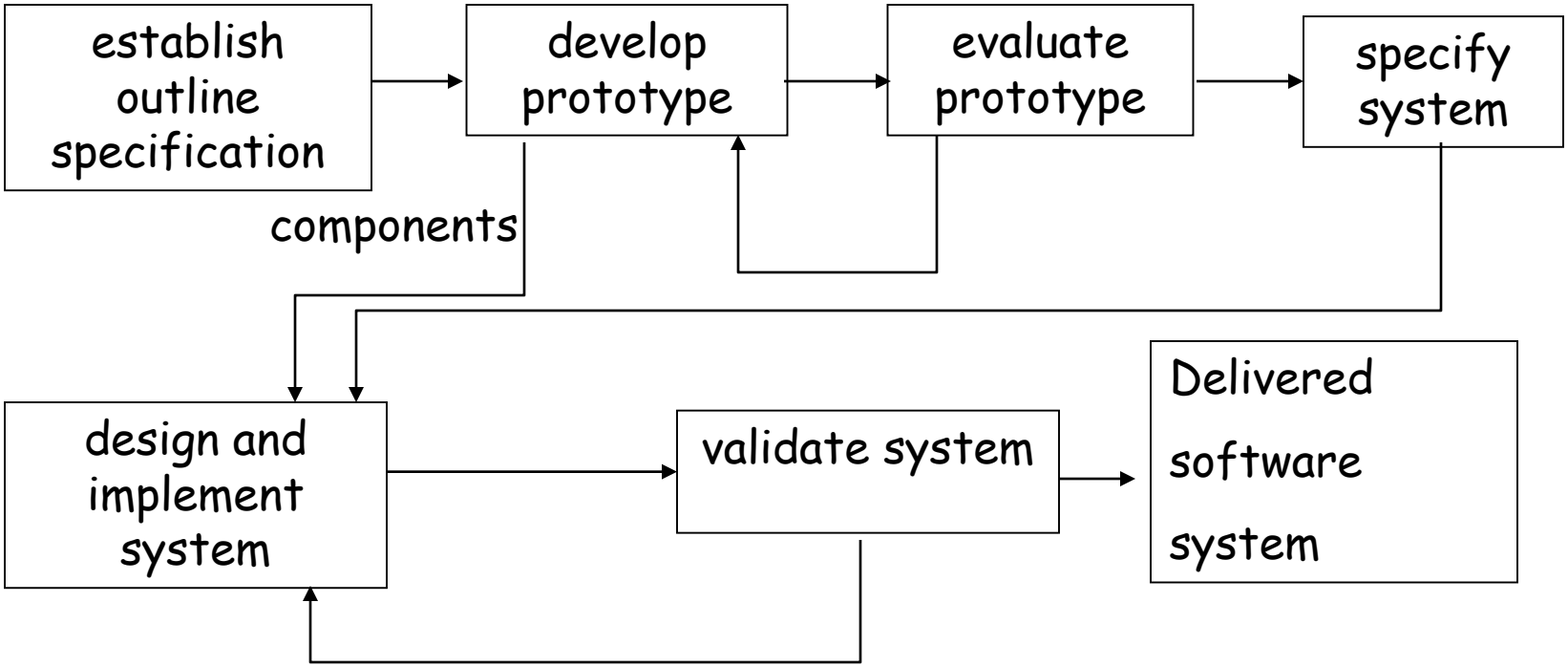
There are two types of Prototyping techniques

* Throw-away Prototyping
* Evolutionary Prototyping

Throw-away and Evolutionary Prototyping



Throw-away Prototyping



Throw-away Prototyping

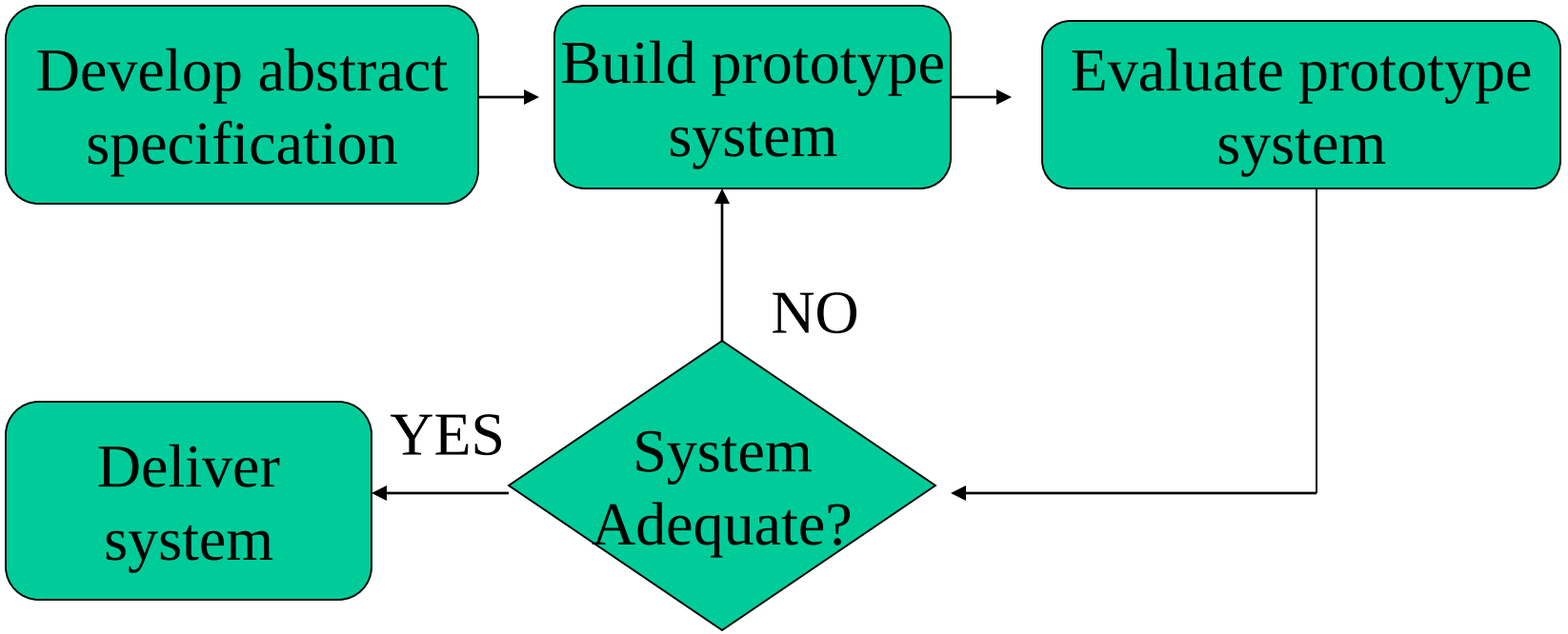
The objective is to understand the system requirements clearly. Starts with poorly understood requirements. Once the requirements are cleared, the system will be developed from the beginning.

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

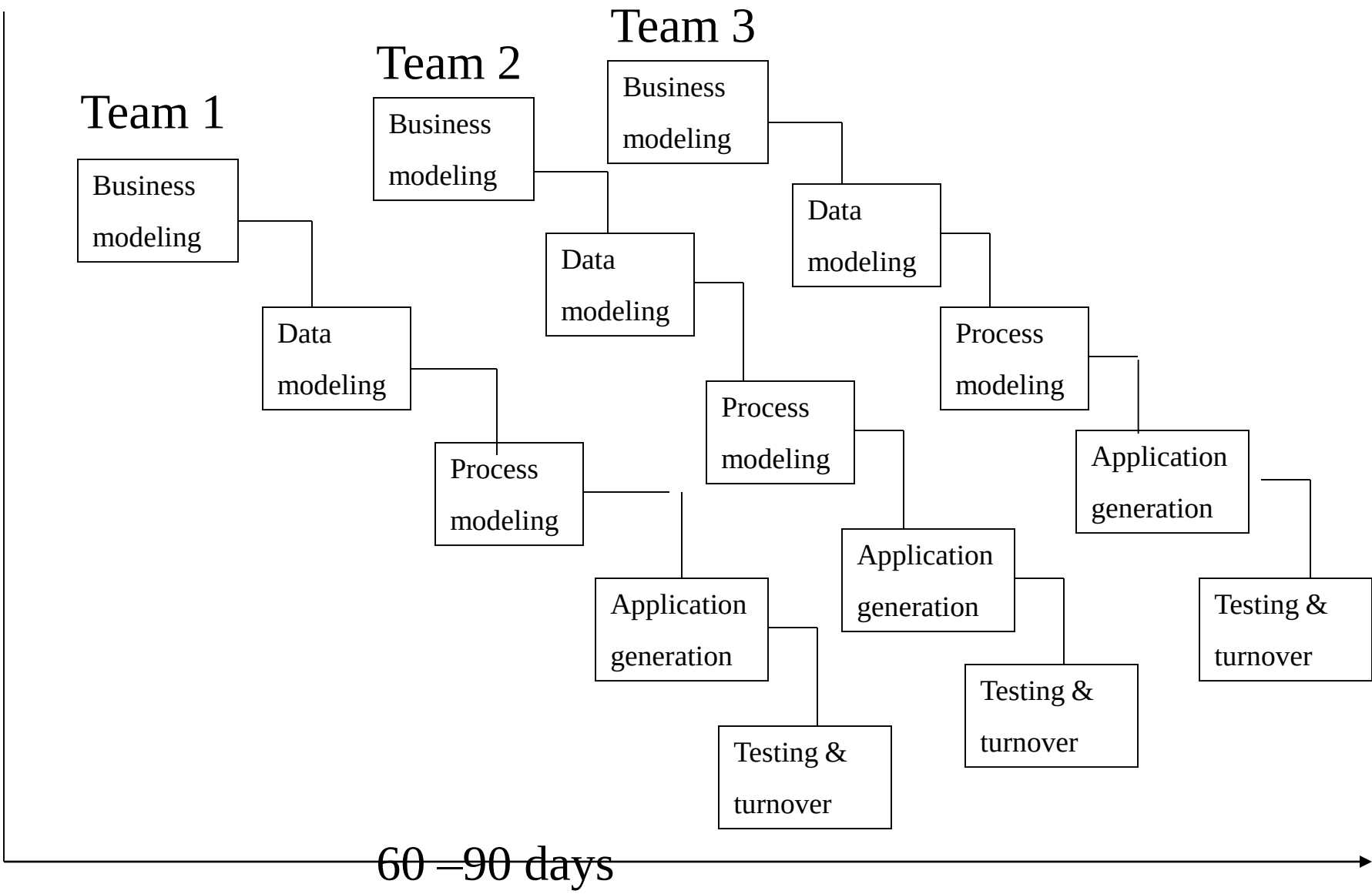
* Effort of prototype is not wasted
* Faster than the Waterfall model
* High level of user involvement from the start
* Technical or other problems discovered early – risk reduced
* mainly suitable for projects with vague and unstable requirements.

Evolutionary Prototyping (continued)

Disadvantages

* Prototype usually evolve so quickly that it is not cost- effective to produce greate 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.

The RAD Model



The RAD Model

Rapid Application Development (RAD) is an incremental software development process model that emphasizes an extremely short development cycle.

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)

Processes in the RAD model

Business modelling

The information flow in a business system considering its functionality.

Data Modelling

The information flow defined as part of the business modelling phase is refined into a set of data objects that are needed to support the business

Process Modelling

The data objects defined in the data modelling phase are transformed to achieve the information flow necessary to implement business functions.

Application generation

RAD assumes the use of 4GL or visual tools to generate the system using reusable components.

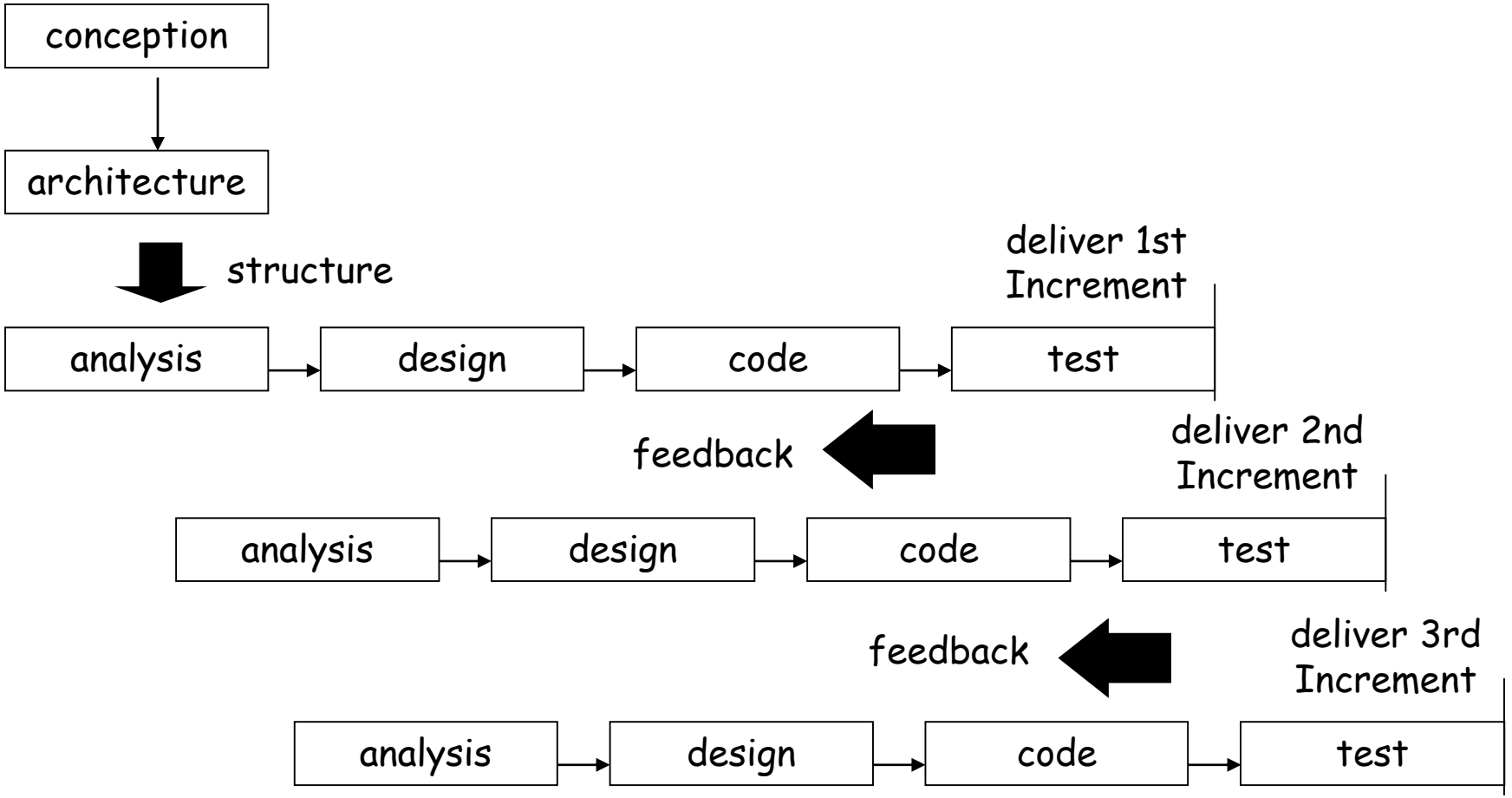
Testing and turnover

New components must be tested and all interfaces must be fully exercised

Some 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.

Incremental Model



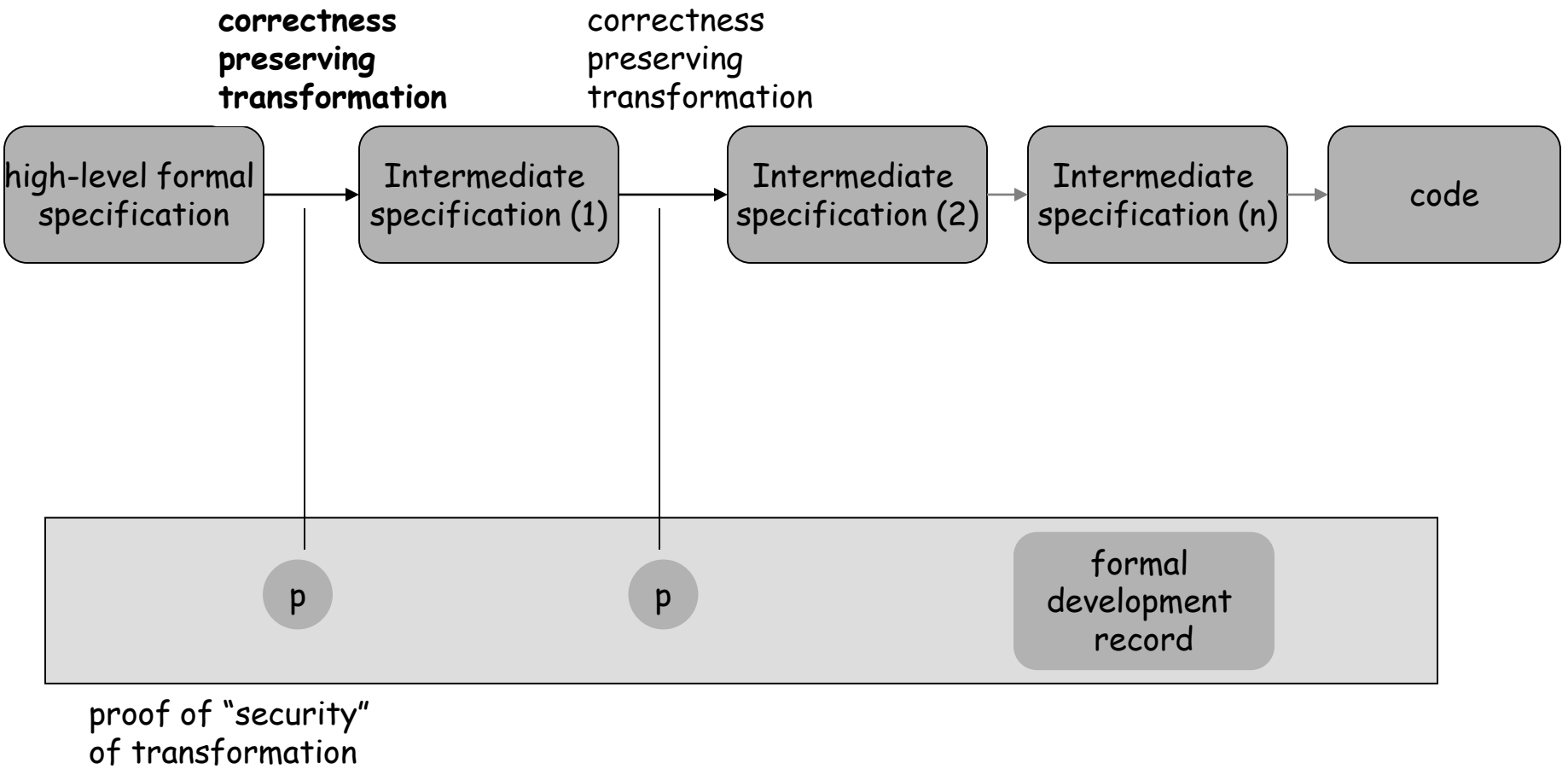
Incremental Development

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. 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

Formal Development Model



Formal Specification

•Formal software specifications are mathematical entities and may be analyzed using mathematical methods. Specification consistency and completeness can be proved mathematically.

• Formal specifications may be automatically processed. Software tools can be used to build programs from formal specifications.

• The development of a formal specification provides insights into and an understanding of the software requirements and the software design.

Some problems with formal development methods

•Many software engineers have not been trained in the techniques required to develop formal specifications.

• Customers may be unwilling to fund development activities that they cannot easily monitor.

• Software management is inherently conservative and is unwilling to adopt new techniques for which payoff is not obvious.

• Most of the effort in specification research has been concerned with the development of languages and their theoretical aspects rather than tools and methods.

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 is divided into four main task regions

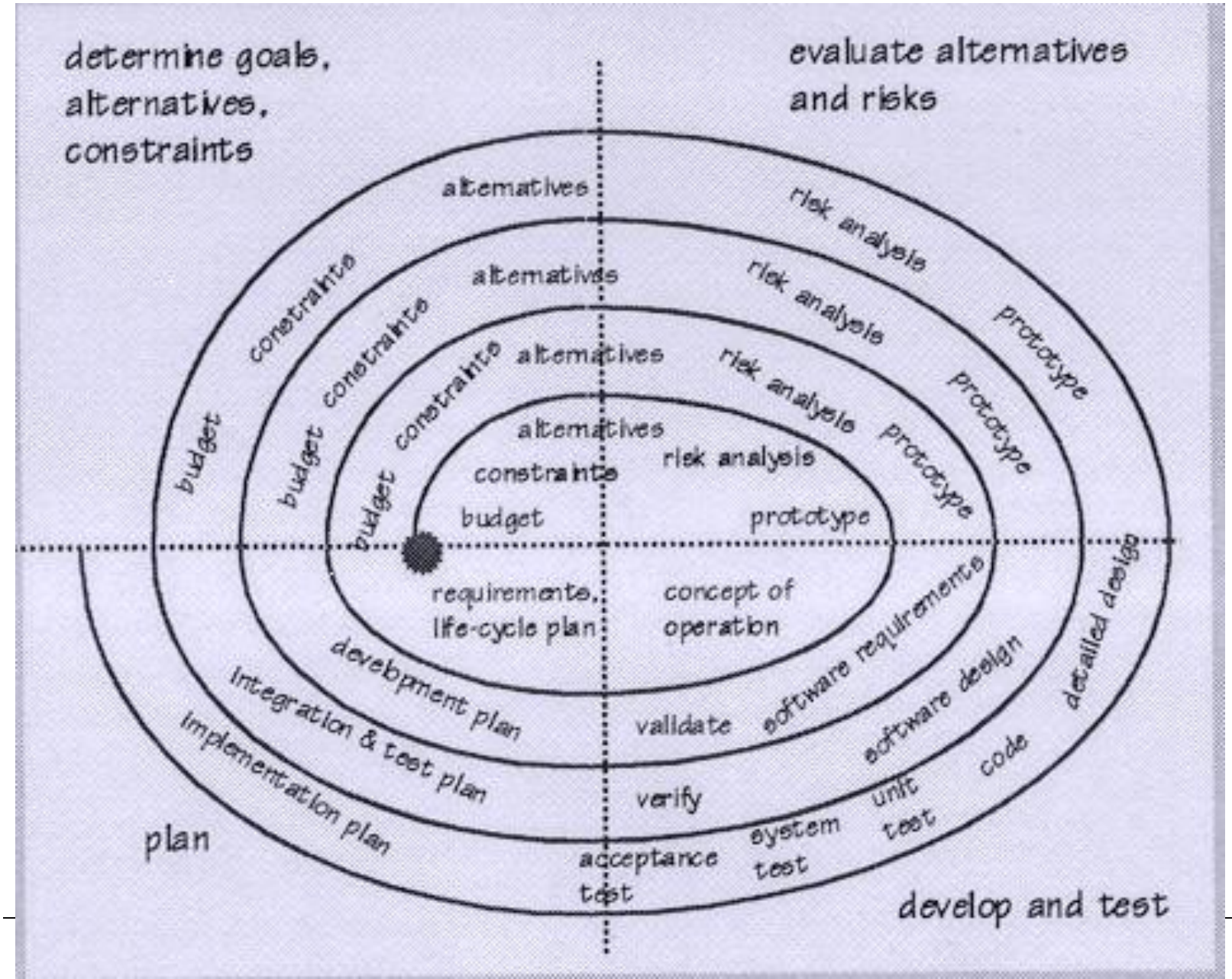
Determine goals, alternatives,constraints

Evaluate alternatives and risks

Develop and test

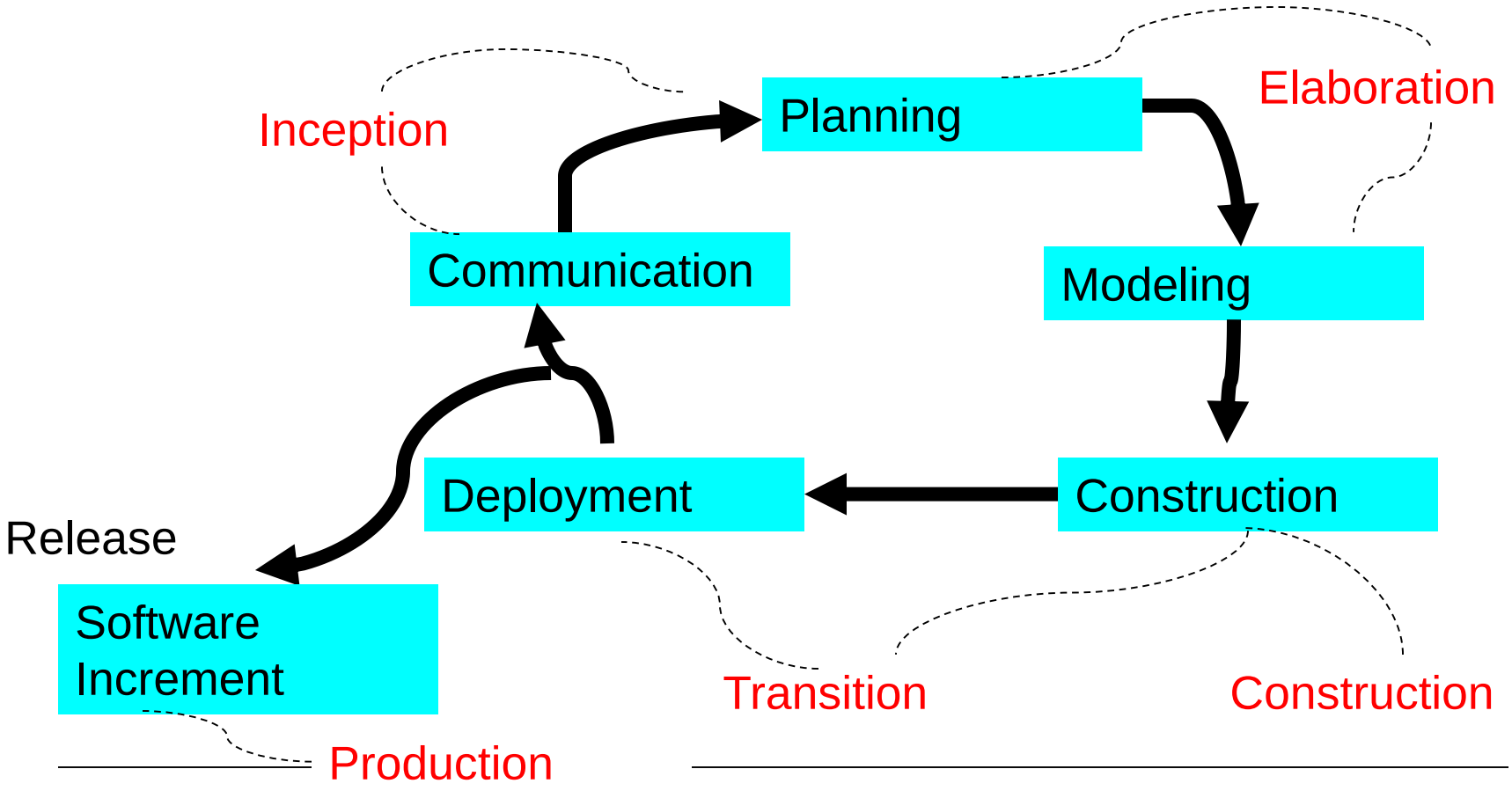
Plan

Spiral Model



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

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 use-case 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.

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

Vision document
Initial use-case model
Initial risk assessment
Project Plan
Business model
Prototypes

Elaboration Phase

Use-case model
Requirements functional, non-functional
Analysis model
Software architecture
Preliminary design model
Revised risk list, revised prototypes

Construction Phase

Design model
Software components
Integrated software increment
Test cases
Test Plan and Procedures
Support documentation

Transition Phase

Delivered software increment
Beta test reports
General user feedback

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.

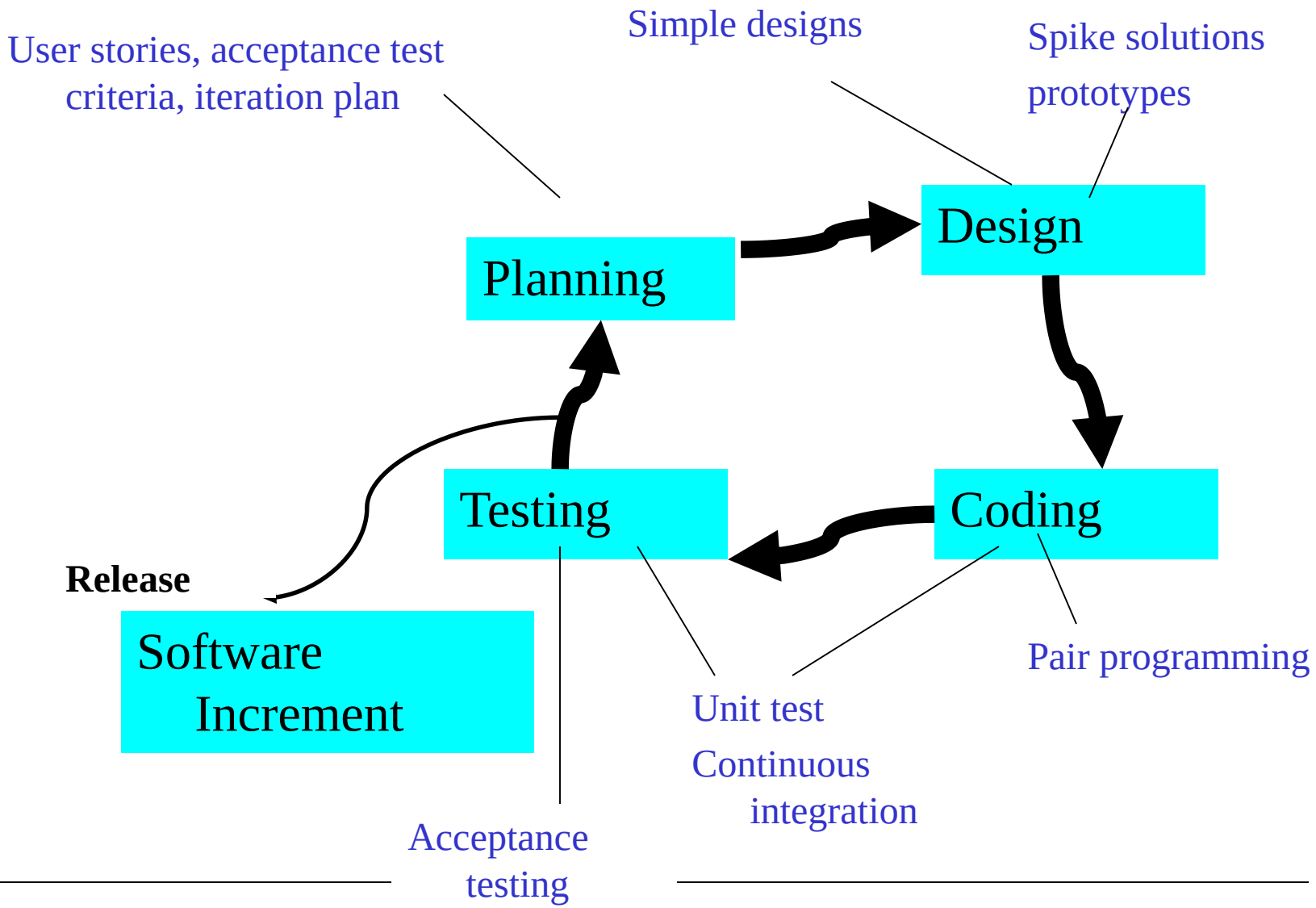
Extreme Programming (XP)

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 is a discipline of software development based on values of simplicity, communication, feedback and courage”

- Ron Jeffries

Extreme Programming (XP) Process



Extreme Programming (XP) Process

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 (XP) Process

Design

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

Extreme Programming (XP) Process

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 (XP) Process

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.

SCRUM

- **Scrum** is an iterative incremental process of software development commonly used with agile software development.
- Although Scrum was intended to be for management of software development projects, it can be used in running software maintenance teams, or as a program management approach.

SCRUM

Scrum has the following principles which follow the agile process manifesto:

- Team sizes are small which is intended to increase communication, increase tacit and informal knowledge and reduce overheads.
- To achieve the best acceptable product the process must be adaptable to both technical and business changes.
- The incrementing software should accommodate to inspect, adjust, test, document and build.
- Acceptance tests are derived from user stories that have been implemented as parts of the software release.

Continued…

- Development work and the teams are low coupled. Hence the development work does not depend on the team members. Even a team member moves out of the project, it does not effect to the flow of the development work.

- Regular testing and documentation is carried out with the product build.

- At a given time there is a product which can be delivered to the customer, as the company needs money and the customer needs a functional product.

The Scrum framework has 5 stages:

- Requirements
- Analysis
- Design
- Evolution
- Delivery

Scrum encourages using set of “Software process patterns” which supports projects with tight timelines, changing requirements and business criticality. Following are some of the jargons and activities found in Scrum:

- *Backlog* :- Set of prioritized requirements that gives business value.

- *Sprints* :- Work units which can be found in the back log requirements. Typically a sprint is a 30 day time box work unit. Sprint is frozen before work starts on it. Hence Scrum provides a short term and stable environment to work.

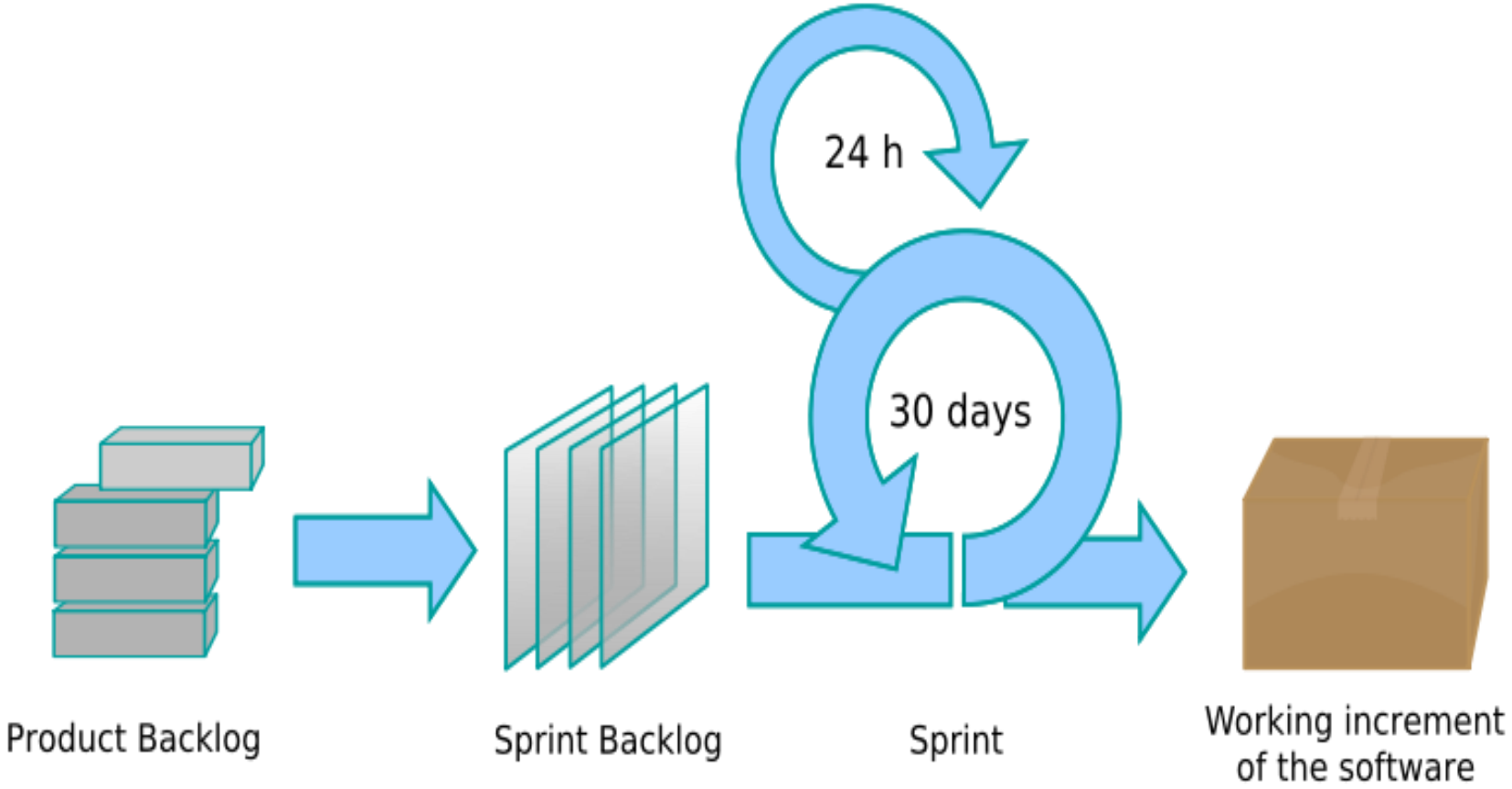
- *Scrum meetings* :- This is a daily meeting which takes around 15 minutes. This will help to:
-

Contd..

1. Identify potential problems as early as possible.
2. Helps on “knowledge socialization”.
3. The leader who conducts the meeting is called “Scrum master”.
4. Discuss matters of the project openly among the team members.

- *Demos* : - A working software application is demonstrated to customer and gets the feedback. This demo will not contain 100% functionality.

SCRUM FUNCTIONALITY



The software process

- A Structured set of activities required eg. Specification, Design, Validation, Verification, Evolution
- Activities vary depending on the organisation and the type of system being developed
- Must be explicitly modelled if it is to be managed
- Process characteristics

- Understandability
- Visibility
- Acceptability
- Reliability
- Rapidity
- Robustness
- Maintainability
- Supportability

Process characteristics

Understandability

To what extent is the process explicitly defined and how easy is it to understand the process definition?

Visibility

Do the process activities culminate in clear results so that the progress of the process is externally visible?

Acceptability

Is the defined process acceptable to and usable by the engineers responsible for producing the software project?

Reliability

Is the process is designed in such a way that process errors are avoided or trapped before they result in product errors?

Process characteristics (continued ..)

Rapidity

How fast can the process of delivering a system from a given specification be completed?

Robustness

-Can the process continue in spite of unexpected problems?

Maintainability

Can the process evolve to reflect changing organizational requirements or identified process improvements?

Supportability

To what extent can the process activities be supported by CASE tools?

Key Points

- A disciplined development process is the cornerstone of software engineering.
- Different processes are appropriate in different circumstances.
- Regardless of the process to be adopted it should always be explicitly documented and the entry and exit criteria of all the constituent should be set down clearly.

Review Questions

1. The following are initial requirements specified by some customers. Identify the most suitable process models for these software projects.

   (a) " I need to develop a simple library system for my school. I know the requirements very well"

   (b) "I need to develop a management information system for my organisation,. I may use it for all the branches in several location in Sri Lanka. I might use it on the Internet".

   (c) " I need to develop a software system to control a space shuttle"

Review Questions (continued)

2. Complete the table, placing a rank 1,2,3 in each cell

	Waterfall	Throw-away Prototyping	Evolutionary Prototyping
Coping with change			
Involving the user			
Good documentation			
Overcoming Technical difficulties along the way			
Structured code			