SOFTWARE DEVELOPMENT PROCESS

EGCO343 SOFTWARE DESIGN

KANAT POOLSAWASD
DEPARTMENT OF COMPUTER ENGINEERING
MAHIDOL UNIVERSITY

PROCESS FRAMEWORK

- A process framework establishes the foundation for a complete software process by identifying a small number of framework activities that are applicable to all software projects.

- Each framework activity is populated by a set of software engineering actions – a collection of related tasks that produces a major software engineering work product (e.g., design is a software engineering action)

GENERIC PROCESS FRAMEWORK

- The following generic process framework is applicable to the vast majority of software projects:
  - Planning
  - Analysis
  - Design
  - Implementation
- These four generic framework activities can be used during the development of small programs, the creation of large web applications, and for the engineering of large complex computer-based systems. The detail of the software process will be quite different in each case, but the framework activities remain the same.

SYSTEMS DEVELOPMENT LIFE CYCLE (1)

- The SDLC has a similar set of four fundamental phases: planning, analysis, design, and implementation. Different projects may emphasise different parts of the SDLC or approach the SDLC phases in different ways, but all projects have elements of these four phases.

SYSTEMS DEVELOPMENT LIFE CYCLE (2)

- **Planning phase** is the fundamental process of understanding why an information system should be built and determining how the project team will go about building it. It has two steps:
  - During project initiation
    - The technical feasibility (Can we build it?)
    - The economic feasibility (Will it provide business value?)
    - The organizational feasibility (If we build it, will it be used?)
- Once the project is approved, it enters—project management

SYSTEMS DEVELOPMENT LIFE CYCLE (3)

- **Analysis phase** answers the questions of who will use the system, what the system will do, and where and when it will be used. During this phase, the project team investigates any current system(s), identifies improvement opportunities, and develops a concept for the new system. This phase has three steps:
  - Analysis Strategy
  - Requirements Gathering
  - System Proposal

SYSTEMS DEVELOPMENT LIFE CYCLE (4)

- **Design phase** decides how the system will operate, in terms of the hardware, software, and network infrastructure; the user interface, forms and reports; and the specific programs, databases, and files that will be needed. Although most of the strategic decisions about the system were made in the development of the system concept during the analysis phase, the steps in the design phase determine exactly how the system will operate. The design phase has four steps:
  - Design Strategy
  - Architecture Design
  - Database and File Specifications
  - Program Design

SYSTEMS DEVELOPMENT LIFE CYCLE (5)

- **Implementation phase**, during which the system is actually built (or purchased, in the case of a packaged software design). This is the phase that usually gets the most attention, because for most systems it is longest and most expensive single part of the development process. This phase has three steps:
  - System Construction
  - System Installation
  - Support Plan

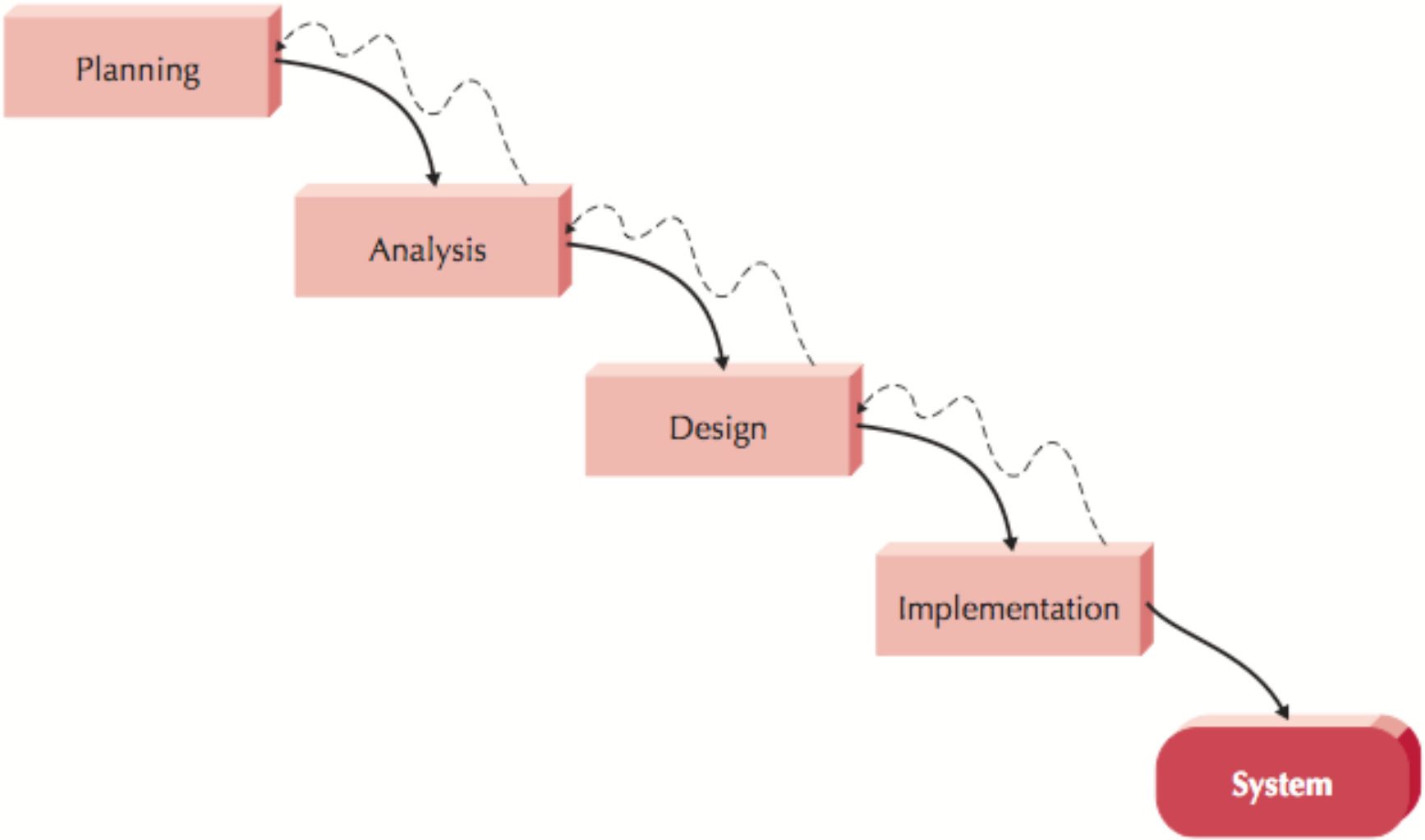
SYSTEM DEVELOPMENT METHODOLOGIES

- A methodology is a formalized approach to implementing the SDLC (i.e., it is a list of steps and deliverables). There are many different systems development methodologies, and each one is unique based on the order and focus it places on each SDLC phase. Some methodologies are formal standards used by government agencies, while others have been developed by consulting firms to sell to clients. Many organizations have internal methodologies that have been honed over the years, and they explain exactly how each phase of the SDLC is to be performed in that company.

WATERFALL MODEL (1)

- The waterfall model, sometime called the classic life cycle, suggests a systematic, sequential approach to software development that begins with customer specification of requirements and progresses through planning, modeling, construction, and deployment, culminating in on-going support of the completed software.
- The waterfall model is the oldest paradigm for software engineering. However, over the past two decades, criticism of this process model has caused even ardent supporters to question its efficacy.

WATERFALL MODEL (2)



WATERFALL MODEL PROBLEMS

- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
- Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
- Few business systems have stable requirements.
- The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.

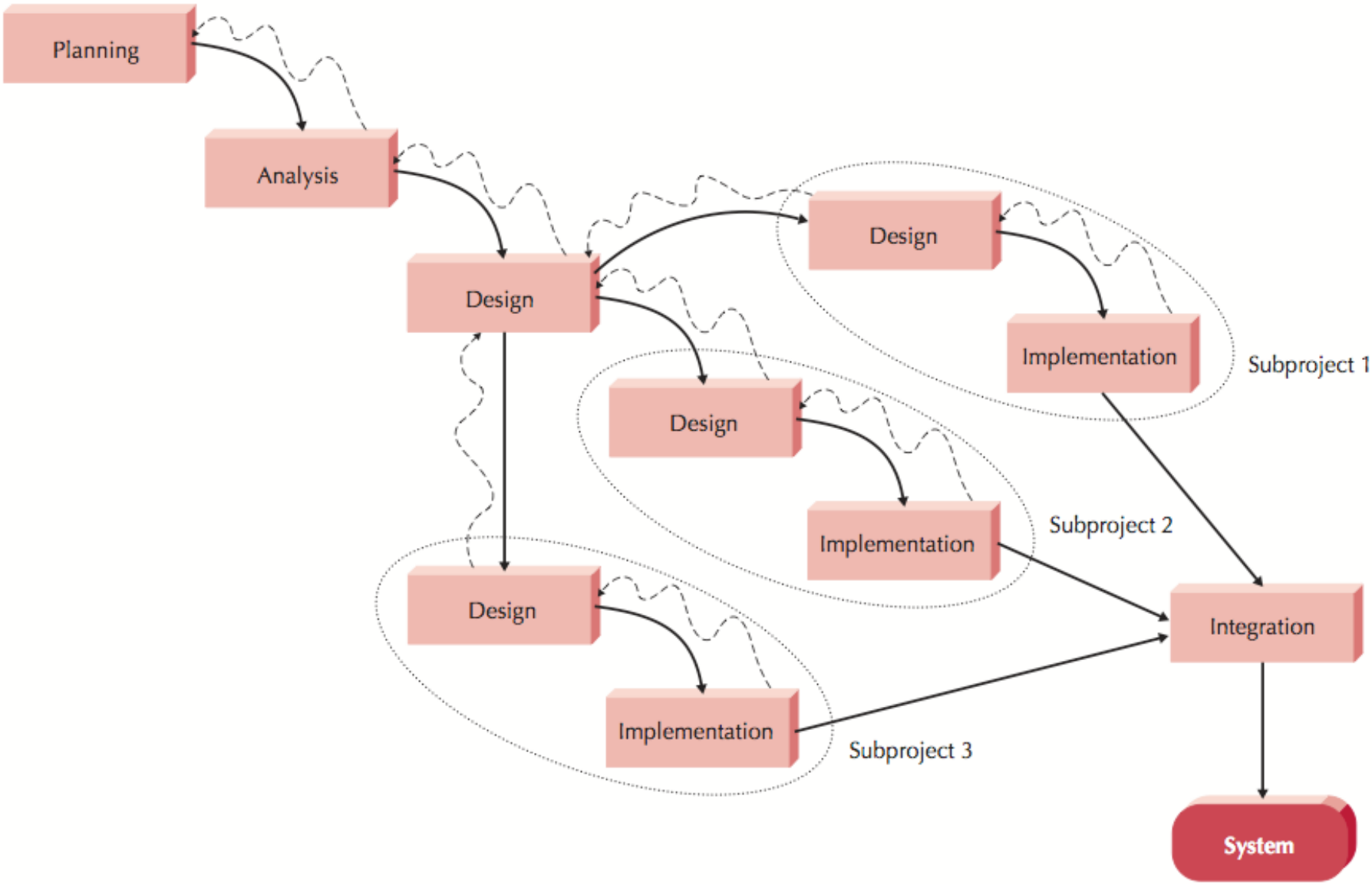
INCREMENTAL MODEL (1)

- The incremental model combines elements of the waterfall model applied in an iterative fashion.
- Referring to figure in next page, the incremental model applies linear sequences in a staggered fashion as calendar time progresses.
- Each linear sequence produces deliverable "increments" of the software.

INCREMENTAL MODEL (2)

- For example, word-processing software developed using incremental paradigm might deliver basic file management, editing, and document production functions in the first increment; more sophisticated editing, and document production capabilities in the second increment; spelling and grammar checking in the third increment; and advantaged page layout capabilities in the fourth increment.

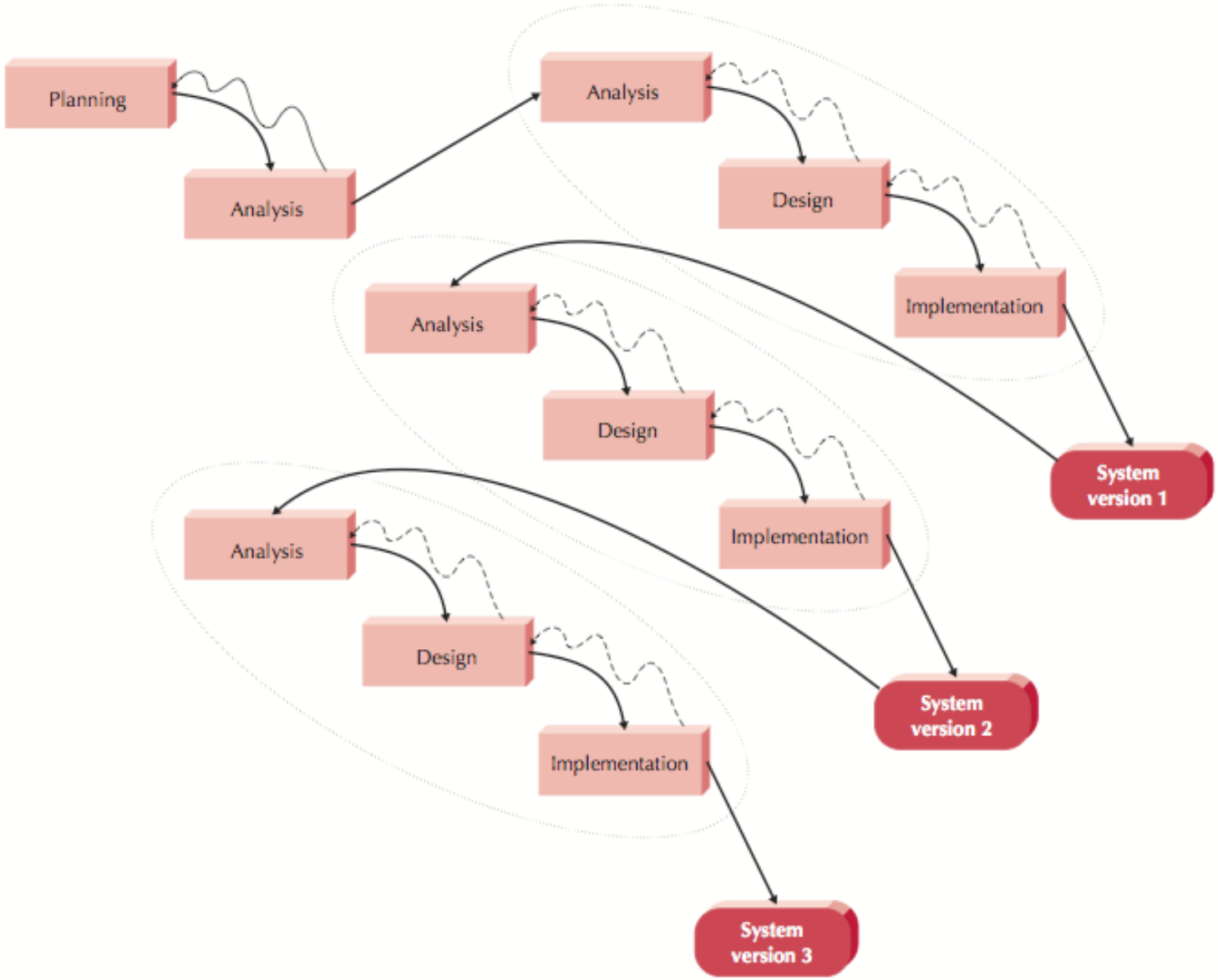
INCREMENTAL MODEL (3)



EVOLUTIONARY PROCESS MODEL (1)

- Evolutionary models are iterative. They are characterized in a manner that enables software engineers to develop increasingly more complete versions of the software
- Objective is to work with customers and to evolve a final system from an initial outline specification. Should start with well-understood requirements and add new features as proposed by the customer.

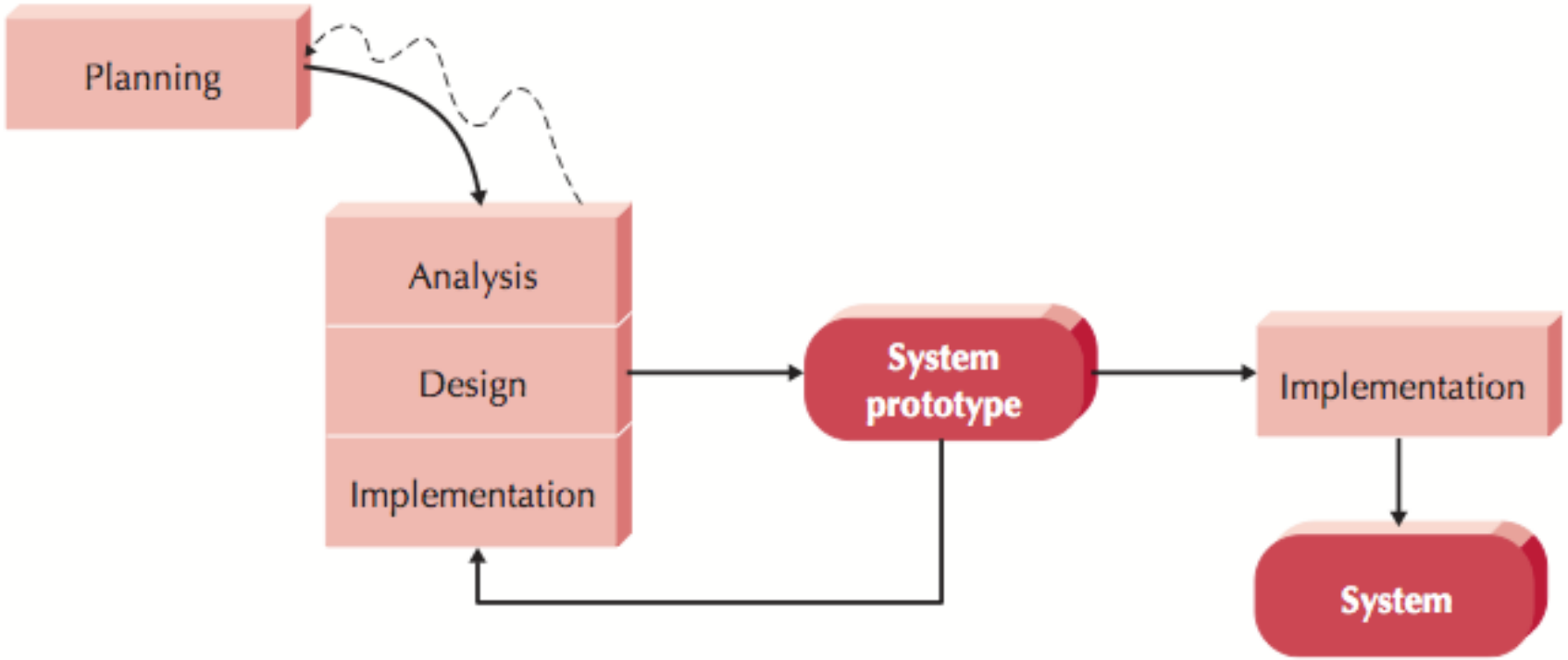
EVOLUTIONARY PROCESS MODEL (2)



PROTOTYPING (1)

- Often, a customer defines a set of general objectives for software, but does not identify detailed input, processing, or output requirements.
- In other cases, the developer may be unsure of the efficiency of an algorithm, the adaptability of an operating system, or the form that human-machine interaction should take.
- In these, and many other situations, a prototyping paradigm may offer the best approach.

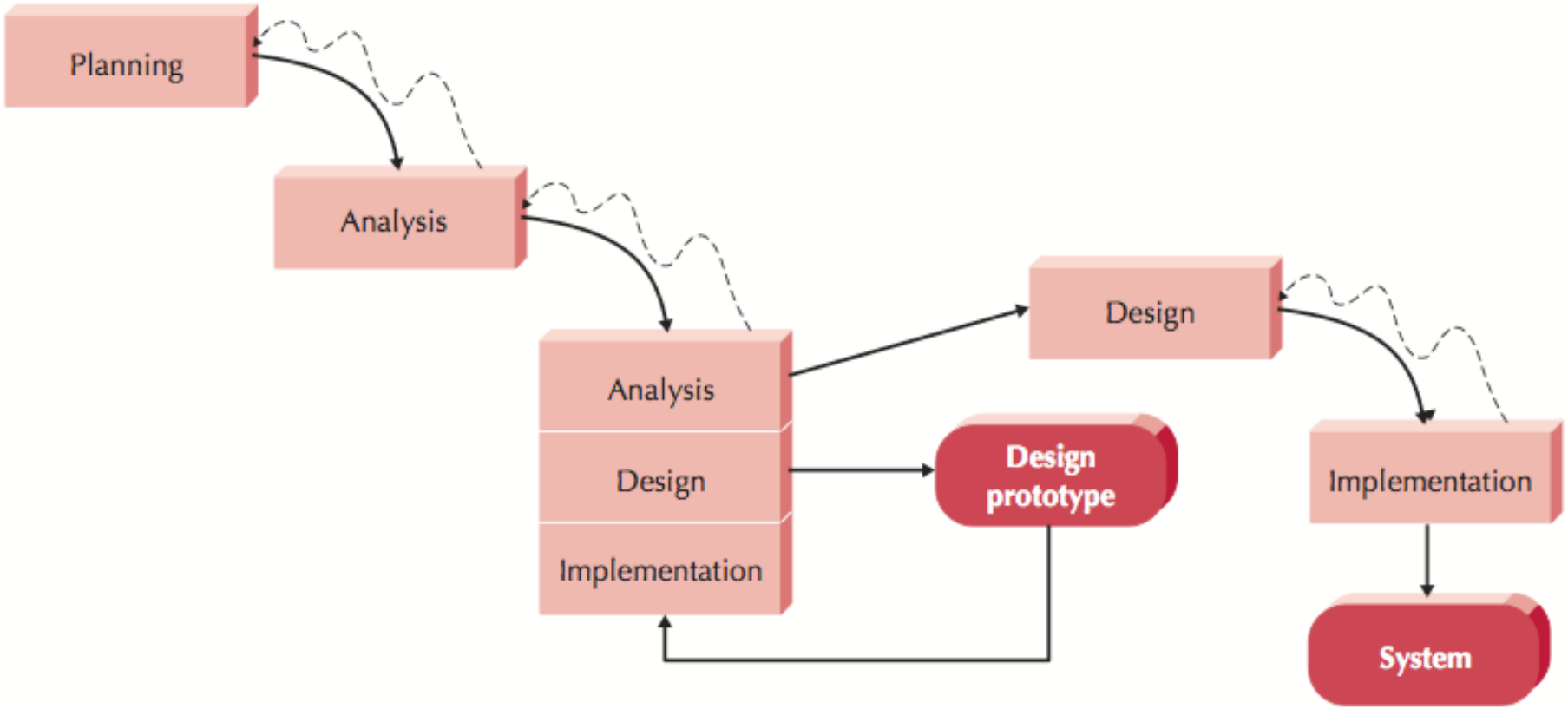
PROTOTYPING (2)



THROWAWAY PROTOTYPING (1)

- Throwaway prototyping-based methodologies are similar to prototyping-based methodologies in that they include the development of prototypes; however throwaway prototypes are done at a different point in the SDLC. These prototypes are used for a very different purpose than ones previously discussed, and they have a very different appearance

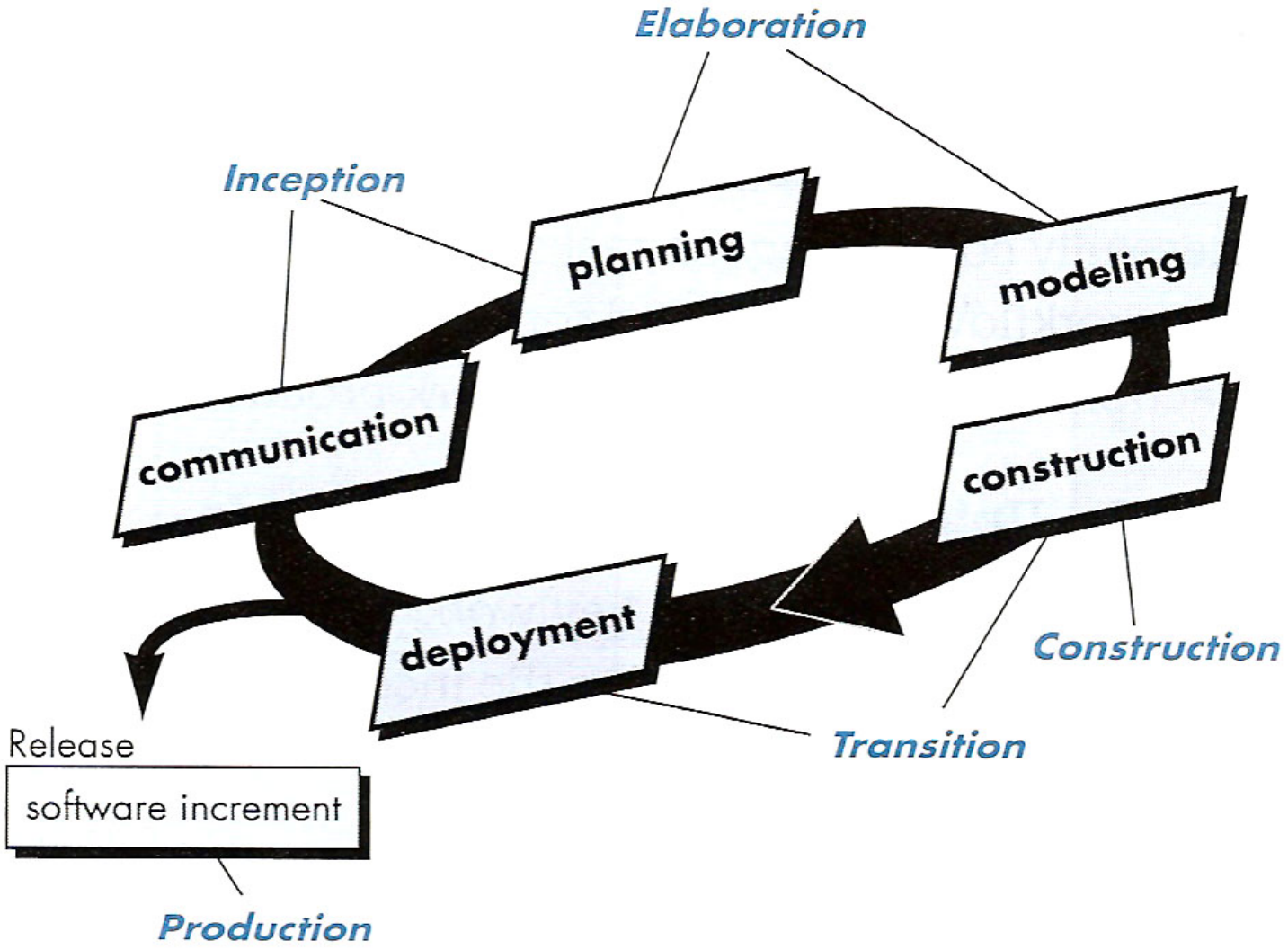
THROWAWAY PROTOTYPING (2)



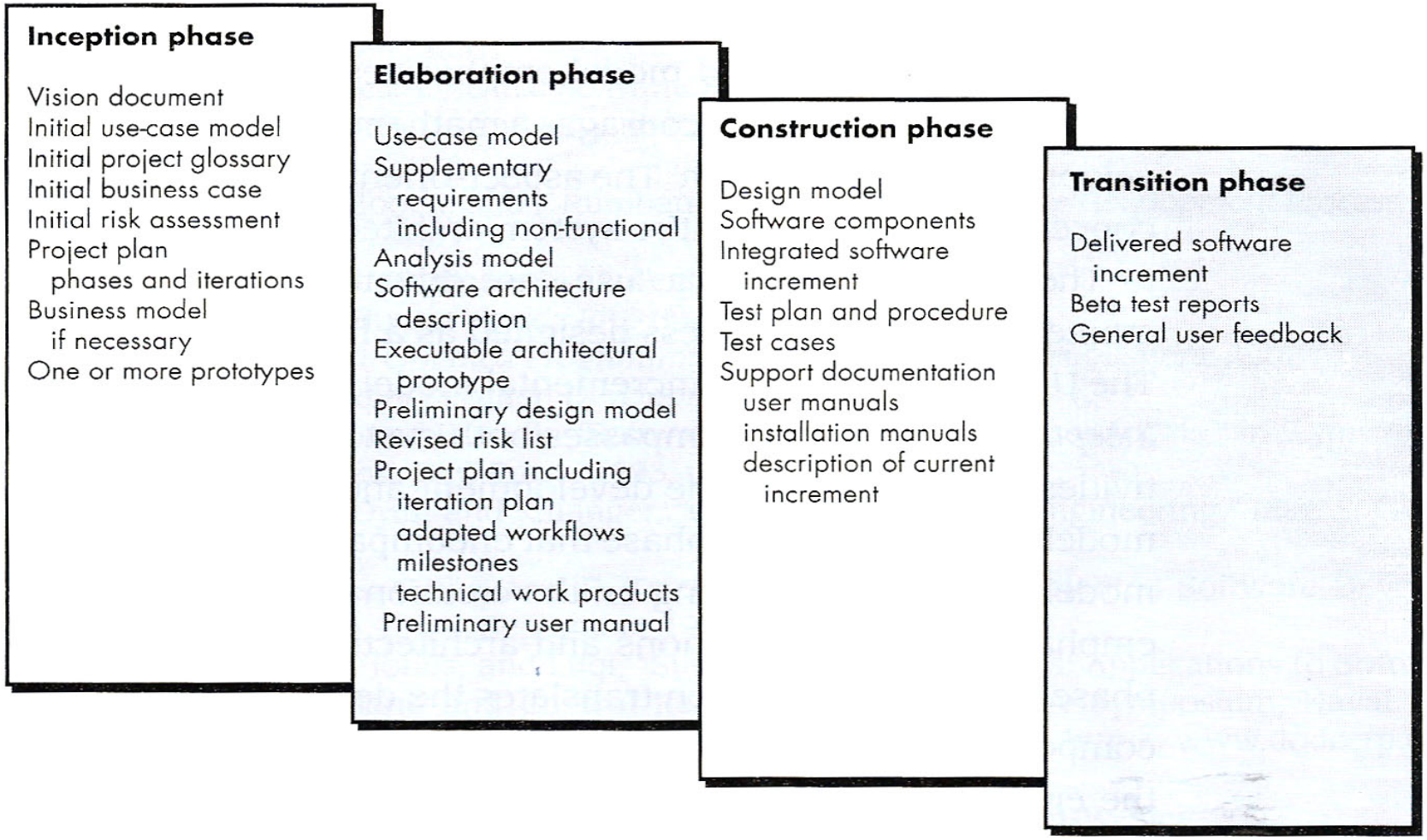
UNIFIED PROCESS (1)

- In some ways the Unified Process (UP) is an attempt to draw on the best features and characteristics of conventional software process models, but characterize them in a way that implements many of the best principles of agile software development.
- Jacobson, Rumbaugh, and Booch developed the Unified Process, a framework for object-oriented software engineering using UML.

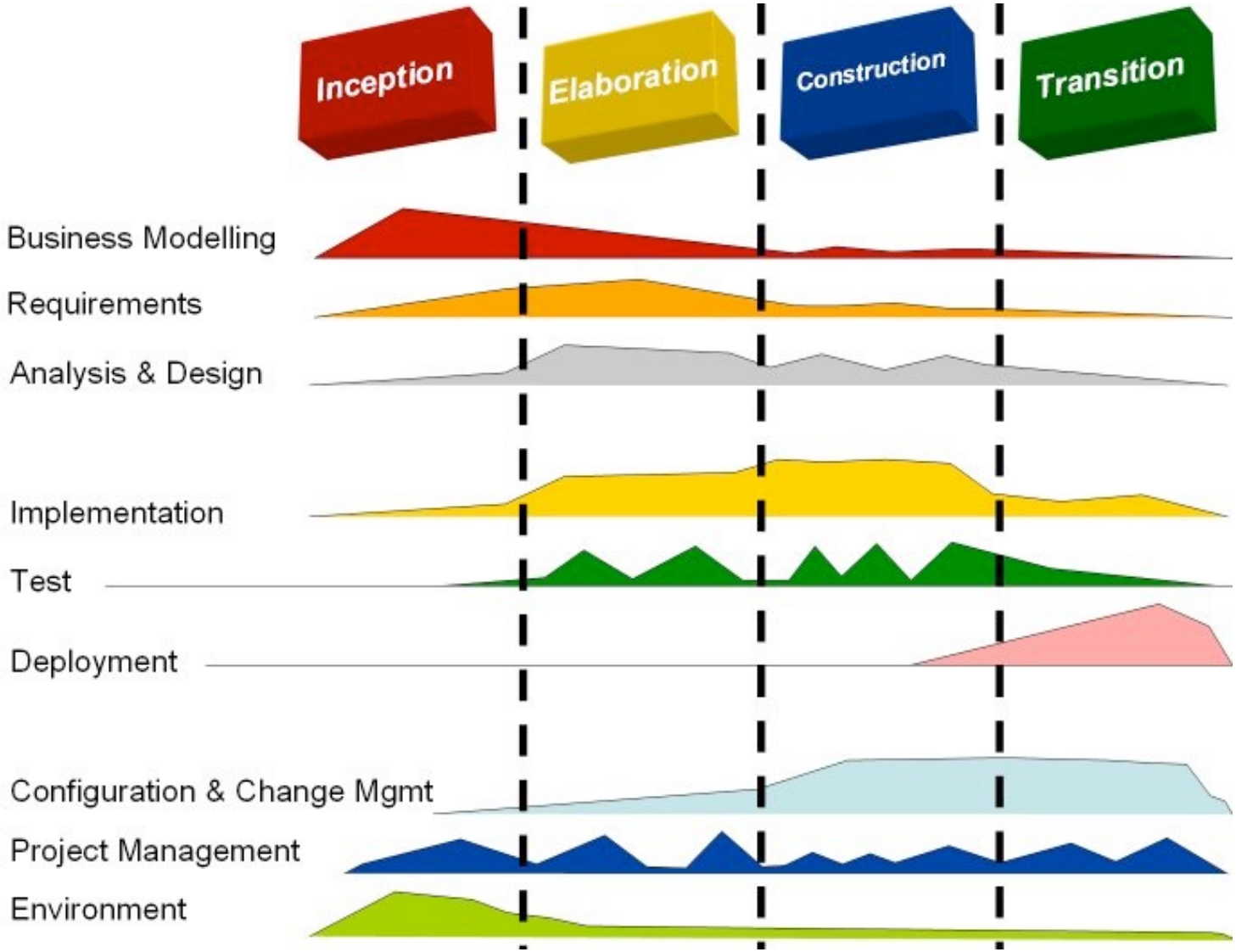
UNIFIED PROCESS (2)



UNIFIED PROCESS (3)



UNIFIED PROCESS (4)



An iteration in the elaboration phase

AGILE DEVELOPMENT

- These programming-centric methodologies have few rules and practices, all of which are fairly easy to follow. They focus on streamlining the SDLC by eliminating much of the modeling and documentation overhead, and the time spent on those tasks. Instead, projects emphasize simple, iterative application development.
- Examples of agile development methodologies include extreme programming (XP), Scrum, and the Dynamic Systems Development Method (DSDM).

THE PRINCIPLES OF AGILE METHODS

PRINCIPLE	DESCRIPTION
Customer Involvement	Customer should be closely involved throughout the development process. Their role is provide prioritize new system requirements and to evaluate the iterations of the system.
Incremental Delivery	The software is developed in increments with the customer specifing the requirements to be included in each increment.
People Not Process	The skills of development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace Change	Expect the system requirements to change and so design the system to accommodate these change.
Maintain Simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

AGILE METHOD APPLICABILITY

- Product development where a software company is developing a small or medium-sized product for sale.
- Custom system development within an organization, where there is a clear commitment from the customer to become involved in the development process and where there are not a lot of external rules and regulations that affect the software.
- Because of their focus on small, tightly-integrated teams, there are problems in scaling agile methods to large systems.

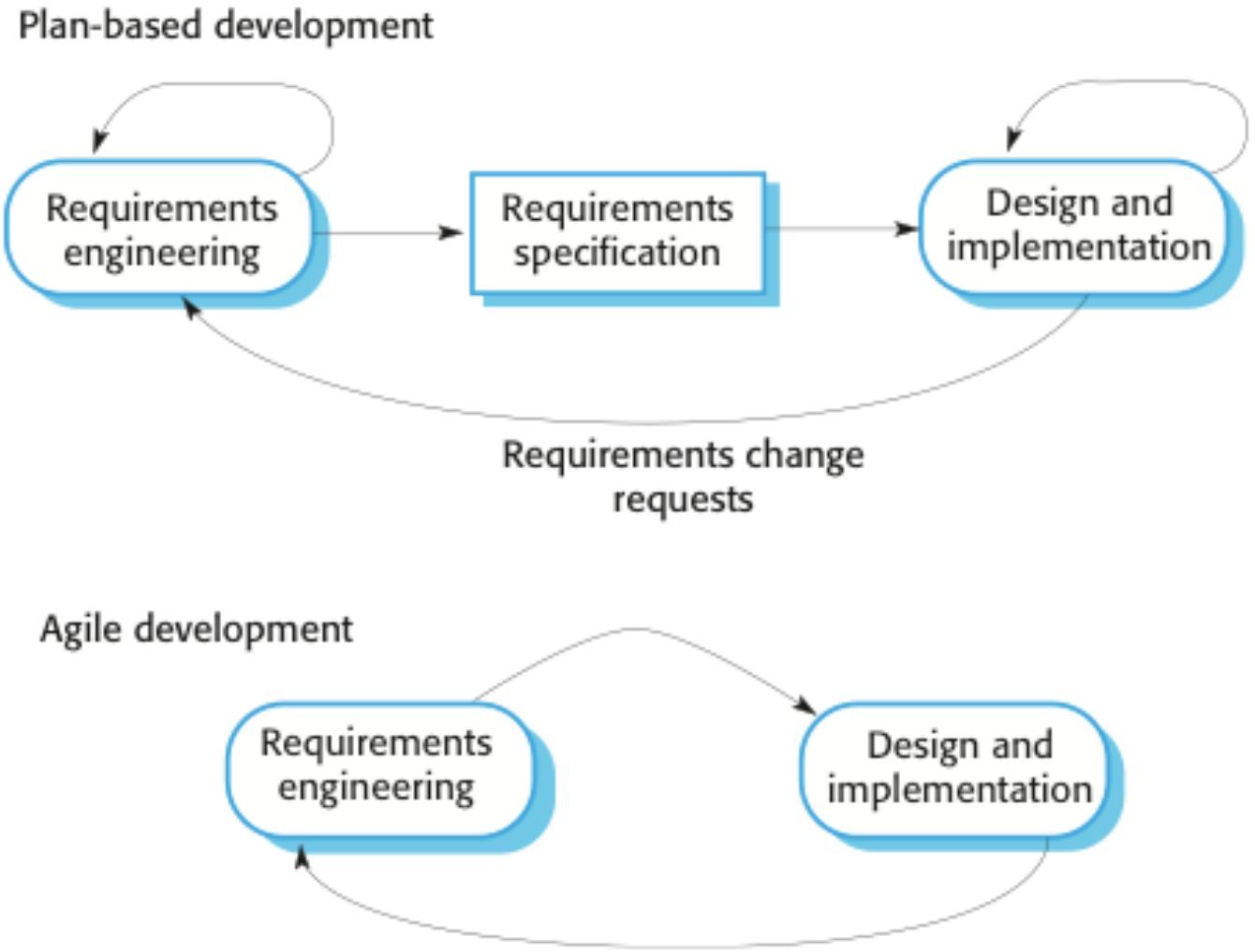
PROBLEMS WITH AGILE METHODS

- It can be difficult to keep the interest of customers who are involved in the process.
- Team members may be unsuited to the intense involvement that characterizes agile methods.
- Prioritizing changes can be difficult where there are multiple stakeholders.
- Maintaining simplicity requires extra work.
- Contracts may be a problem as with other approaches to iterative development.

AGILE METHODS AND SOFTWARE MAINTENANCE

- Most organizations spend more on maintaining existing software than they do on new software development. So, if agile methods are to be successful, they have to support maintenance as well as original development.
- Two key issues:
  - Are systems that are developed using an agile approach maintainable, given the emphasis in the development process of minimizing formal documentation?
  - Can agile methods be used effectively for evolving a system in response to customer change requests?
- Problems may arise if original development team cannot be maintained.

PLAN-DRIVEN AND AGILE SPECIFICATION



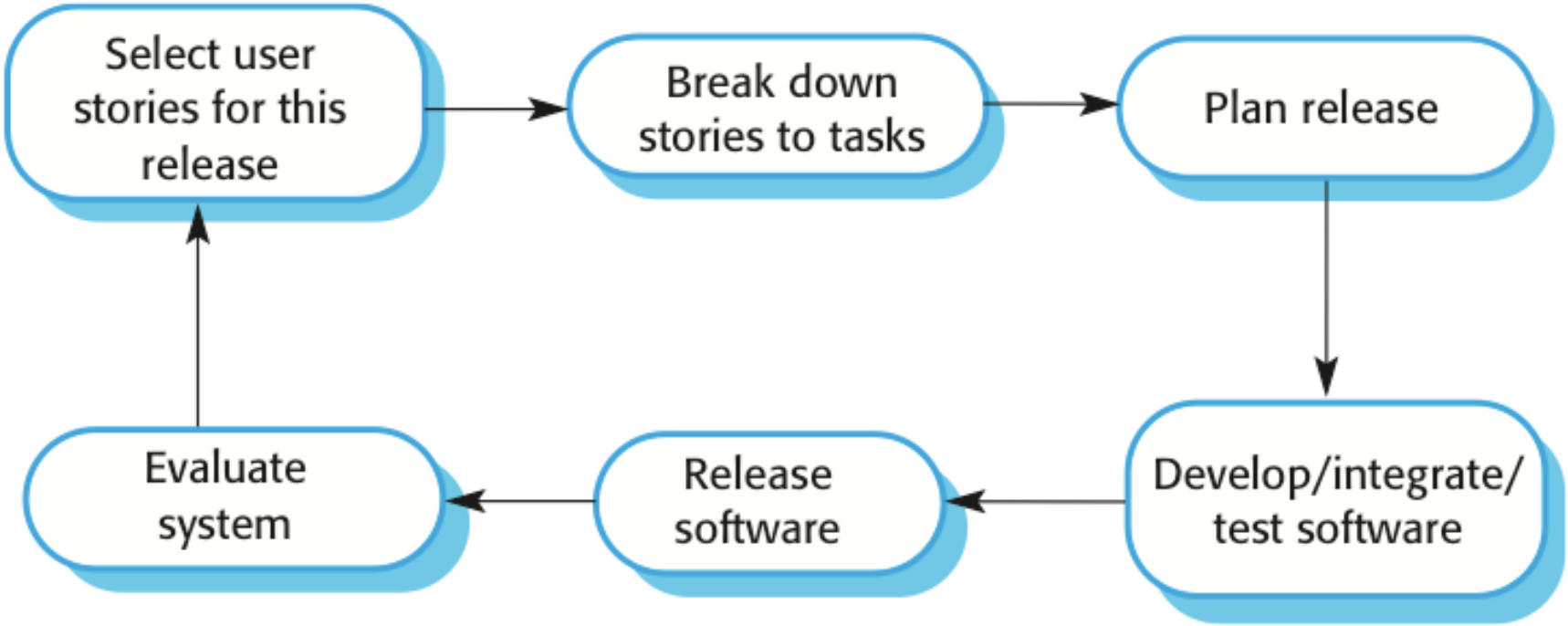
EXTREME PROGRAMMING (XP)

- Perhaps the best-known and most widely used agile method.
- Extreme Programming (XP) takes an *extreme* approach to iterative development.
  - New versions may be built several times per day.
  - Increments are delivered to customers every 2 weeks.
  - All tests must be run for every build and the build is only accepted if tests run successfully.

XP AND AGILE PRINCIPLES

- Incremental development is supported through small, frequent system releases.
- Customer involvement means full-time customer engagement with the team.
- People not process through pair programming, collective ownership and a process that avoids long working hours.
- Change supported through regular system releases.
- Maintaining simplicity through constant refactoring of code.

THE XP RELEASE CYCLE



EXTREME PROGRAMMING PRACTICES (1)

PRINCIPLE	DESCRIPTION
Incremental Planning	Requirements are recorded on story cards and stories to be included in a release are determined by the time available and their relative priority. The developers break this stories into development task.
Small Releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent incrementally add functionality to the first release.
Simple Design	Enough design is carried out to meet the current requirements and no more.
Test-First Development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keep the code simple and maintainable.

EXTREME PROGRAMMING PRACTICES (2)

PRINCIPLE	DESCRIPTION
Pair Programming	Developers work in pair, checking each other's work and providing the support to always do a good job.
Collective Ownership	The pair of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all of the code. Anyone can change anything.
Continuous Integration	As soon as the work on a task is complete, it is integrated into the whole system. After such integration, all the unit tests in the system must pass.
Sustainable Pace	Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium term productivity.
On-Site Customer	The representative of the end-user of the system should be available full-time for the XP team. In an XP process, the customer is a member of the development team an is responsible for bringing system requirements to the team for implement.

REQUIREMENTS SCENARIOS

- In XP, a customer or user is part of the XP team and is responsible for making decisions on requirements.
- User requirements are expressed as scenarios or user stories.
- These are written on cards and the development team break them down into implementation tasks. These tasks are the basis of schedule and cost estimates.
- The customer chooses the stories for inclusion in the next release based on their priorities and the schedule estimates.

XP AND CHANGE

- Conventional wisdom in software engineering is to design for change. It is worth spending time and effort anticipating changes as this reduces costs later in the life cycle.
- XP, however, maintains that this is not worthwhile as changes cannot be reliably anticipated.
- Rather, it proposes constant code improvement (refactoring) to make changes easier when they have to be implemented.

REFACTORING

- Programming team look for possible software improvements and make these improvements even where there is no immediate need for them.
- This improves the understandability of the software and so reduces the need for documentation.
- Changes are easier to make because the code is well-structured and clear.
- However, some changes requires architecture refactoring and this is much more expensive.

TESTING IN XP

- Testing is central to XP and XP has developed an approach where the program is tested after every change has been made.
- XP testing features:
  - Test-first development.
  - Incremental test development from scenarios.
  - User involvement in test development and validation.
  - Automated test harnesses are used to run all component tests each time that a new release is built.

TESTING FIRST DEVELOPMENT

- Writing tests before code clarifies the requirements to be implemented.
- Tests are written as programs rather than data so that they can be executed automatically. The test includes a check that it has executed correctly.
- All previous and new tests are run automatically when new functionality is added, thus checking that the new functionality has not introduced errors.

CUSTOMER INVOLVEMENT

- The role of the customer in the testing process is to help develop acceptance tests for the stories that are to be implemented in the next release of the system.
- The customer who is part of the team writes tests as development proceeds. All new code is therefore validated to ensure that it is what the customer needs.
- However, people adopting the customer role have limited time available and so cannot work full-time with the development team. They may feel that providing the requirements was enough of a contribution and so may be reluctant to get involved in the testing process.

TEST AUTOMATION

- Test automation means that tests are written as executable components before the task is implemented
  - These testing components should be stand-alone, should simulate the submission of input to be tested and should check that the result meets the output specification. An automated test framework (e.g. Junit) is a system that makes it easy to write executable tests and submit a set of tests for execution.
- As testing is automated, there is always a set of tests that can be quickly and easily executed
  - Whenever any functionality is added to the system, the tests can be run and problems that the new code has introduced can be caught immediately.

PAIR PROGRAMMING

- In XP, programmers work in pairs, sitting together to develop code.
- This helps develop common ownership of code and spreads knowledge across the team.
- It serves as an informal review process as each line of code is looked at by more than 1 person.
- It encourages refactoring as the whole team can benefit from this.
- Measurements suggest that development productivity with pair programming is similar to that of two people working independently.

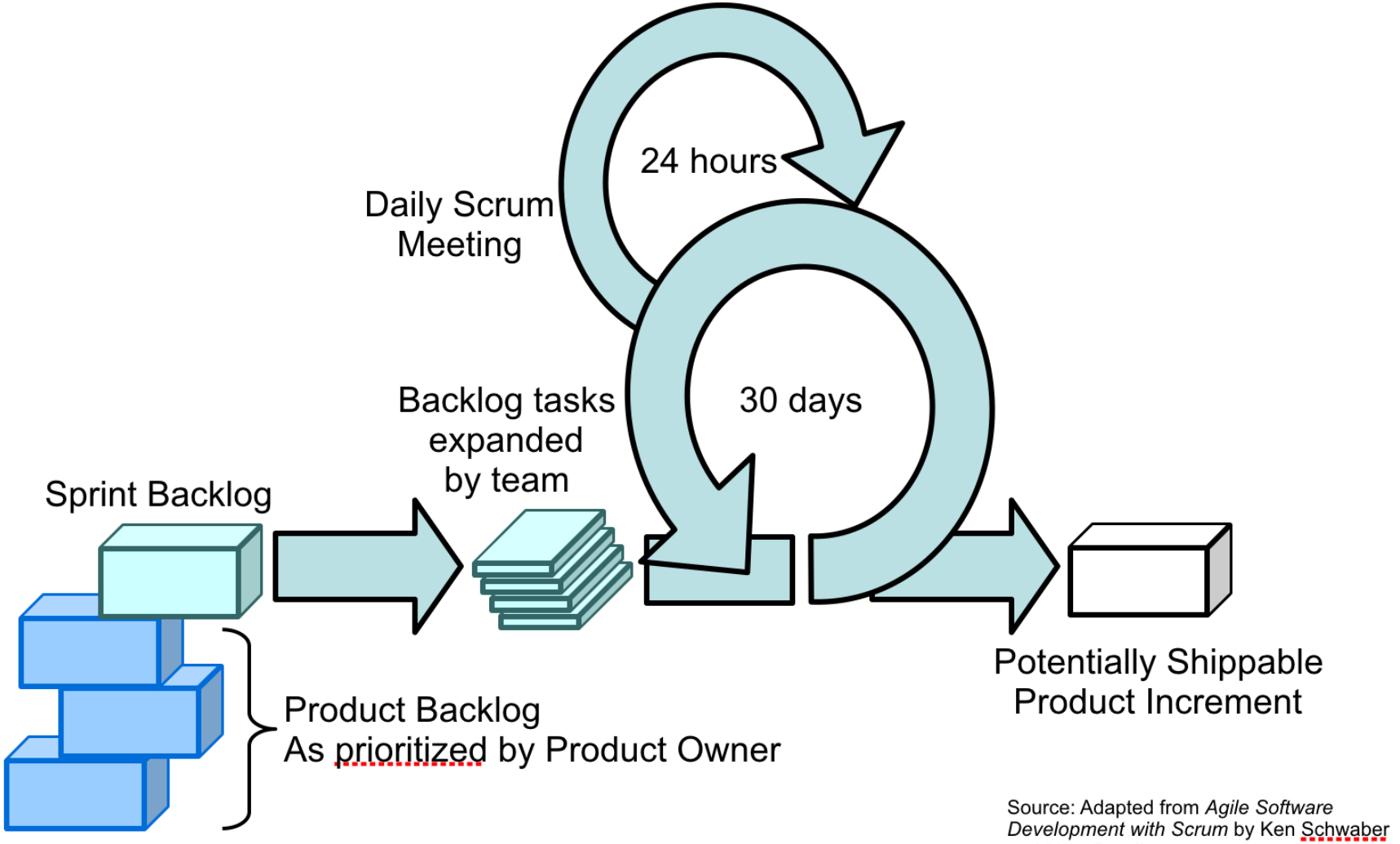
AGILE PROJECT MANAGEMENT

- The principal responsibility of software project managers is to manage the project so that the software is delivered on time and within the planned budget for the project.
- The standard approach to project management is plan-driven. Managers draw up a plan for the project showing what should be delivered, when it should be delivered and who will work on the development of the project deliverables.
- Agile project management requires a different approach, which is adapted to incremental development and the particular strengths of agile methods.

SCRUM

- The Scrum approach is a general agile method but its focus is on managing iterative development rather than specific agile practices.
- There are three phases in Scrum.
  - The initial phase is an outline planning phase where you establish the general objectives for the project and design the software architecture.
  - This is followed by a series of sprint cycles, where each cycle develops an increment of the system.
  - The project closure phase wraps up the project, completes required documentation such as system help frames and user manuals and assesses the lessons learned from the project.

THE SCRUM PROCESS



Source: Adapted from *Agile Software Development with Scrum* by Ken Schwaber and Mike Beedle.

THE SPRINT CYCLE (1)

- Sprints are fixed length, normally 2 - 4 weeks. They correspond to the development of a release of the system in XP.
- The starting point for planning is the product backlog, which is the list of work to be done on the project.
- The selection phase involves all of the project team who work with the customer to select the features and functionality to be developed during the sprint.

THE SPRINT CYCLE (2)

- Once these are agreed, the team organise themselves to develop the software. During this stage the team is isolated from the customer and the organisation, with all communications channeled through the so-called "*Scrum Master*".
  - The role of the Scrum master is to protect the development team from external distractions.
- At the end of the sprint, the work done is reviewed and presented to stakeholders. The next sprint cycle then begins.

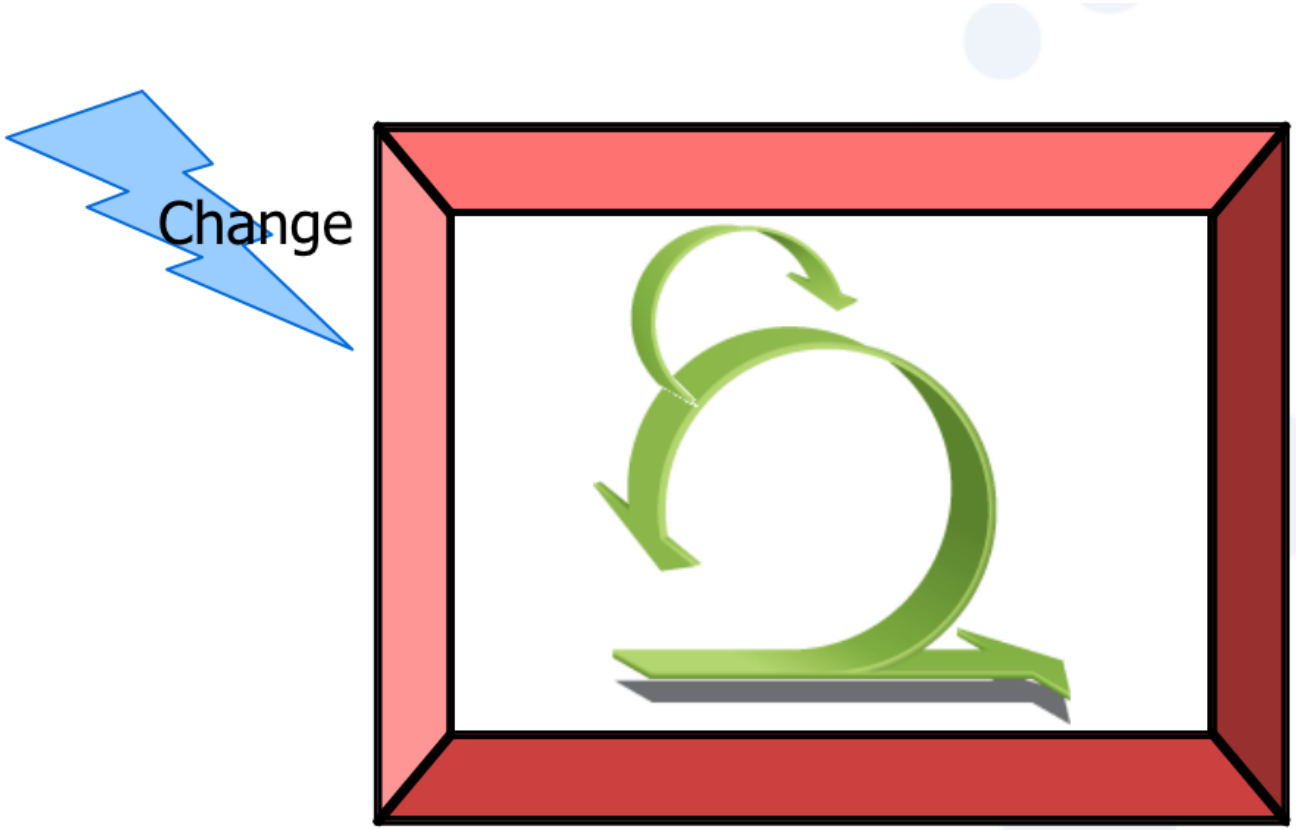
DAILY SCRUM MEETING

- Parameters
  - Daily, ~15 minutes, Stand-up
  - Anyone late pays a $1 fee
- Not for problem solving
  - Whole world is invited
  - Only team members, Scrum Master, product owner, can talk
  - Helps avoid other unnecessary meetings
- Three questions answered by each team member:
  - What did you do yesterday?
  - What will you do today?
  - What obstacles are in your way?

THE SPRINT REVIEW

- Team presents what it accomplished during the sprint
- Typically takes the form of a demo of new features or underlying architecture
- Informal
  - 2-hour prep time rule
  - No slides
- Whole team participates

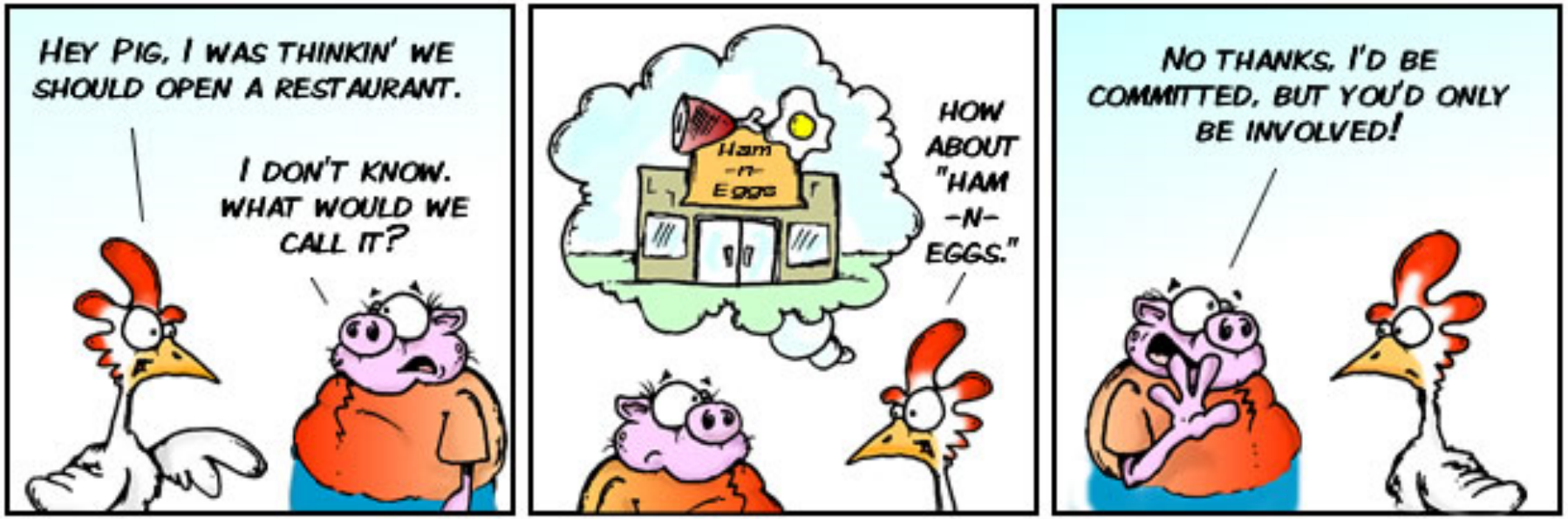
NO CHANGES DURING A SPRINT



TEAMWORK IN SCRUM

- The “Scrum master” is a facilitator who arranges daily meetings, tracks the backlog of work to be done, records decisions, measures progress against the backlog and communicates with customers and management outside of the team.
- The whole team attends short daily meetings where all team members share information, describe their progress since the last meeting, problems that have arisen and what is planned for the following day.
- This means that everyone on the team knows what is going on and, if problems arise, can re-plan short-term work to cope with them.

"PIGS" AND "CHICKENS"



By Clark & Vizdos

SCRUM BENEFITS

- The product is broken down into a set of manageable and understandable chunks.
- Unstable requirements do not hold up progress.
- The whole team have visibility of everything and consequently team communication is improved.
- Customers see on-time delivery of increments and gain feedback on how the product works.
- Trust between customers and developers is established and a positive culture is created in which everyone expects the project to succeed.

SCALING AGILE METHODS

- Agile methods have proved to be successful for small and medium sized projects that can be developed by a small co-located team.
- It is sometimes argued that the success of these methods comes because of improved communications which is possible when everyone is working together.
- Scaling up agile methods involves changing these to cope with larger, longer projects where there are multiple development teams, perhaps working in different locations.

KEY POINTS (1)

- Agile methods are incremental development methods that focus on rapid development, frequent releases of the software, reducing process overheads and producing high-quality code. They involve the customer directly in the development process.
- The decision on whether to use an agile or a plan-driven approach to development should depend on the type of software being developed, the capabilities of the development team and the culture of the company developing the system.
- Extreme programming is a well-known agile method that integrates a range of good programming practices such as frequent releases of the software, continuous software improvement and customer participation in the development team.

KEY POINTS (2)

- A particular strength of extreme programming is the development of automated tests before a program feature is created. All tests must successfully execute when an increment is integrated into a system.
- The Scrum method is an agile method that provides a project management framework. It is centered round a set of sprints, which are fixed time periods when a system increment is developed.
- Scaling agile methods for large systems is difficult. Large systems need up-front design and some documentation.

CASE STUDY: AN EXPENSIVE FALSE START (1)

- A real-estate group in the federal government cosponsored a data warehouse with the IT department. A formal proposal was written by IT in which costs were estimated at $800,000, the project duration was estimated to be eight months, and the responsibility for funding was defined as the business unit's.
- The IT department proceeded with the project before hearing whether the proposal was ever accepted. The project actually lasted two years because requirements gathering took nine months instead of one and a half, the planned user base grew from 200 to 2,500, and the approval process to buy technology for the project took a year.

Source: "Data Warehousing Failure: Case Studies and Findings" The Journal of Data Warehousing, by Hugh J. Watson et al, 4 (1), 1999, pp. 44–54.

CASE STUDY: AN EXPENSIVE FALSE START (2)

- Three weeks prior to technical delivery, the IT Director canceled the project. This failed Endeavor cost the organization $2.5 million.

QUESTION:

1. Why did this system fail?
2. Why would a company spend money and time on a project and then cancel it?
3. What could have been done to prevent this?

Source: "Data Warehousing Failure: Case Studies and Findings" The Journal of Data Warehousing, by Hugh J. Watson et al, 4 (1), 1999, pp. 44–54.