1. Software and Software Engineering

Abdus Sattar

Assistant Professor

Department of Computer Science and Engineering

Daffodil International University

Email: abdus.cse@diu.edu.bd

Discussion Topics

- Appreciate Software Engineering
- Nature of Software Engineering
- Software Applications
- Legacy of Software Engineering
- Software Quality
- Software Quality and Stakeholders
- A Layered Technology
- Software Process
- Software Myths

What is Software?

- *Software is:*
  - *Instructions (computer programs) that when executed provide desired features, function, and performance;*
  - *Data structures that enable the programs to adequately manipulate information and*
  - *Documentation that describes the operation and use of the programs.*

Characteristics/Nature of Software

- **Software is developed or engineered, it is not manufactured in the classical sense.**
- **Software is intangible(unable to touch)**
  - Hard to understand development effort
- **Software is easy to reproduce**
  - Cost is in its *development*
- **The industry is labor-intensive**
  - Hard to automate

Characteristics/Nature of Software

- **Untrained people can hack something together**
  - Quality problems are hard to notice
- **Software is easy to modify**
  - People make changes without fully understanding it
- **Software does not 'wear out'**
  - Relationship between failure rate and time.
  - the failure rate as a function of time for hardware

Characteristics/Nature of Software

- **Conclusions**
  - Much software has poor design and is getting worse
  - We have to learn to ‘engineer’ software

Software Applications

- **System Software :**
  - Operating system, drivers, networking software, telecommunications processors, Compilers.
- **Application Software:**
  - Microsoft Office, Excel and Outlook, Google Chrome, Mozilla Firefox and Skype. Games and mobile applications such as "Clash of Clans," SoundCloud, Spotify and Uber, are also considered application software. Other specific examples include Steam, "Minecraft," Adobe Reader and Photoshop.
- **Engineering/scientific software:**
  - Computer-aided Design and Computer-aided Manufacturing Software, Civil Engineering and Architectural Software , Electrical Engineering software, Geographic Information Systems Software, Simulation software, Interactive Software.
- **Embedded Software :**
  - key pad control for a microwave oven, fuel control, dashboard displays, and braking systems, control and monitoring system.

Software Applications (Cont..)

- **Product-line software/Data Processing System:**
  - Inventory control products, word processing, spreadsheets, computer graphics, multimedia, entertainment, database management, and personal and business financial applications
- **WebApps (Web applications)**
  - Integrated with corporate databases and business applications: Booking application, Chatting application, Upload, E-Business, E-Commerce application.
- **AI software**
  - Include robotics, expert systems, pattern recognition (image and voice), artificial neural networks, theorem, proving, and game playing.
- **Gaming Software**
- **Mobile Device Software**

Software—New Categories

- Open world computing—pervasive/widespread, distributed computing
- Ubiquitous computing—wireless networks.
- Net sourcing—the Web as a computing engine.
- Open source—”free” source code open to the computing community (a blessing, but also a potential curse!)

Legacy Software

- Legacy implies that the software is out of date or in need of replacement, however it may be in good working order so the business or individual owner does not want to upgrade or update the software.

Legacy Software

Why must it change?

- software must be adapted to meet the needs of new computing environments or technology.
- software must be enhanced to implement new business requirements.
- software must be extended to make it interoperable with other more modern systems or databases.
- software must be re-architected to make it viable within a network environment.

Characteristics of WebApps

- **Network intensiveness.** A WebApp resides on a network and must serve the needs of a diverse community of clients.
- **Concurrency.** A large number of users may access the WebApp at one time.
- **Unpredictable load.** The number of users of the WebApp may vary by orders of magnitude from day to day.
- **Performance.** If a WebApp user must wait too long (for access, for server-side processing, for client-side formatting and display), he or she may decide to go elsewhere.
- **Availability.** Although expectation of 100 percent availability is unreasonable, users of popular WebApps often demand access on a "24/7/365" basis.

Characteristics of WebApps(Cont…)

- **Data driven.** The primary function of many WebApps is to use hypermedia to present text, graphics, audio, and video content to the end-user.
- **Content sensitive.** The quality and aesthetic nature of content remains an important determinant of the quality of a WebApp.
- **Continuous evolution.** Unlike conventional application software that evolves over a series of planned, chronologically-spaced releases, Web applications evolve continuously.
- **Immediacy.** Although *immediacy*—the compelling need to get software to market quickly—is a characteristic of many application domains, WebApps often exhibit a time to market that can be a matter of a few days or weeks.
- **Security.** Because WebApps are available via network access, it is difficult, if not impossible, to limit the population of end-users who may access the application.
- **Aesthetics.** An undeniable part of the appeal of a WebApp is its look and feel.

Software Engineering

- **The IEEE definition:**
  - *Software Engineering:*
    - *The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.*
    - *The study of approaches as in (1).*

Software Engineering (Cont..)

- The process of solving customers' problems by the systematic development and evolution of large, high-quality software systems within cost, time and other constraints

Software Engineering (Cont..)

- **Solving customers' problems**
  - The goal
  - Sometimes the solution is to *buy*, *not build*
  - Adding unnecessary features often makes software worse
  - Software engineers must *communicate effectively* to identify and understand the problem

Software Engineering (Cont..)

- Systematic development and evolution
  - An engineering process involves applying *well understood techniques* in a organized and *disciplined* way
  - Many well-accepted practices have been formally standardized
    - e.g. by the IEEE or ISO
  - Most development work is *evolution*

Software Engineering (Cont..)

- **Large, high quality software systems**
  - Software engineering techniques are needed because large systems *cannot be completely understood* by one person
  - Teamwork and co-ordination are required
  - Key challenge: Dividing up the work and ensuring that the parts of the system work properly together
  - The end-product must be of sufficient quality

Software Engineering (Cont..)

- **Cost, time and other constraints**
  - Finite resources
  - The benefit must outweigh the cost
  - Others are competing to do the job cheaper and faster
  - Inaccurate estimates of cost and time have caused many project failures

Software Quality

- **Usability**
  - Users can learn it and fast and get their job done easily
- **Efficiency**
  - It doesn't't waste resources such as CPU time and memory
- **Reliability**
  - It does what it is required to do without failing
- **Maintainability**
  - It can be easily changed
- **Reusability**
  - Its parts can be used in other projects, so reprogramming is not needed

Software Quality and Stakeholders

Customer (those who pay):
solves problems at
an acceptable cost in
terms of money paid and
resources used

User:
easy to learn;
efficient to use;
helps get work done

QUALITY
SOFTWARE

Developer:
easy to design;
easy to maintain;
easy to reuse its parts

Development manager:
sells more and
pleases customers
while costing less
to develop and maintain

A Layered Technology

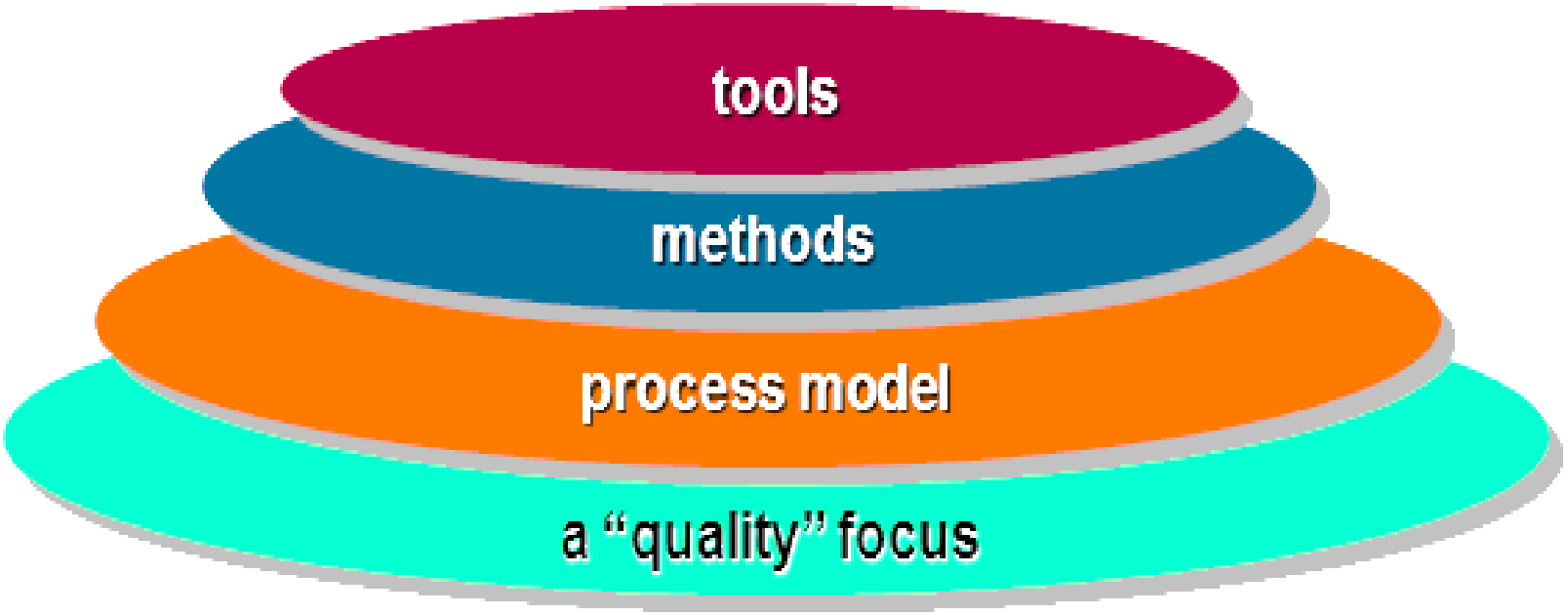


Fig: Software engineering layers

Software Process

- A ***process*** is a collection of activities, actions, and tasks that are performed when some work product is to be created.
- An ***activity*** strives to achieve a broad objective (e.g., communication with stakeholders) and is applied regardless of the application domain, size of the project, complexity of the effort, or degree of rigor with which software engineering is to be applied.
- An ***action*** (e.g., architectural design) encompasses a set of tasks that produce a major work product (e.g., an architectural design model).
- A ***task*** focuses on a small, but well-defined objective (e.g., conducting a unit test) that produces a tangible outcome.

The Essence of Practice

- **George Polya outlined The essence of software engineering practice:**
  - *Understand the problem* (communication and analysis).
  - *Plan a solution* (modeling and software design).
  - *Carry out the plan* (code generation).
  - *Examine the result for accuracy* (testing and quality assurance).

Software Myths

- Pressman describes a number of common beliefs or myths that software managers, customers, and developers believe falsely.
- He describes these myths as ``misleading attitudes that have caused serious problems." We look at these myths to see why they are false, and why they lead to trouble.
  - Affect managers, customers (and other non-technical stakeholders) and practitioners
  - Are believable because they often have elements of truth,
  - *but ...*
  - Invariably lead to bad decisions,
  - *therefore ...*
  - Insist on reality as you navigate your way through software engineering

Hooker's General Principles

- 1: *The Reason It All Exists*
- 2: *KISS (Keep It Simple, Stupid!)*
- 3: *Maintain the Vision*
- 4: *What You Produce, Others Will Consume*
- 5: *Be Open to the Future*
- 6: *Plan Ahead for Reuse*
- 7: *Think!*

References:

1. **Software Engineering by Ian Sommerville,** 9th edition, Addison-Wesley, 2011

2. **Software Engineering A practitioner's Approach** by Roger S. Pressman, 7th edition, McGraw Hill, 2010.

2. Process Models

Abdus Sattar

Assistant Professor

Department of Computer Science and Engineering

Daffodil International University

Email: abdus.cse@diu.edu.bd

Topics Covered

- Software Process
- Process Model
- A generic Process Model
- Software Framework Activities
- Software Process Model
- Selection of Process Model

Definition of Software Process

- ***Software Process:***
  - A **framework** for the activities, actions, and tasks that are required to build high-quality software.
  - SP defines the approach that is taken as software is engineered.
  - Is not equal to software engineering, which also encompasses **technologies** that populate the process– technical methods and automated tools.
- ***Process Model :***
  - A Process Model describes the **sequence of phases** for the entire lifetime of a product. Therefore it is sometimes also called Product Life Cycle.
  - This covers everything from the initial commercial idea until the final de-installation or disassembling of the product after its use.

What / who / why is Process Models?

- ***What:*** Go through a series of predictable steps--- a road map that helps you create a timely, high-quality results.
- ***Who:*** Software engineers and their managers, clients also. People adapt the process to their needs and follow it.
- ***Why:*** Provides stability, control, and organization to an activity that can if left uncontrolled, become quite chaotic. However, modern software engineering approaches must be agile and demand ONLY those activities, controls and work products that are appropriate.
- ***What Work products:*** Programs, documents, and data
- ***What are the steps:*** The process you adopt depends on the software that you are building. One process might be good for aircraft avionic system, while an entirely different process would be used for website creation.
- ***How to ensure right:*** A number of software process assessment mechanisms that enable us to determine the maturity of the software process. However, the quality, timeliness and long-term viability of the software are the best indicators of the efficacy of the process you use.

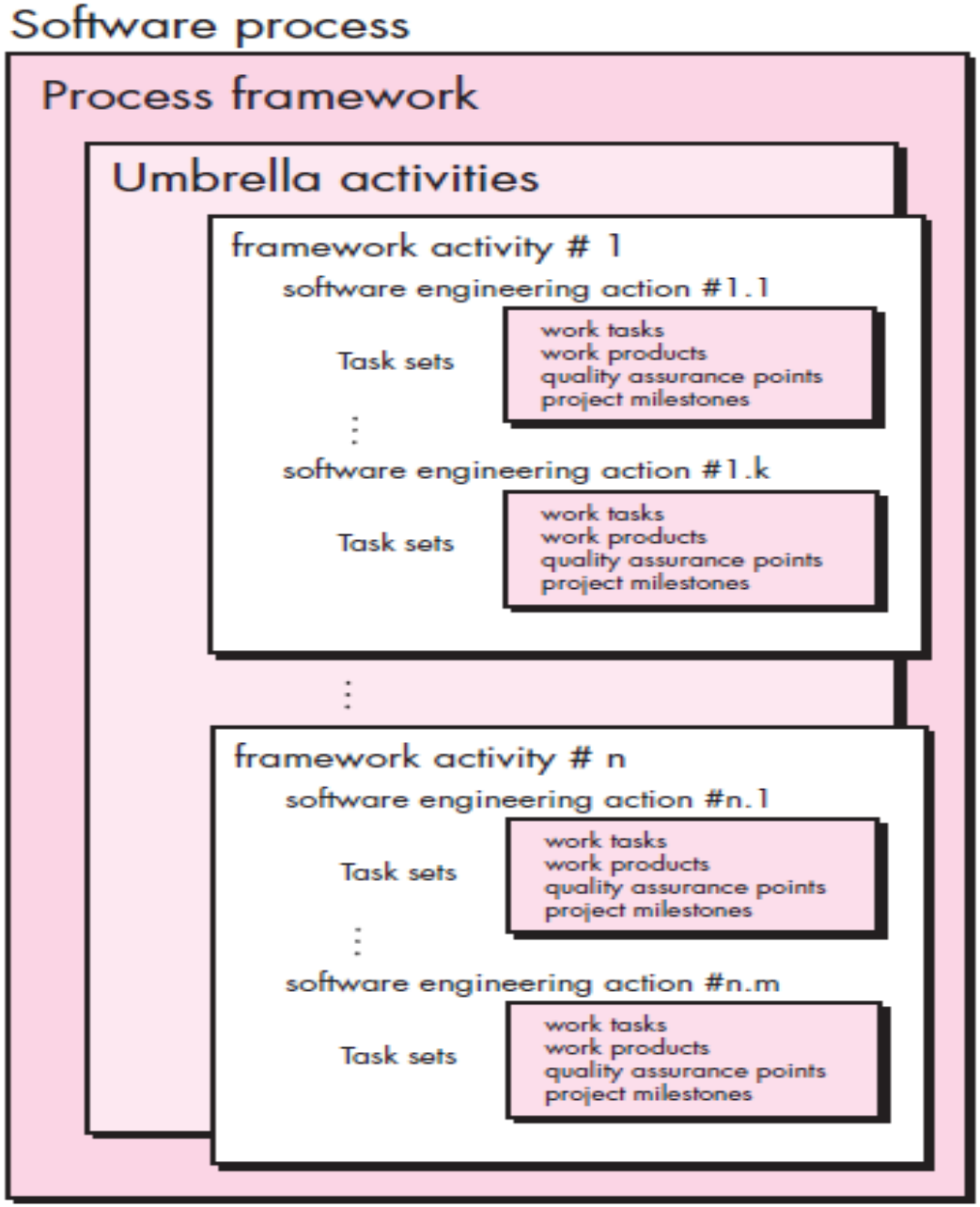
A Generic Process Model

- As we discussed before, a generic process framework for software engineering defines five framework activities- communication, planning, modeling, construction, and deployment.
- In addition, a set of umbrella activities- project tracking and control, risk management, quality assurance, configuration management, technical reviews, and others are applied throughout the process.

A Generic Process Model

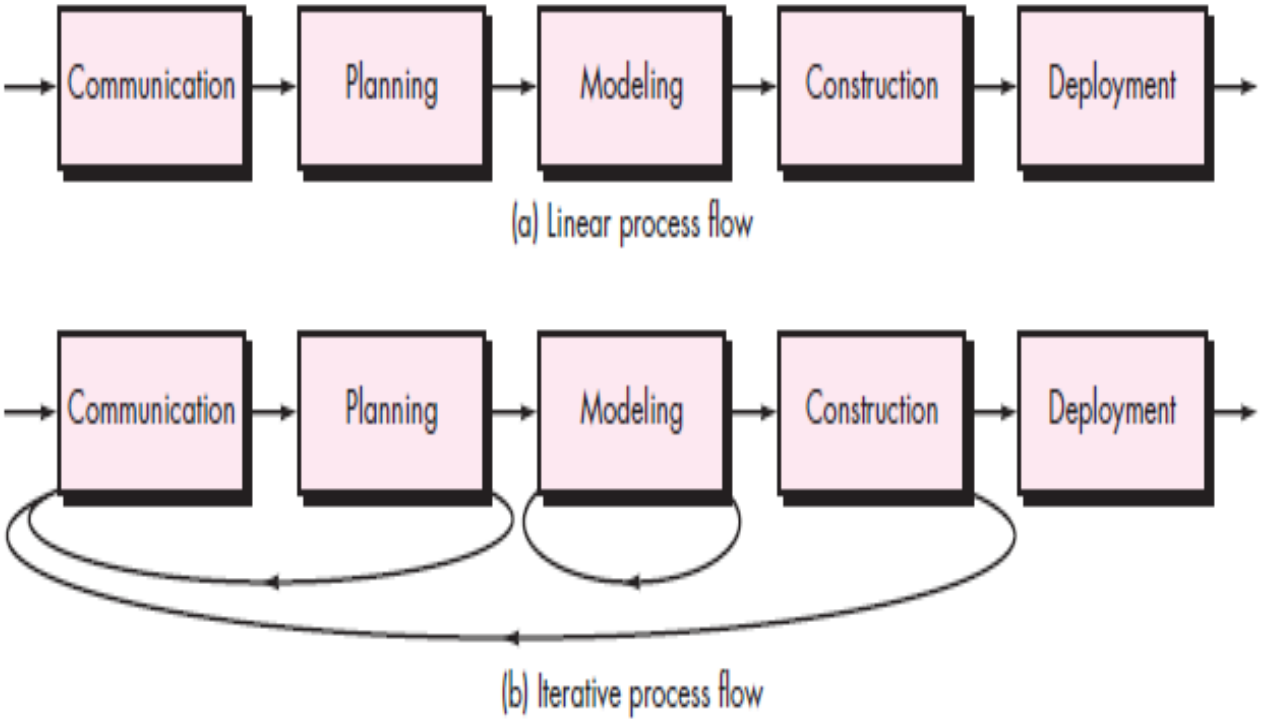
- ***Communication :*** This activity involves heavy communication with customers and other stakeholders in order to gather requirements and other related activities.
- ***Planning :*** Here a plan to be followed will be created which will describe the technical tasks to be conducted, risks, required resources, work schedule etc.
- ***Modeling :*** A model will be created to better understand the requirements and design to achieve these requirements.
- ***Construction :*** Here the code will be generated and tested.
- ***Deployment :*** Here, a complete or partially complete version of the software is represented to the customers to evaluate and they give feedbacks based on the evaluation.

A Generic Process Model



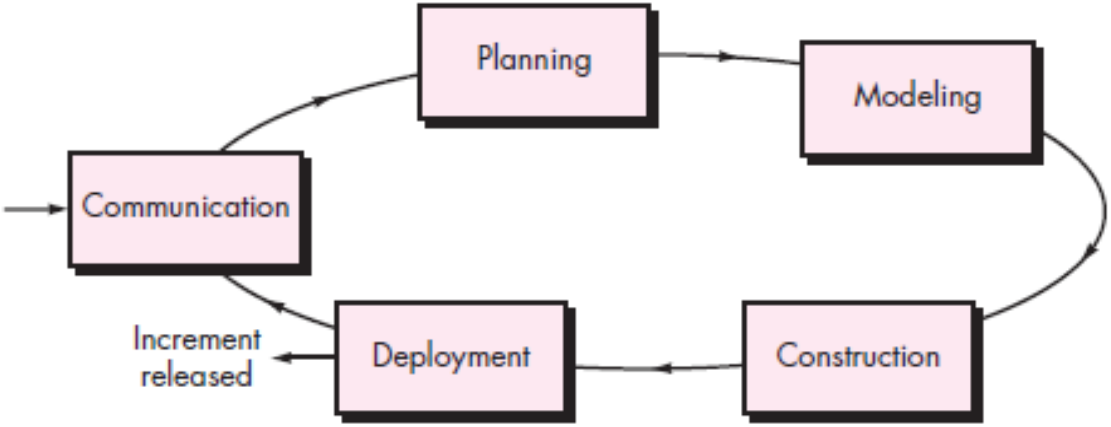
Framework Activities

Process Flow

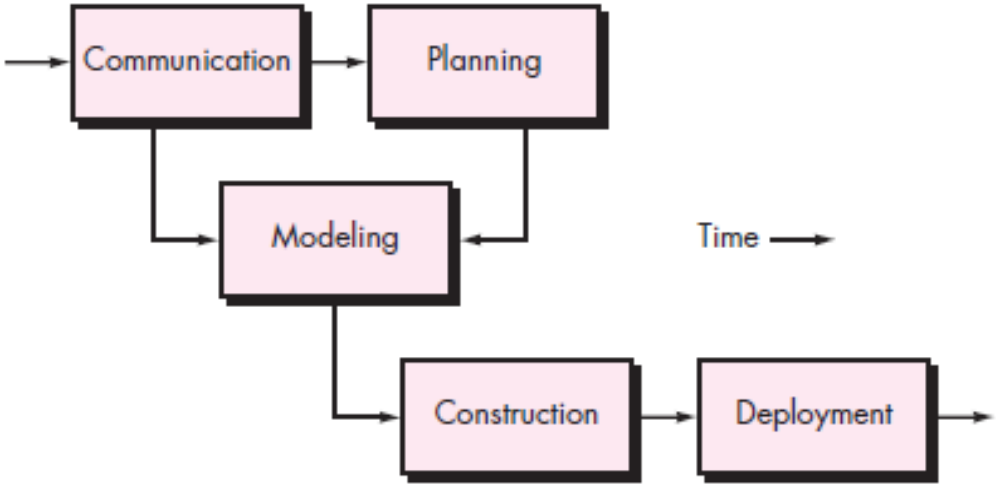


Process Flow

Process Flow



(c) Evolutionary process flow



(d) Parallel process flow

Process Flow

- Linear process flow executes each of the five activities in sequence.
- An iterative process flow repeats one or more of the activities before proceeding to the next.
- An evolutionary process flow executes the activities in a circular manner. Each circuit leads to a more complete version of the software.
- A parallel process flow executes one or more activities in parallel with other activities (modeling for one aspect of the software in parallel with construction of another aspect of the software.

Identifying a Task Set

- For example, a small software project requested by one person with simple requirements, the communication activity might encompass little more than a phone all with the stakeholder. Therefore, the only necessary action is phone conversation, the work tasks of this action are:

1. ***Make contact with stakeholder via telephone.***
2. ***Discuss requirements and take notes.***
3. ***Organize notes into a brief written statement of requirements.***
4. ***E-mail to stakeholder for review and approval.***

Example of a Task Set for Small Project

- The task sets for Requirements gathering action for a **simple** project may include:

1. ***Make a list of stakeholders for the project.***
2. ***Invite all stakeholders to an informal meeting.***
3. ***Ask each stakeholder to make a list of features and functions required.***
4. ***Discuss requirements and build a final list.***
5. ***Prioritize requirements.***
6. ***Note areas of uncertainty***.

Example of a Task Set for a big project

- **The task sets for Requirements gathering action for a big project may include:**
  1. ***Make a list of stakeholders for the project.***
  2. ***Interview each stakeholders separately to determine overall wants and needs.***
  3. ***Build a preliminary list of functions and features based on stakeholder input.***
  4. ***Schedule a series of facilitated application specification meetings.***
  5. ***Conduct meetings.***
  6. ***Produce informal user scenarios as part of each meeting.***
  7. ***Refine user scenarios based on stakeholder feedback.***
  8. ***Build a revised list of stakeholder requirements.***
  9. ***Use quality function deployment techniques to prioritize requirements.***
  10. ***Package requirements so that they can be delivered incrementally.***
  11. ***Note constraints and restrictions that will be placed on the system.***
  12. ***Discuss methods for validating the system.***

Process Patterns

- A *process pattern*
  - describes a process-related problem that is encountered during software engineering work,
  - identifies the environment in which the problem has been encountered, and
  - suggests one or more proven solutions to the problem.

Process Patterns

- Template for describing a process pattern:
  - ***Pattern Name:***
    - The pattern name is given a meaningful name that describe its function within software process.
    - Example: Customer-Communication
  - ***Intent:***
    - The objective of this pattern is describe briefly.
    - Example: Intent of the communication customer to connection establish with customer.
  - ***Type:***
    - The pattern type is specified.
    - Three types:
      - **Task Pattern** - action or work task
      - **Stage Pattern** – frame activity.
      - **Phase Pattern** – sequence of frame.

Process Patterns (Cont…)

- ***Initial Context:***
  - Describe which pattern are applies
  - Prior to the initialization of the pattern.
- ***Resulting Context:***
  - The condition that will result once the pattern has been successfully implementers are describe.
- ***Related Pattern:***
  - A list of process pattern that are directly related to this one are provided as a hierarchy on in some other diagrammatic form.

Software Process Model

- A Process Model describes the sequence of phases for the entire lifetime of a product.
- It is sometimes also called Product Life Cycle.
- This covers everything from the initial commercial idea until the final de-installation or disassembling of the product after its use
- All software process models can accommodate the generic framework activities.

Waterfall Model

Requirement
- Requirement Doc.
- Prepare Use Cases

Design
- Software architecture
- Map the stakeholders

Implementation
- Construct the software
- Data storage & retrieval

Verification
- Install
- Test and Debug

Maintenance
- Check errors
- Optimize capabilities

Waterfall Model (Cont..)

- The *first published model* of the software development process was derived from more general system engineering processes (Royce, 1970).
- This model is known as the '*waterfall model*' or software life cycle
- *The waterfall model* This takes the fundamental process activities of specification, development, validation, and evolution and represents them as separate process phases such as requirements specification, software design, implementation, testing, and so on.

Waterfall Model (Cont..)

- **The sequential phases in Waterfall model are:**
  - ***Requirement Gathering and analysis:*** All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification doc.
  - ***System Design:*** The requirement specifications from first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture.
  - ***Implementation:*** With inputs from system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality which is referred to as Unit Testing.
  - ***Integration and Testing:*** All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
  - ***Deployment of system:*** Once the functional and non functional testing is done, the product is deployed in the customer environment or released into the market.
  - ***Maintenance:*** There are some issues which come up in the client environment. To fix those issues patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

Waterfall Model (Cont..)

- **Problems:**
  - Real projects rarely follow the sequential flow that the model proposes. Although the linear model can accommodate iteration, it does so indirectly. As a result, changes can cause confusion as the project team proceeds.
  - It is often difficult for the customer to state all requirements explicitly. The waterfall model requires this and has difficulty accommodating the natural uncertainty that exists at the beginning of many projects.
  - The customer 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.

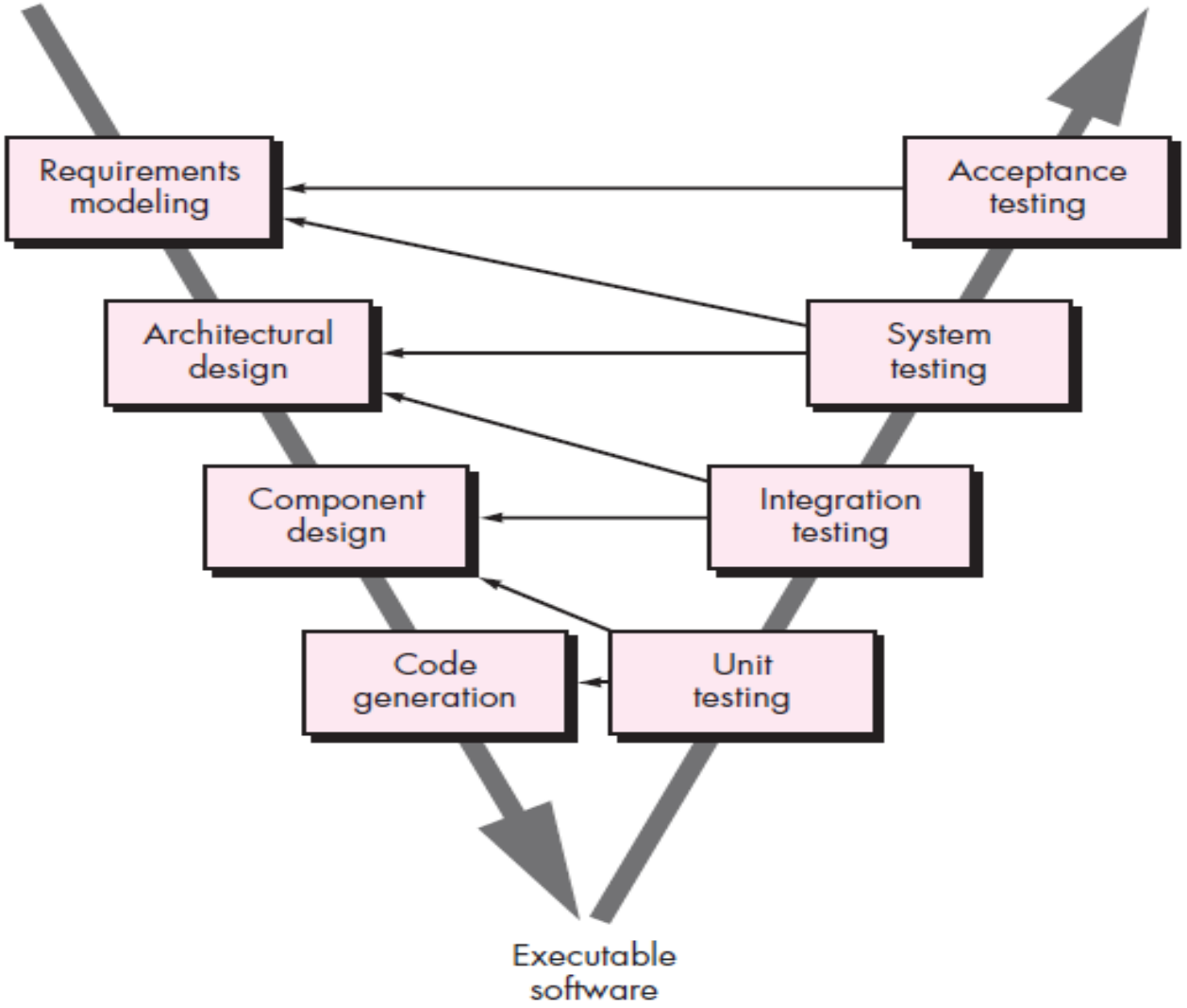
Waterfall Model (Cont..)

- **Problems:**
  - Rarely linear, iteration needed.
  - Hard to state all requirements explicitly. Blocking state.
  - Code will not be released until very late.

- **Waterfall Model Application**
  - Every software developed is different and requires a suitable SDLC approach to be followed based on the internal and external factors. Some situations where the use of Waterfall model is most appropriate are:
  - Requirements are very well documented, clear and fixed.
  - Product definition is stable.
  - Technology is understood and is not dynamic.
  - There are no ambiguous requirements.
  - Ample resources with required expertise are available to support the product.
  - The project is short.


1
Assess Needs
2
Design Specifications
3
Design / Develop / Test software
4
Implement Systems
5
Support Operations
6
Evaluate Performance

The V-Model


Requirements modeling
Architectural design
Component design
Code generation
Unit testing
Integration testing
System testing
Acceptance testing
Executable software

The V-Model(Cont..)

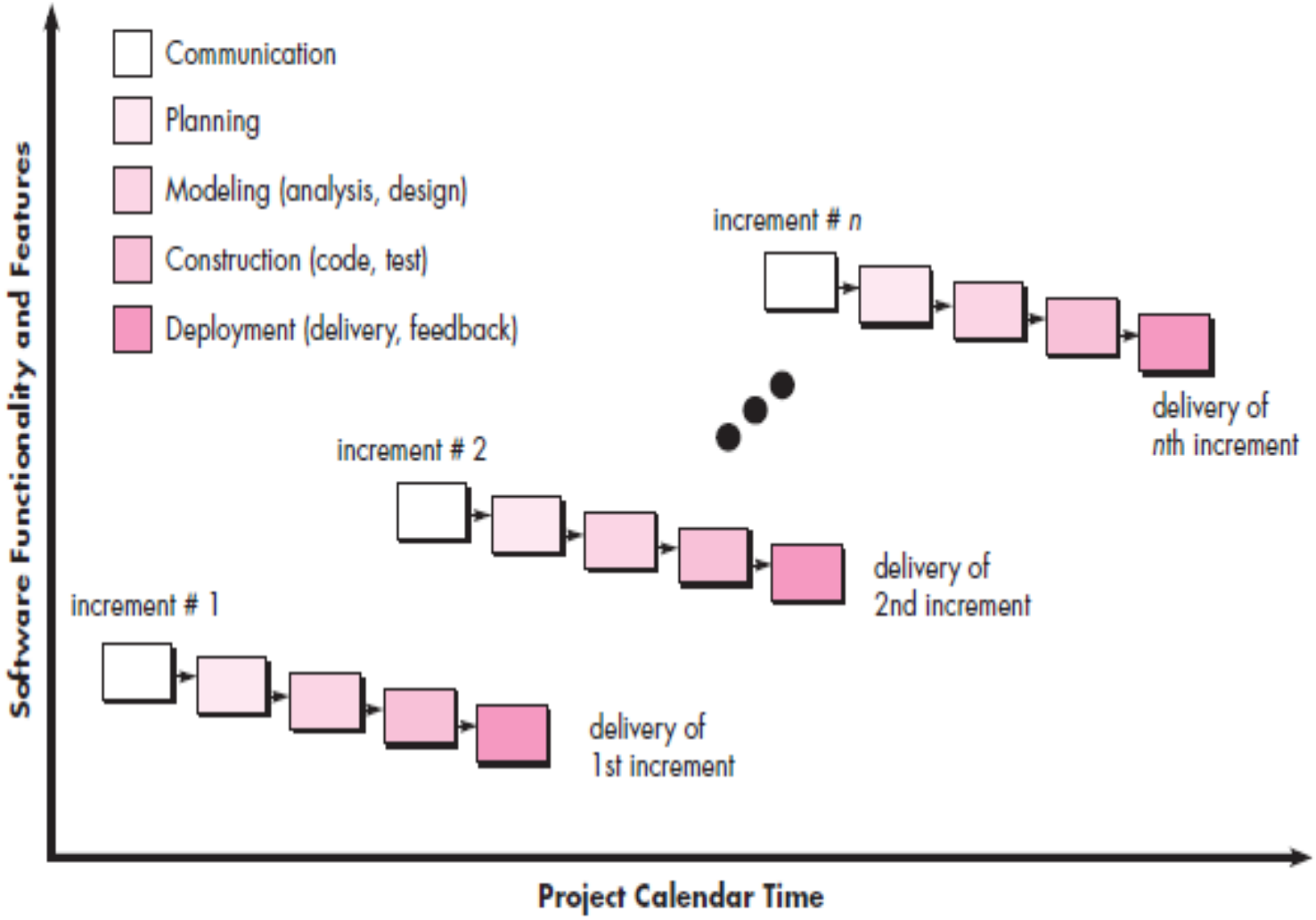
- The V - model is SDLC model where execution of processes happens in a sequential manner in V-shape.
- It is also known as ***Verification and Validation model***.
- V - Model is an extension of the waterfall model and is based on association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle there is a directly associated testing phase.
- This is a highly disciplined model and next phase starts only after completion of the previous phase.

The V-Model(Cont..)

- A variation of waterfall model depicts the relationship of quality assurance actions to the actions associated with communication, modeling and early code construction activates.

- Team first moves down the left side of the V to refine the problem requirements. Once code is generated, the team moves up the right side of the V, performing a series of tests that validate each of the models created as the team moved down the left side.

The Incremental Model



The Incremental Model(Cont..)

- The incremental model combines element of waterfall model applied in an iterative fashion.
- The incremental model applies linear sequence like as calendar time progresses. Each linear sequence.
- **For Example:-**
  - Word processing software development using the incremental paradigm.
  - Basic file management, editing, document production function in the first increment.
  - More sophisticated editing, and document production capabilities in the second increment.
  - Spelling and grammar checking in the third increment.
  - Advance page layout capability in the fourth.

The Spiral Model

Planning
estimation
scheduling
risk analysis

Communication

Modeling
analysis
design

Start

Deployment
delivery
feedback

Construction
code
test

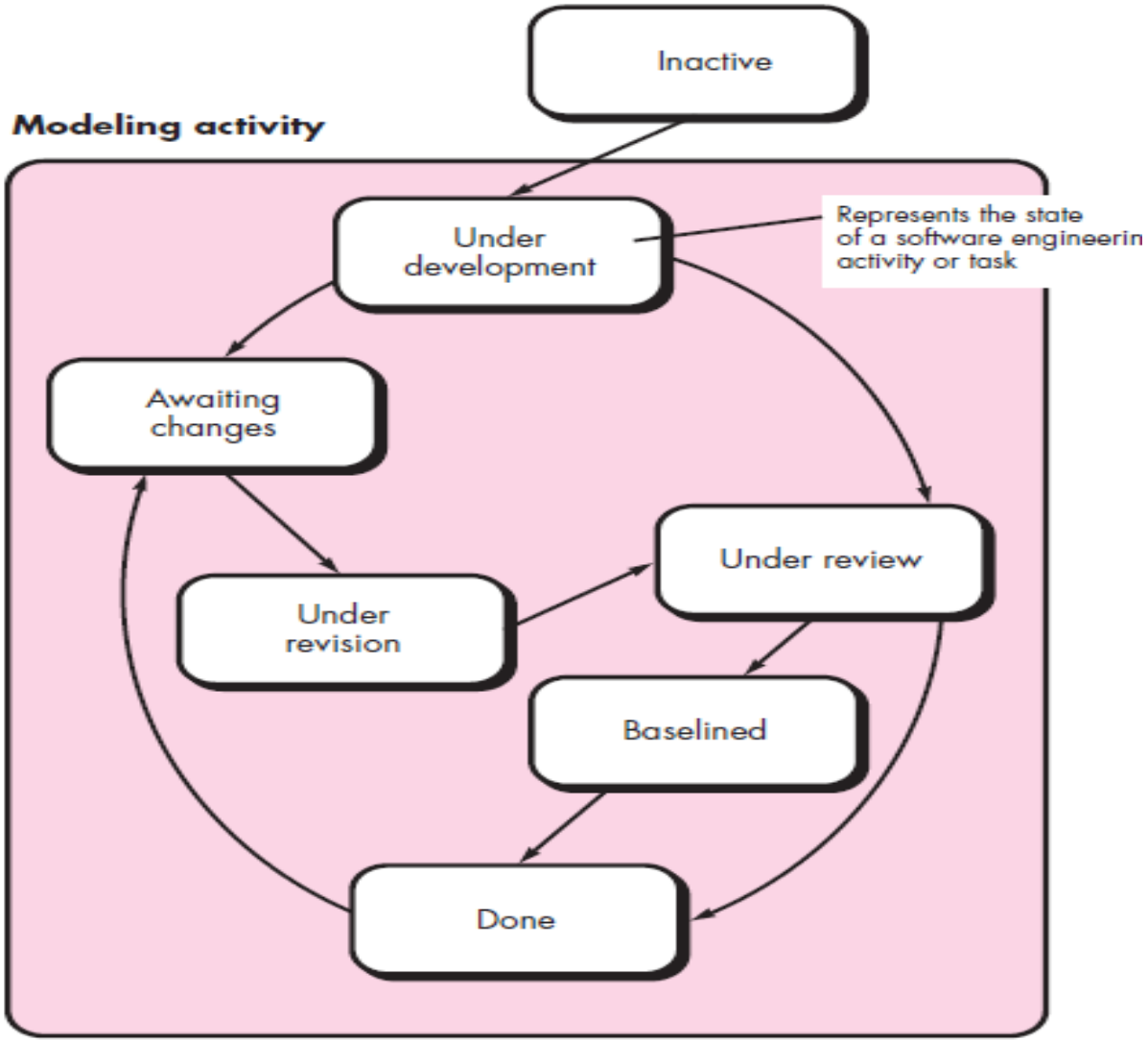
The Spiral Model(Cont..)

- The spiral model is a evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspect of the waterfall model.
- It provides the potential for rapid development of increasingly more complete version of the software.
- A spiral model is divided into a set of framework activities defined by software engineering team. Each framework activities represent one segment of the spiral path.
- Activities are implied by a circuit around the spiral clockwise direction, beginning at the center.

The Spiral Model(Cont..)

- Concept development project
  - Which starts at the core of the spiral and continues for multiple iteration.
- New product development
  - The new product will develop through a number of iteration around the spiral.
- The spiral model is a realistic approach to development of large scale system and software. Because software development the process progresses.

Concurrent Models



Concurrent Model(Cont..)

- **The concurrent development model -** called concurrent engineering
  - It provides an accurate state of the current state of a project.
  - Focus on concurrent engineering activities in a software engineering process such as prototyping, analysis modeling, requirements specification and design.
  - Represented schematically as a series of major technical activities, tasks and their associated states.
  - Defined as a series of events that trigger transitions from state to state for each of the software engineering activities.

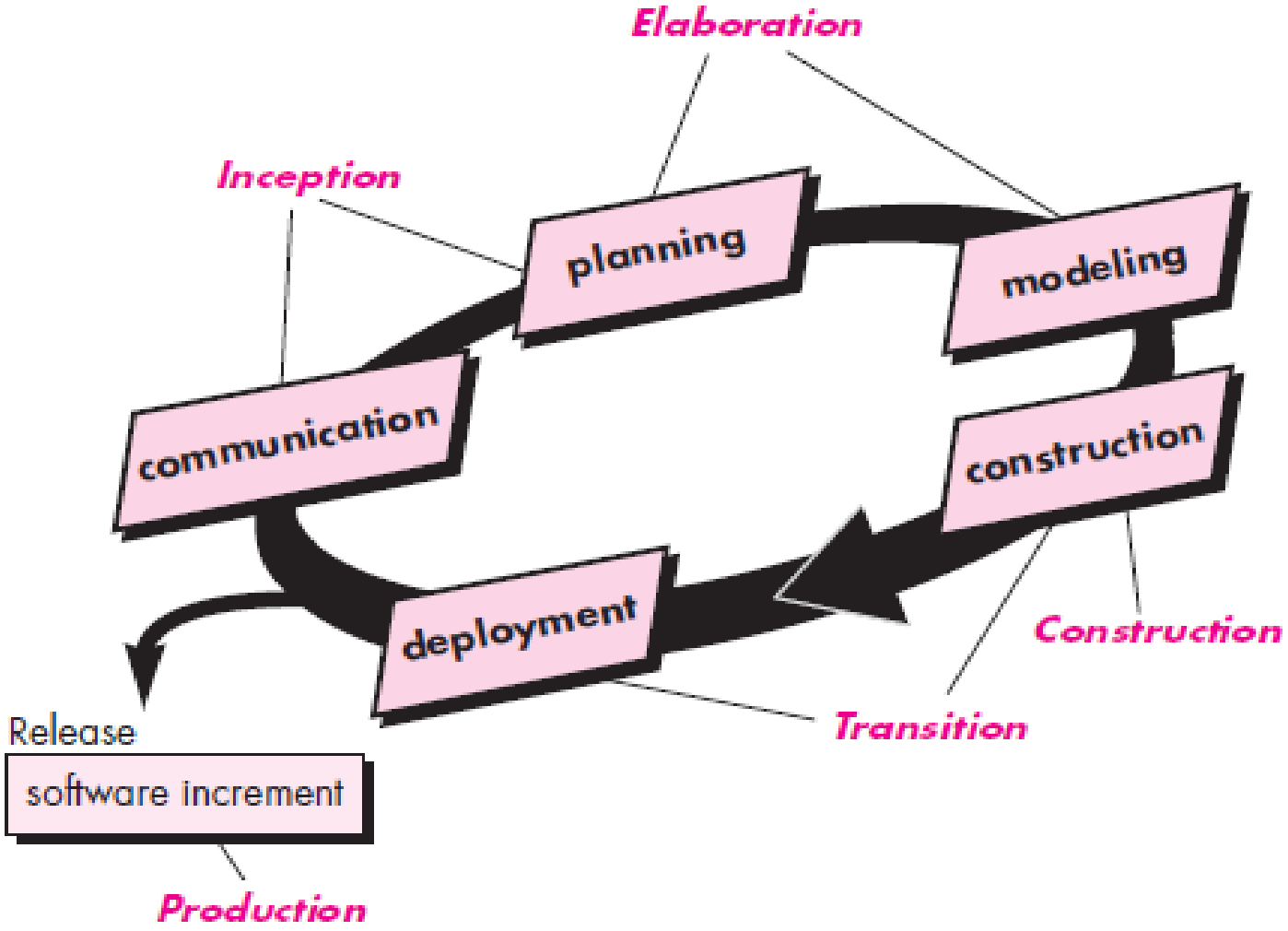
Concurrent Model(Cont..)

- Often used as the paradigm for the development of client/server applications. A client/server system is composed of a set of functional components.
- When applied to client/server, the concurrent process model defines activities in two dimensions :
  - (i) System dimension -System level issues are addressed using three activities: design, assembly, and use.
  - (ii) The component dimension is addressed with two activities: design and realization.

Unified Process Model

- Unified process (UP) is an **architecture-centric**, **use-case driven**, **iterative and incremental** development process that leverages unified modeling language and is compliant with the system process engineering meta-model.
- Unified process can be applied to different software systems with different levels of technical and managerial complexity across various domains and organizational cultures.
- UP is also referred to as the unified software development process.

The Unified Process (UP)



The Unified Process (UP)

Inception phase

Vision document
Initial use-case model
Initial project glossary
Initial business case
Initial risk assessment.
Project plan,
  phases and iterations.
Business model,
  if necessary.
One or more prototypes

Elaboration phase

Use-case model
Supplementary requirements
  including non-functional
Analysis model
Software architecture
  Description.
Executable architectural
  prototype.
Preliminary design model
Revised risk list
Project plan including
  iteration plan
  adapted workflows
  milestones
  technical work products
Preliminary user manual

Construction phase

Design model
Software components
Integrated software
  increment
Test plan and procedure
Test cases
Support documentation
  user manuals
  installation manuals
  description of current
    increment

Transition phase

Delivered software increment
Beta test reports
General user feedback

Selection of a Life Cycle Model

- Selection of a model is based on:
  - Requirements
  - Development team
  - Users
  - Project type and associated risk

Selection of a Life Cycle Model

- Selection of a model is based on:
  - Requirements
  - Development team
  - Users
  - Project type and associated risk

Based On Characteristics Of Requirements

Requirements	Waterfall	Prototype	Iterative enhancement	Evolutionary development	Spiral	RAD
Are requirements easily understandable and defined?	Yes	No	No	No	No	Yes
Do we change requirements quite often?	No	Yes	No	No	Yes	No
Can we define requirements early in the cycle?	Yes	No	Yes	Yes	No	Yes
Requirements are indicating a complex system to be built	No	Yes	Yes	Yes	Yes	No

Based On Status Of Development Team

Development team	Waterfall	Prototype	Iterative enhancement	Evolutionary development	Spiral	RAD
Less experience on similar projects?	No	Yes	No	No	Yes	No
Less domain knowledge (new to the technology)	Yes	No	Yes	Yes	Yes	No
Less experience on tools to be used	Yes	No	No	No	Yes	No
Availability of training if required	No	No	Yes	Yes	No	Yes

Based On User's Participation

Involvement of Users	Waterfall	Prototype	Iterative enhancement	Evolutionary development	Spiral	RAD
User involvement in all phases	No	Yes	No	No	No	Yes
Limited user participation	Yes	No	Yes	Yes	Yes	No
User have no previous experience of participation in similar projects	No	Yes	Yes	Yes	Yes	No
Users are experts of problem domain	No	Yes	Yes	Yes	No	Yes

References:

1. **Software Engineering A practitioner's Approach**

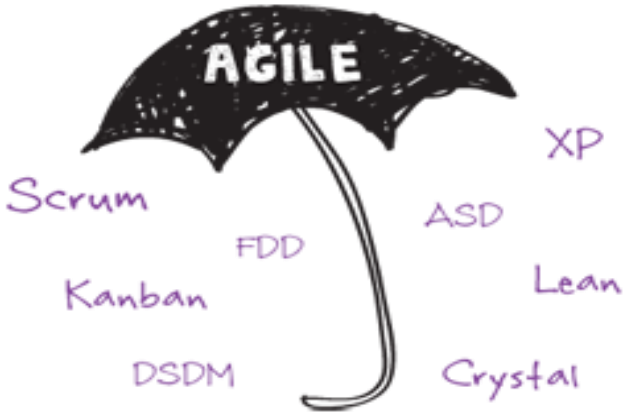
   by Roger S. Pressman, 7th edition, McGraw Hill, 2010.

2. **Software Engineering by Ian Sommerville,**

   9th edition, Addison-Wesley, 2011

3. **Software Engineering Practices**

   Mohamed Sami, https://melsatar.blog/2012/03/15/software-development-life-cycle-models-and-methodologies/



Week-3 : Lesson-1 Agile Model

Abdus Sattar

Assistant Professor

Department of Computer Science and Engineering

Daffodil International University

Email: abdus.cse@diu.edu.bd

Daffodil International University

Topics Covered

- What is agility?
- Use Agile Model
- 12 Agile Principles
- Agile Models
- Extreme Programming(XP)
- Kanban Model
- Adaptive Software Development(ASD)
- Dynamic Systems Development (DSD) Method
- Scrum Agile Process

Learning Goals

- Understand the rationale for agile software development methods, the agile manifesto, and the differences between agile and plan driven development.
- Know the key practices in extreme programming and how these relate to the general principles of agile methods.
- Understand the Scrum approach to agile project management.

What is “Agility”?

- Ability to move quickly and easily.
- Effective (rapid and adaptive) response to change
- Effective communication among all stakeholders
- Drawing the customer onto the team
- Rapid, incremental delivery of software

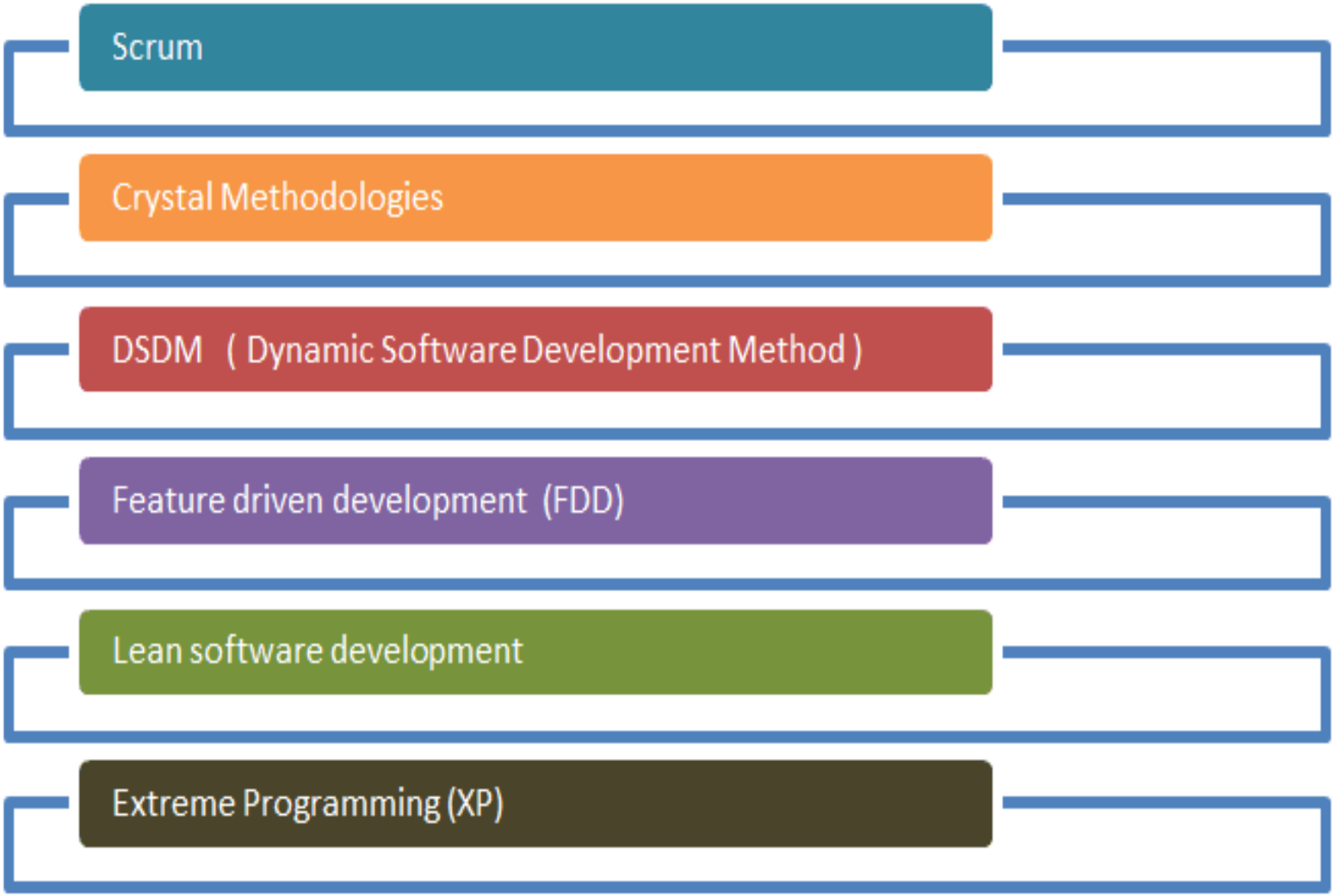
12 Agility Principles

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face–to–face conversation.

12 Agility Principles(Cont..)

7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity – the art of maximizing the amount of work not done – is essential.
11. The best architectures, requirements, and designs emerge from self–organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

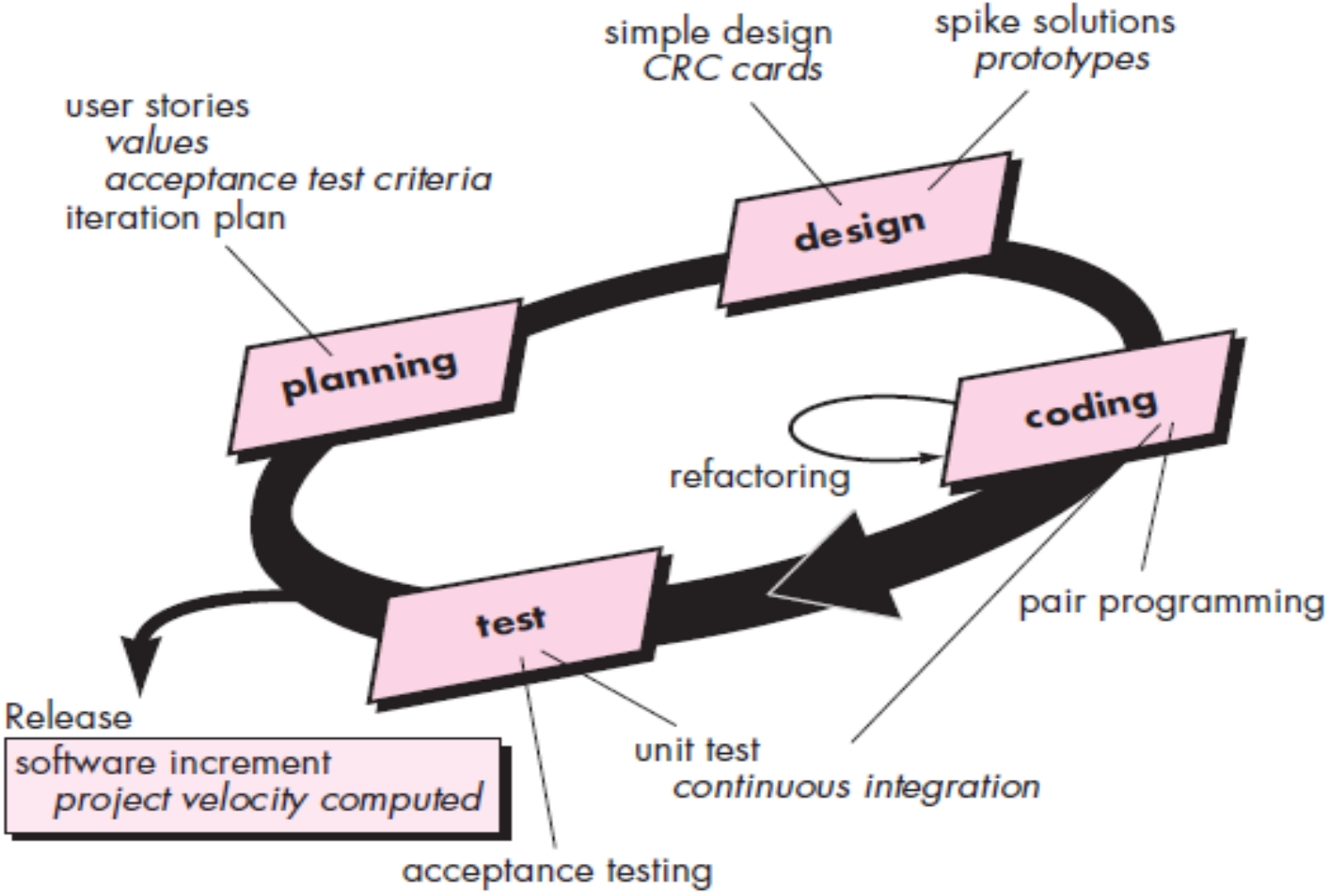
Agility Methodology



Extreme Programming(XP)

- XP uses an object oriented approach.
- XP encompasses a set of rules and practice that occur within the context of four framework activities.
  - XP Planning
  - XP Design
  - XP Coding
  - XP Testing

Extreme Programming(XP)



Extreme Programming(XP)

- **There are four basic activities that XP proposes for software development process :**
  1. **XP Planning :**
     - The planning activity begins with the creating of a set of stories that describe required features and functionally for software to be built.
     - Each stories is written by the customer and placed on an index card.
     - The customer assign a value to the story on the overall business value of the feature or function.
     - ¨Members of the XP team the access each story and assign a cost measured in development week to it.
  2. **XP Design:**
     - XP design follows the KIS-Keep it simple principle.
     - A simple design is always preferred over a more complex representation.
     - The design provides implementation giddiness for a story as it is written nothing less, nothing more.
     - The XP team conducts the design exercise using a process and the CRC cards are the only design work product produced as the part of XP process.
     - XP recommends the immediate creating of an operational prototype of that portion of the design called spike solution.

Extreme Programming(XP)

3. XP Coding :

- XP recommends that after stories are developed and preliminary design is done, the team should not move to code, but rather develop a series of unit test.
- Once the code is complete, it can be unit tested immediately, thereby providing instaneous feedback to the developers.
- During the coding activity is pair programming.
- XP recommends that two people work together at one computer work station to create code for a story. This provides a mechanism for real time problem solving and real time quality assurance.

4. XP Testing:

- The creation of unit test before coding commence is a key element of the XP approach.
- The unit test that are created should be implemented using a framework the enable them to be automated.
- Integration and validation testing of the system can occur a daily basis.
- XP acceptance test, also called customer test are specified by the customer and focus on overall system.

Kanban Model

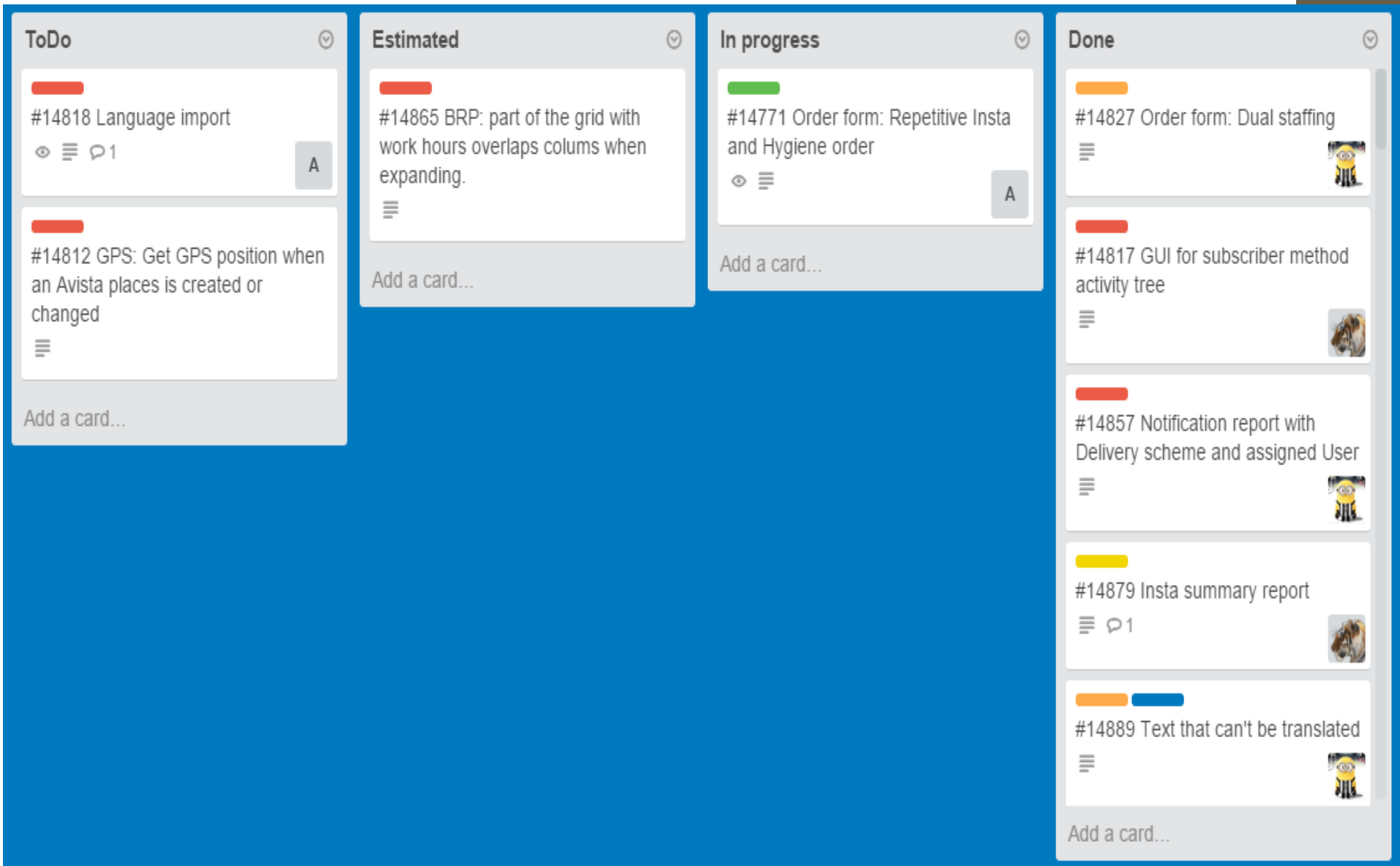
- Kanban is a visual system for managing work as it moves through a process.
- Kanban visualizes both the process (the workflow) and the actual work passing through that process.
- The goal of Kanban is to identify potential bottlenecks in your process and fix them so work can flow through it cost-effectively at an optimal speed or throughput.

The Three Principles of Kanban Development

Three core principles allow you to use Kanban in your project:

- **Visualize what you do today (workflow):** seeing all the items in context of each other can be very informative
- **Limit the amount of work in progress (WIP):** this helps balance the flow-based approach so teams don't start and commit to too much work at once
- **Enhance flow:** when something is finished, the next highest thing from the backlog is pulled into play

Kanban Board Example



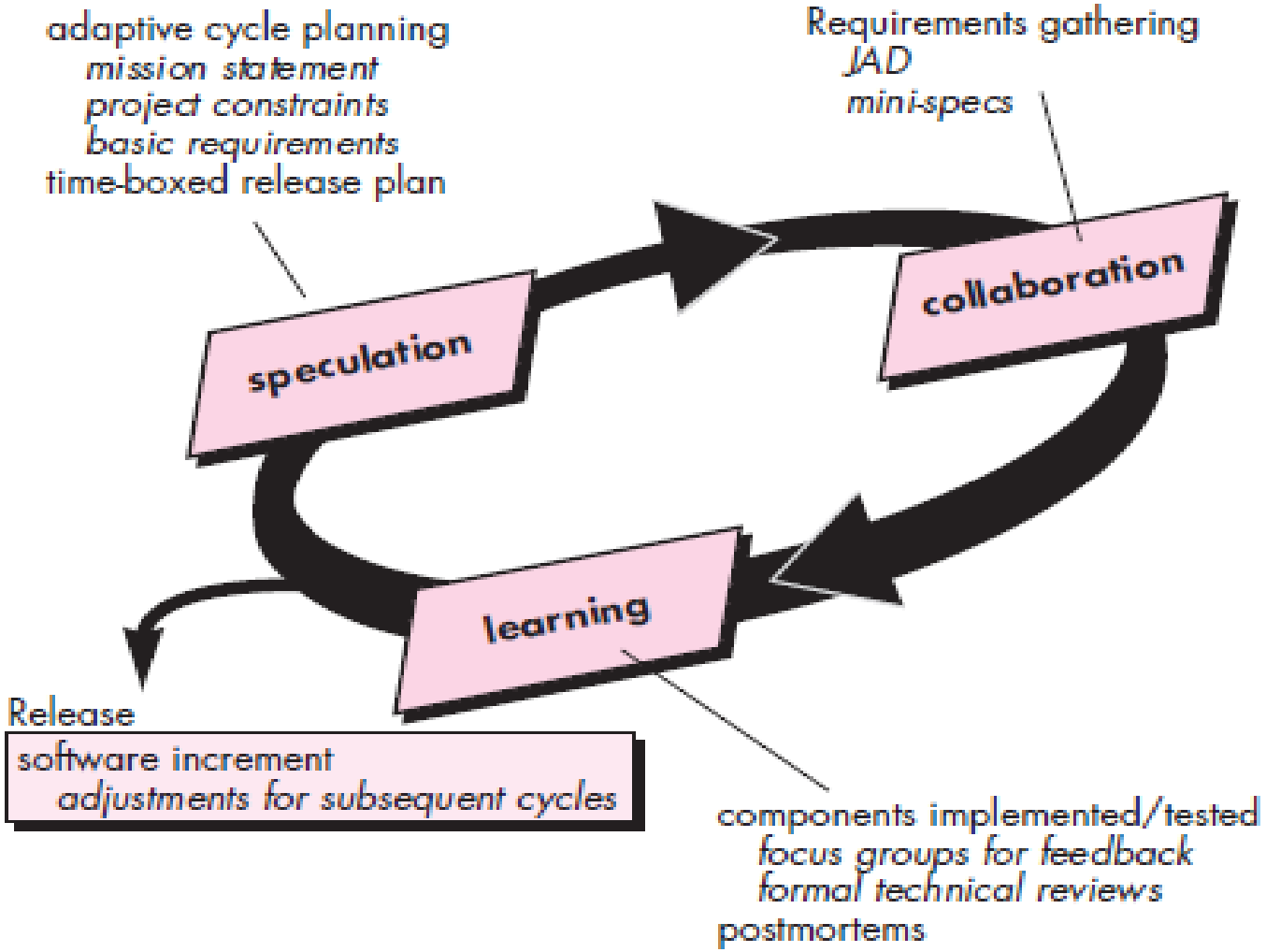
Kanban Board Example(Cont…)

- **To Do section:** contains tasks that were received from the customer and required to be analyzed. Each task is marked with color according to its priority.
- **Estimated Section:** When tasks from the first section have been analyzed and estimated by the Team, they are moved to the Estimated section.
- **In Progress:** When the developer takes a task to be developed, he moves it from Estimated to In Progress section and marks it with his tag to show who handles each task.
- **Done:** When a task is done, it's moved to the Done section.

Adaptive Software Development(ASD)

- Originally proposed by Jim Highsmith
- ASD technique proposed for building complex software and system.
- ASD focus on human collaboration and team-self-organization.
- ASD incorporates three phases:-
  - Speculation
  - Collaboration
  - Learning

Adaptive Software Development



ASD Three Phases

1. **Speculation:**
   - During speculation , the project is initiated and adapted cycle planning is conducted.
   - Adapting cycle planning uses project initiation – information the customers mission statement, project constraints and basic requirements to define the set of release cycle.
2. **Collaboration**
   - The collaboration approach is requiring theme in all agile methods, but collaboration is not easy.
   - It is not simply communicate, although communicate is a part of it.
   - It is nota rejection individualism, because individual creativity plays on important role in collaboration thinking.
   - People working together must trust one another to:-
     - Criticize without animosity
     - Assist without resentment.
     - Work as hard of harder as they do.
     - Have the skill set to contribute to the work at hard
     - Communicate problem

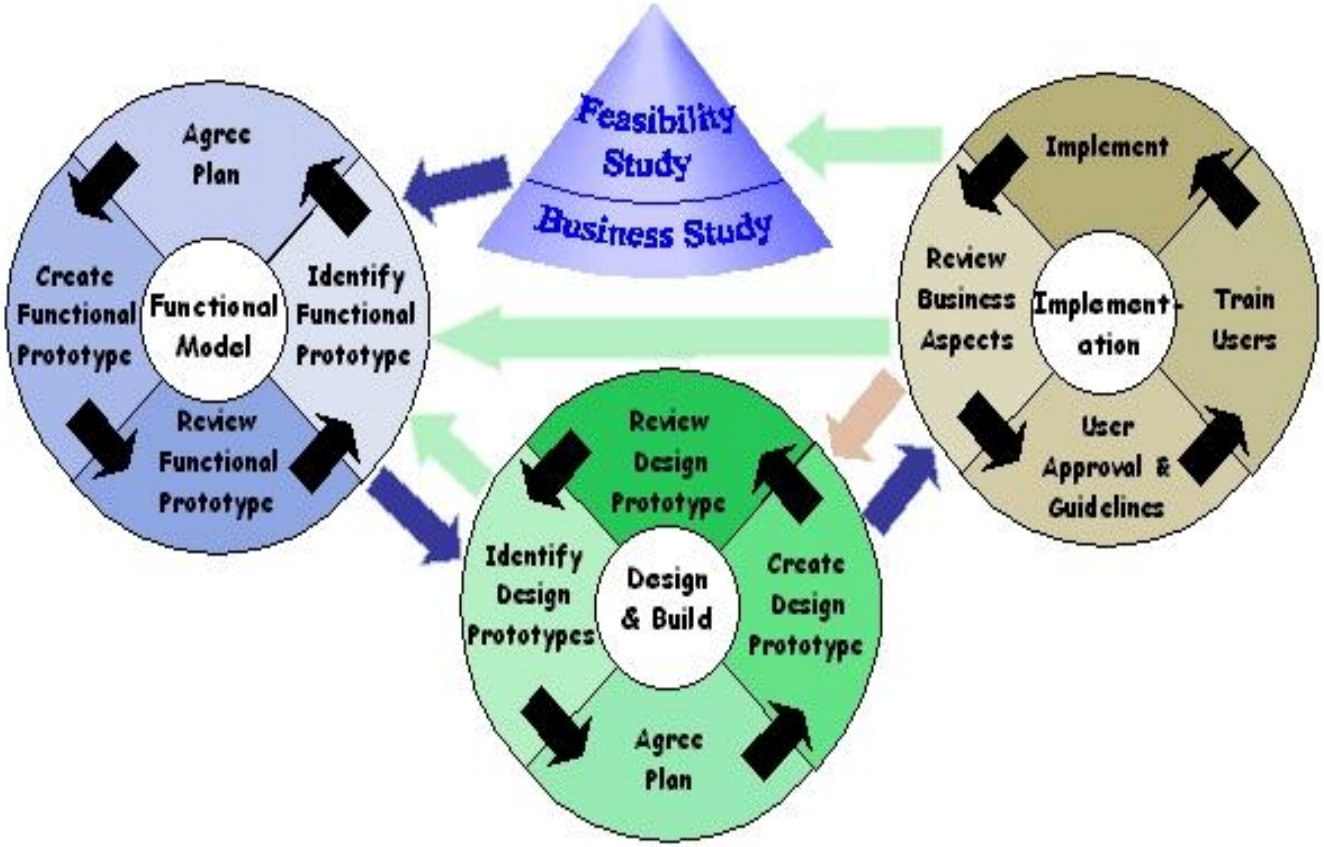
Adaptive Software Development

3. **Learning**
   - ASD teams learns three ways:-
   - Focus Group:
     - The customer or end user provide feedback on software increments that are being delivered.
   - Formal technical review:
     - ASD team members review the software components that are develop, improving quality and learning as they proceed.
   - Postmaster
     - The ASD team becomes introspective , addressing its own performance and process.

Dynamic Systems Development (DSD) Method

- The Dynamic System Development method (DSDM) is an agile software development approach that provides a framework for building and maintain system which meet tight time constraints through the use of incremental prototyping in a controlled project environment.
- The DSDM life cycle defines, three iterative cycle precede by two additional life cycle.
  - Feasibility study
  - Business study
  - Functional model iteration
  - Design and iteration
  - Implementation

Dynamic Systems Development (DSD) Method



DSDM Iterative life cycle

- **Feasibility study**
  - Established the basic business requirements and constraints associated with the applicants to be built.
- **Business study**
  - Establishes the functional information requirements that will allow the applicants to provide business value.
- **Functional model iteration**
  - Produce a set of incremental prototype that demonstrate functionality for the customer.
- **Design and iteration**
  - Revisits prototype built during the functional model iteration to ensure that each has been engineered in a manner.
- **Implementation**
  - Places the latest software increment into the operational environment.
    - It should be noted that-
      - The increment may not be 100 percent complete
      - Changes may be requested as the increment is put into place.

Scrum Agile Process

- Scrum is an agile software development method that was conceived by Jeff Sutherland and his development team in the early 1990s.
- In recent years, further development on the Scrum methods has been performed by Schwaber and Beedle
- Scrum principles are consistent with the agile manifesto and are used to guide development activities within a process that incorporates the following framework activities: requirements, analysis, design, evolution, and delivery.

Scrum Agile Process

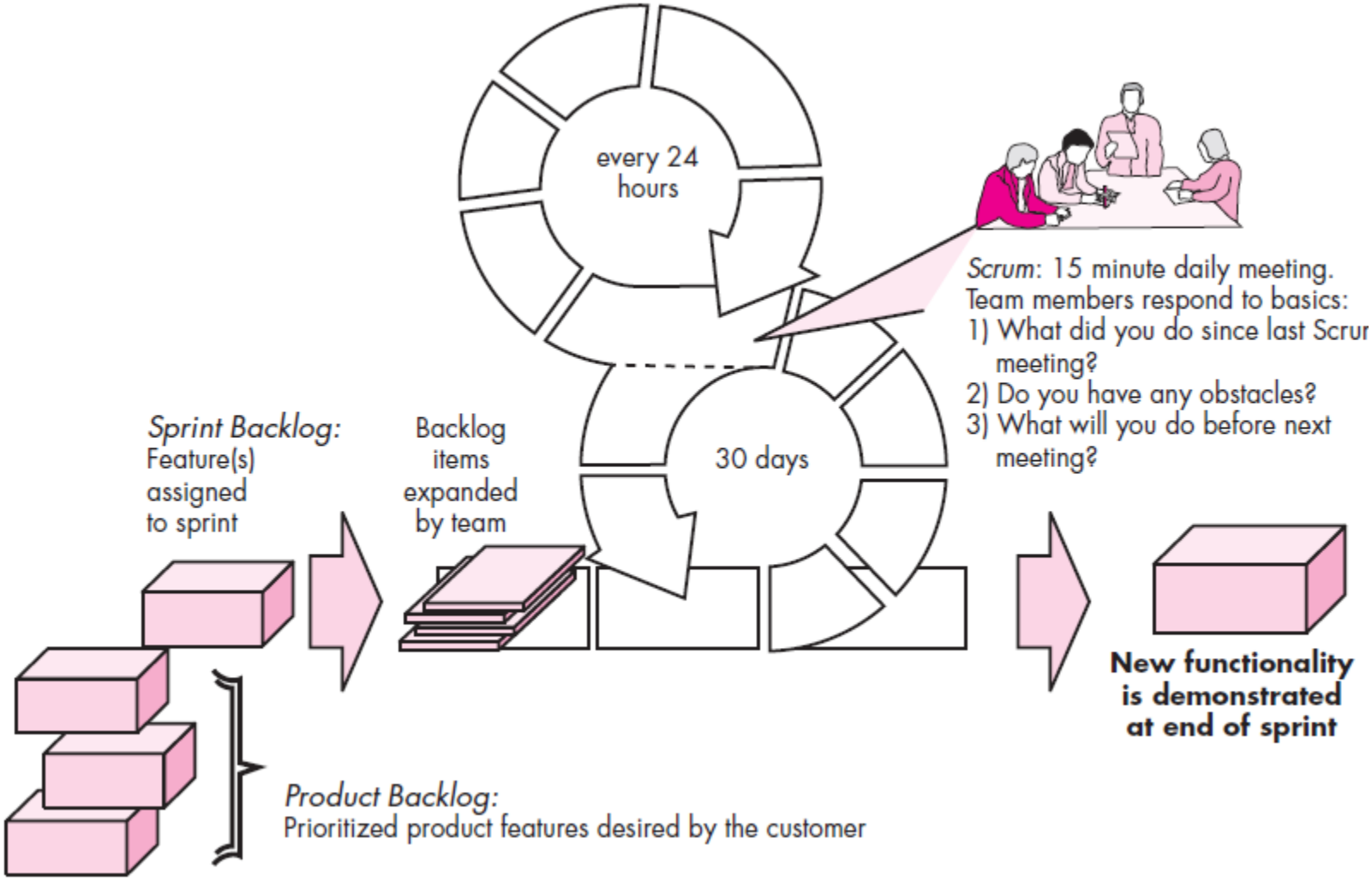


Fig: Scrum Process Flow

Scrum Agile Principles

- **Small working teams are organized to maximize communication, minimize overload and maximize sharing tacit, informal knowledge.**
- **The process must be adaptable to both technical and business changes – to ensure the best possible product is produced.**
- **The process yields frequent software increment that can be inspected, adjusted, tested , documented and built on.**
- **Development work and people who perform it are partitioned into clean low coupling partitions or packets.**
- **Constant testing and documentation is preferred as the product is built.**
- **The scrum process provides the ability to declare a product done whenever required.**

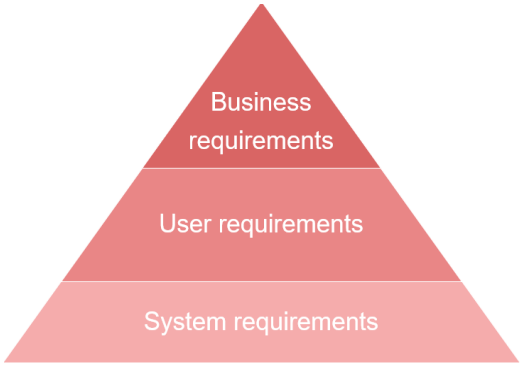
References:

1. **Software Engineering A practitioner's Approach**

   by Roger S. Pressman, 7th edition, McGraw Hill, 2010.

2. **Software Engineering by Ian Sommerville,**

   9th edition, Addison-Wesley, 2011



Week-4-Lesson-1
Understanding Requirements, Use Case and Use case Description

Abdus Sattar
Assistant Professor
Department of Computer Science and Engineering
Daffodil International University
Email: abdus.cse@diu.edu.bd

Daffodil International University

Topics Covered

- Requirements Engineering
- Requirements analysis
- Elements of Requirements Engineering
- Classification of Requirements
- Functional Requirements
- Non-Functional Requirements
- Use Case Diagram
- Use Case Description

Requirements Engineering

- **Requirements Engineering**
  - Requirements are statements of what the system must do, how it must behave, the properties it must exhibit, the qualities it must possess, and the constraints that the system and its development must satisfy.
- **Requirements analysis**
  - specifies software's operational characteristics
  - indicates software's interface with other system elements
  - establishes constraints that software must meet

Requirements Engineering

1. Inception—ask a set of questions that establish …
   - basic understanding of the problem
   - the people who want a solution
   - the nature of the solution that is desired, and
   - the effectiveness of preliminary communication and collaboration between the customer and the developer
2. Elicitation—elicit requirements from all stakeholders
3. Elaboration—create an analysis model that identifies data, function and behavioral requirements
4. Negotiation—agree on a deliverable system that is realistic for developers and customers

Requirements Engineering

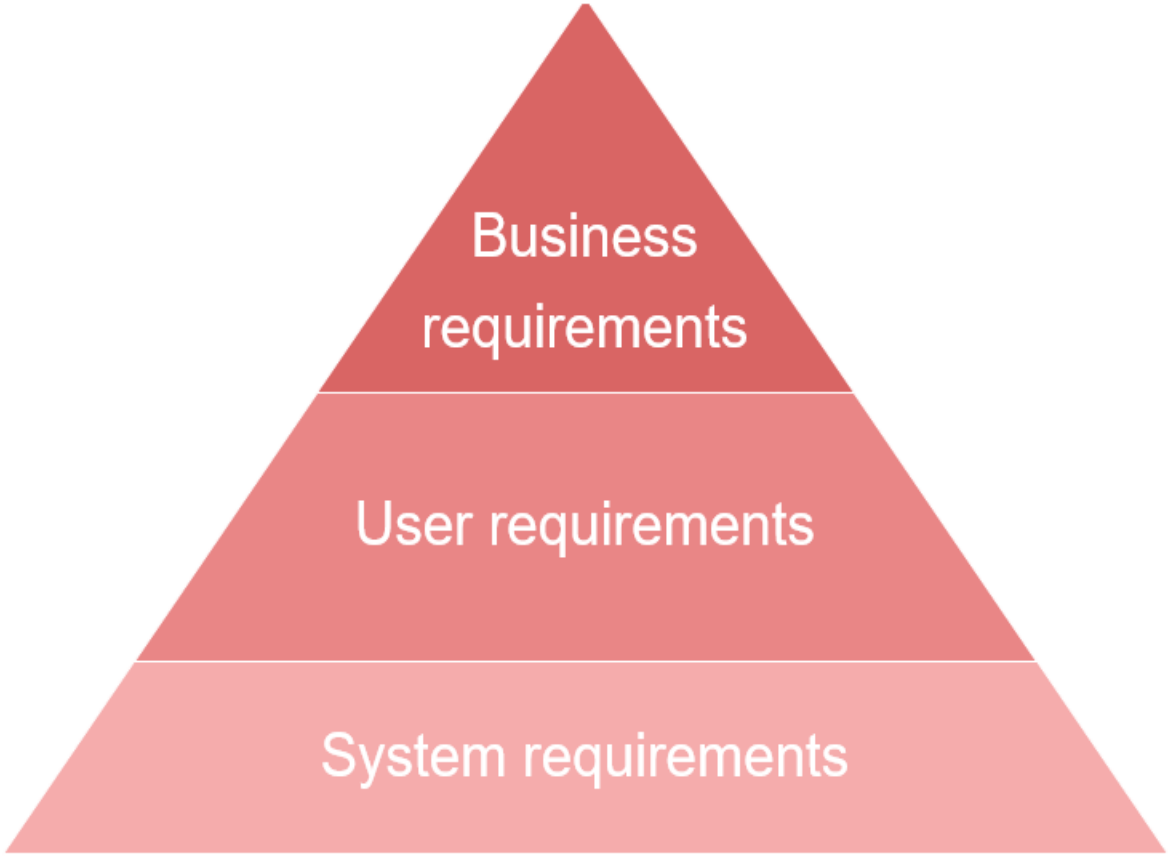
5. Specification—can be any one (or more) of the following:
   - A written document
   - A set of models
   - A formal mathematical
   - A collection of user scenarios (use-cases)
   - A prototype
6. Validation—a review mechanism that looks for
   - errors in content or interpretation
   - areas where clarification may be required
   - missing information
   - inconsistencies (a major problem when large products or systems are engineered)
   - conflicting or unrealistic (unachievable) requirements.
7. Requirements management

Classification of Requirements

Requirements classification

High level

Detailed



Classification of requirements

- **Business requirements.** These include high-level statements of goals, objectives, and needs.
- **Stakeholder requirements.** The needs of discrete stakeholder groups are also specified to define what they expect from a particular solution.
- **Solution requirements.** Solution requirements describe the characteristics that a product must have to meet the needs of the stakeholders and the business itself.
  - **Nonfunctional** requirements describe the general characteristics of a system. They are also known as *quality attributes*.
  - **Functional** requirements describe how a product must behave, what its features and functions.
- **Transition requirements.** An additional group of requirements defines what is needed from an organization to successfully move from its current state to its desired state with the new product.

Functional Requirements

- Functional requirements describe system behavior under specific conditions and include the product features and functions which web & app developers must add to the solution. Such requirements should be precise both for the development team and stakeholders.
- The list of examples of functional requirements includes:
  - Business Rules
  - Transaction corrections, adjustments, and cancellations
  - Administrative functions
  - Authentication
  - Authorization levels
  - Audit Tracking
  - External Interfaces
  - Certification Requirements
  - Reporting Requirements
  - Historical Data

Example of Functional Requirements

Here, are some examples of non-functional requirement:

1. The software automatically validates customers against the ABC Contact Management System
2. The Sales system should allow users to record customers sales
3. The background color for all windows in the application will be blue and have a hexadecimal RGB color value of 0x0000FF.
4. Only Managerial level employees have the right to view revenue data.
5. The software system should be integrated with banking API
6. The software system should pass Section 508 accessibility requirement.

Non-Functional Requirements

- A non-functional requirement defines the quality attribute of a software system. They represent a set of standards used to judge the specific operation of a system. Example, how fast does the website load?
- Some typical non-functional requirements are:
  - Performance – for example Response Time, Throughput, Utilization, Static Volumetric
  - Capacity
  - Availability
  - Reliability
  - Recoverability
  - Maintainability
  - Serviceability
  - Security
  - Regulatory
  - Manageability
  - Environmental
  - Data Integrity
  - Usability

Example of Non-Functional Requirements

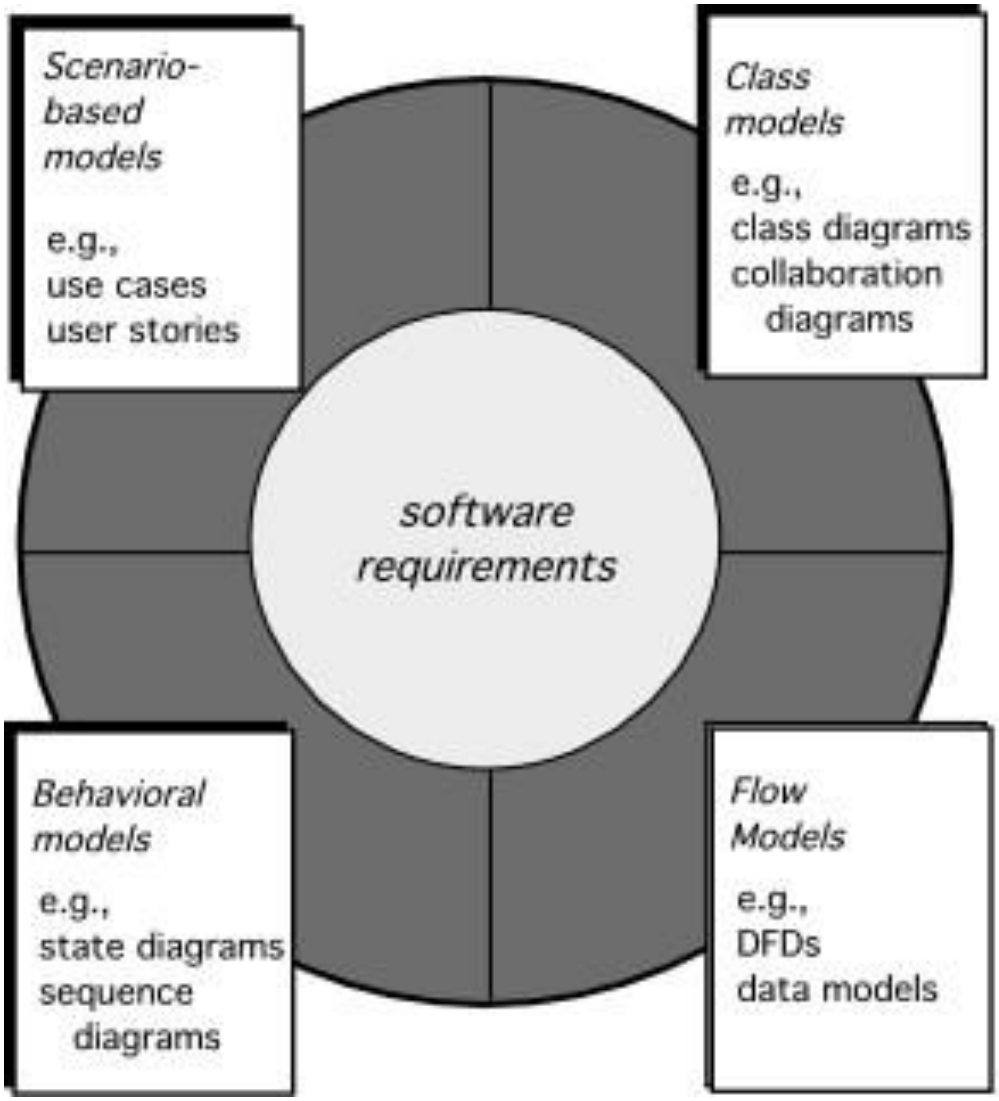
Here, are some examples of non-functional requirement:

1. Users must change the initially assigned login password immediately after the first successful login. Moreover, the initial should never be reused.
2. Employees never allowed to update their salary information. Such attempt should be reported to the security administrator.
3. Every unsuccessful attempt by a user to access an item of data shall be recorded on an audit trail.
4. A website should be capable enough to handle 20 million users with affecting its performance
5. The software should be portable. So moving from one OS to other OS does not create any problem.
6. Privacy of information, the export of restricted technologies, intellectual property rights, etc. should be audited.

Elements of the analysis model

- Elements of the analysis model
  - Scenario-based elements
    - **Functional**—processing narratives for software functions
    - **Use-case**—descriptions of the interaction between an "actor" and the system
  - Class-based elements
    - Implied by scenarios
  - Behavioral elements
    - State diagram
  - Flow-oriented elements
    - Data flow diagram

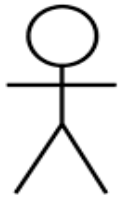
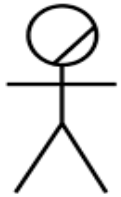
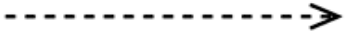
Elements of Requirements Analysis



Use-Cases

- Use cases describe the interaction between the system and external users that leads to achieving particular goals.
- Each use case includes three main elements:
  - **Actors.** These are the users outside the system that interact with the system.
  - **System**. The system is described by functional requirements that define an intended behavior of the product.
  - **Goals**. The purposes of the interaction between the users and the system are outlined as goals.
- There are two formats to represent use cases:
  - Use case specification/description
  - Use case diagram

Use-Cases Elements

Symbol Name	Symbol
Actor	
Business Actor	
Use Case	
Business Use Case	
Association	
Dependency	
Generalization	

Use-Cases Elements

Connection between Actor and Use Case

Boundary of system

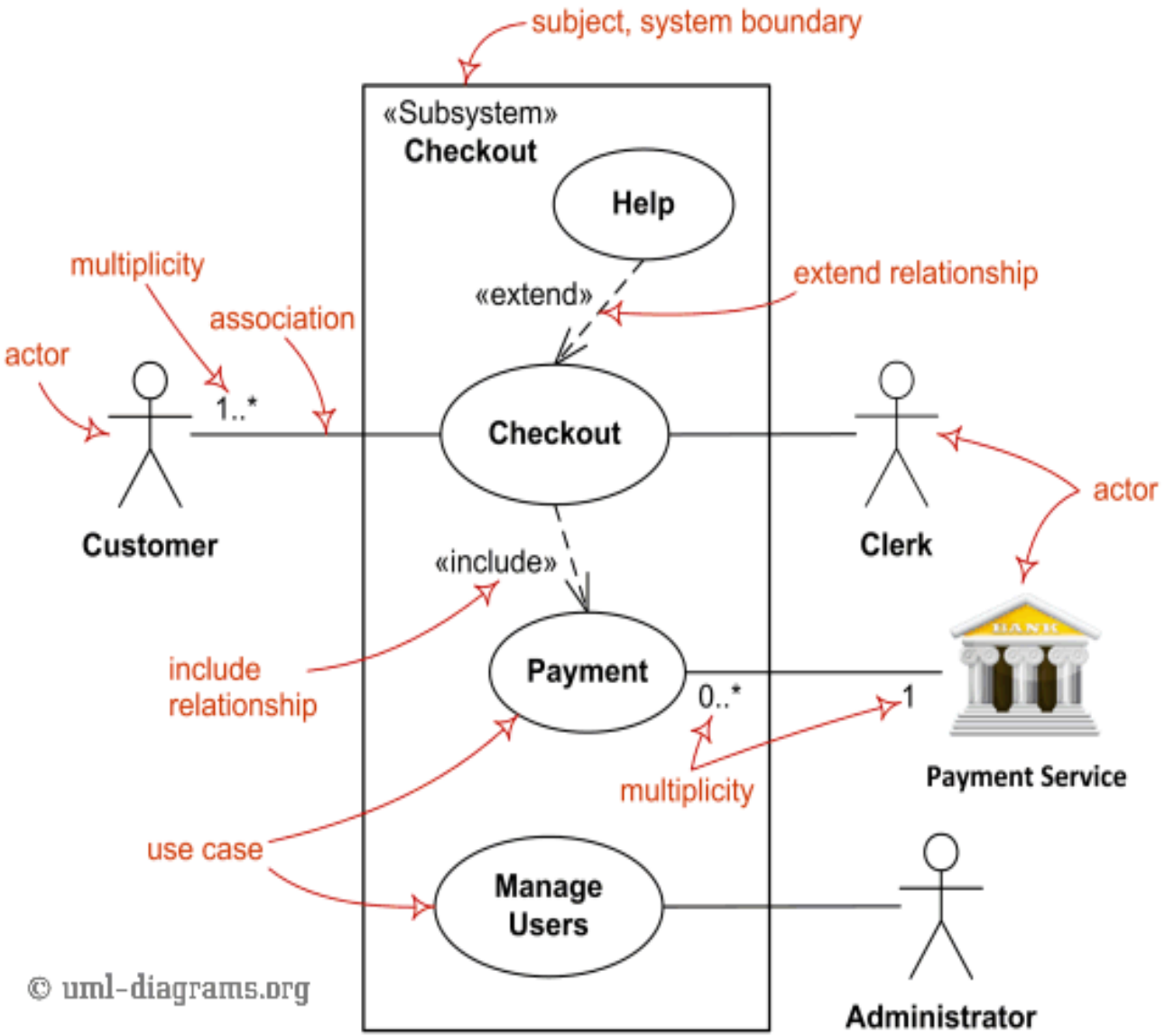
<<include>>

Include relationship between Use Cases (one UC must call another; e.g., Login UC includes User Authentication UC)

<<extend>>

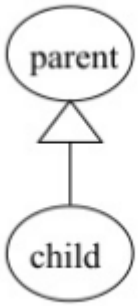
Extend relationship between Use Cases (one UC calls Another under certain condition; think of if-then decision points)

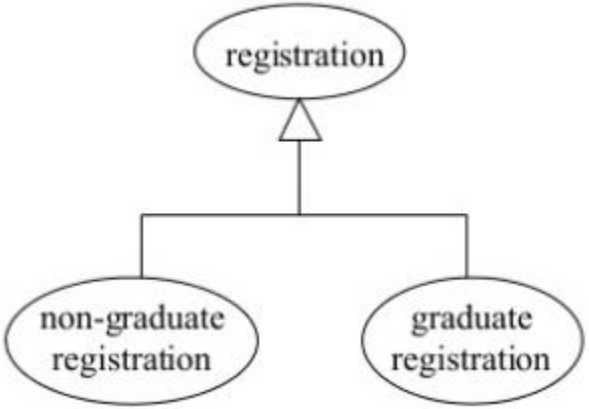
Use Cases Elements

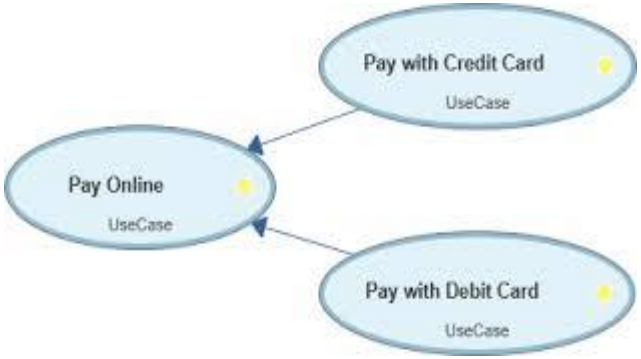


Use-Cases Generalization

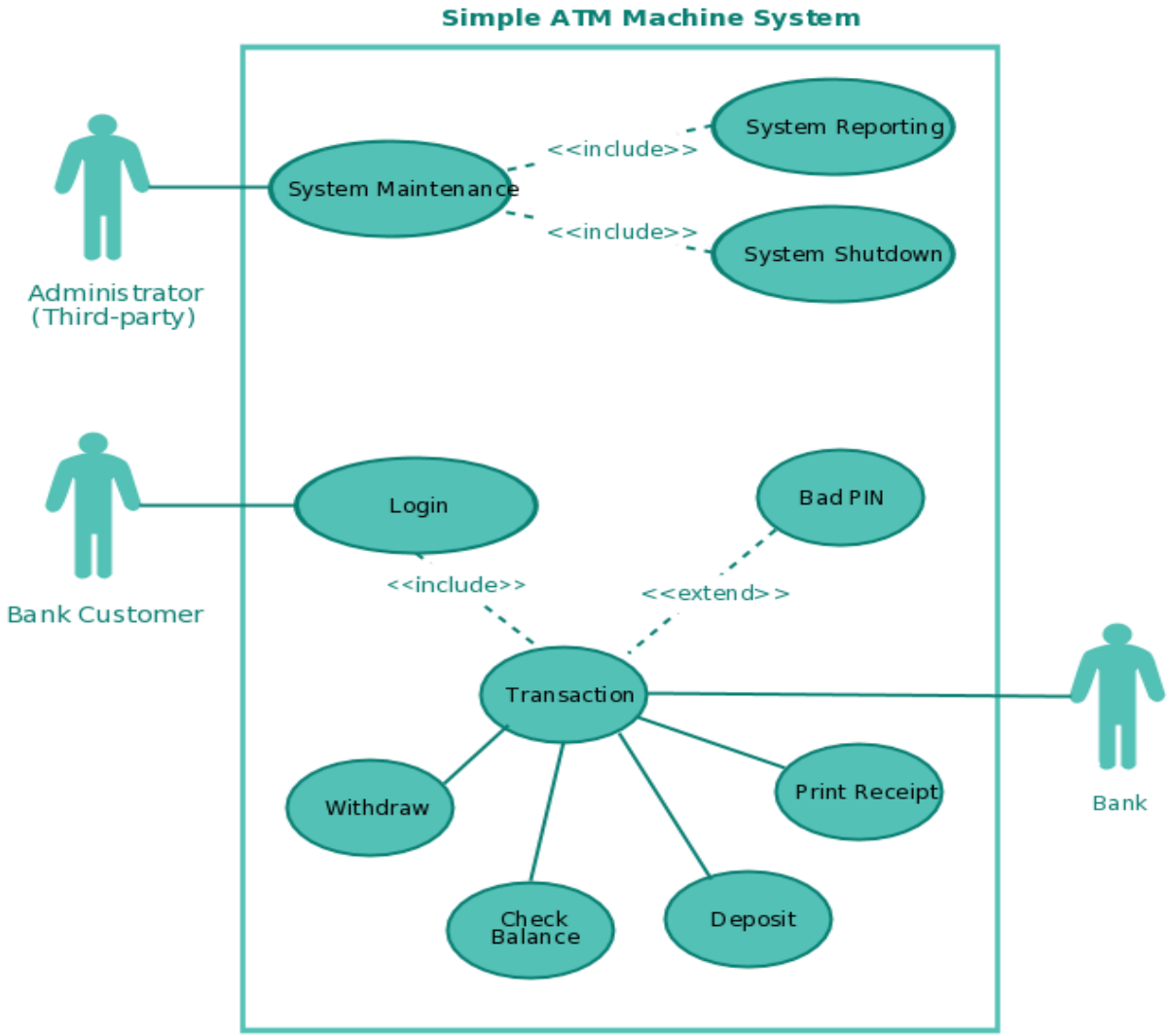
- The child use case inherits the behavior meaning of the parent use case
- The Child may add to or override the behavior of its parent.



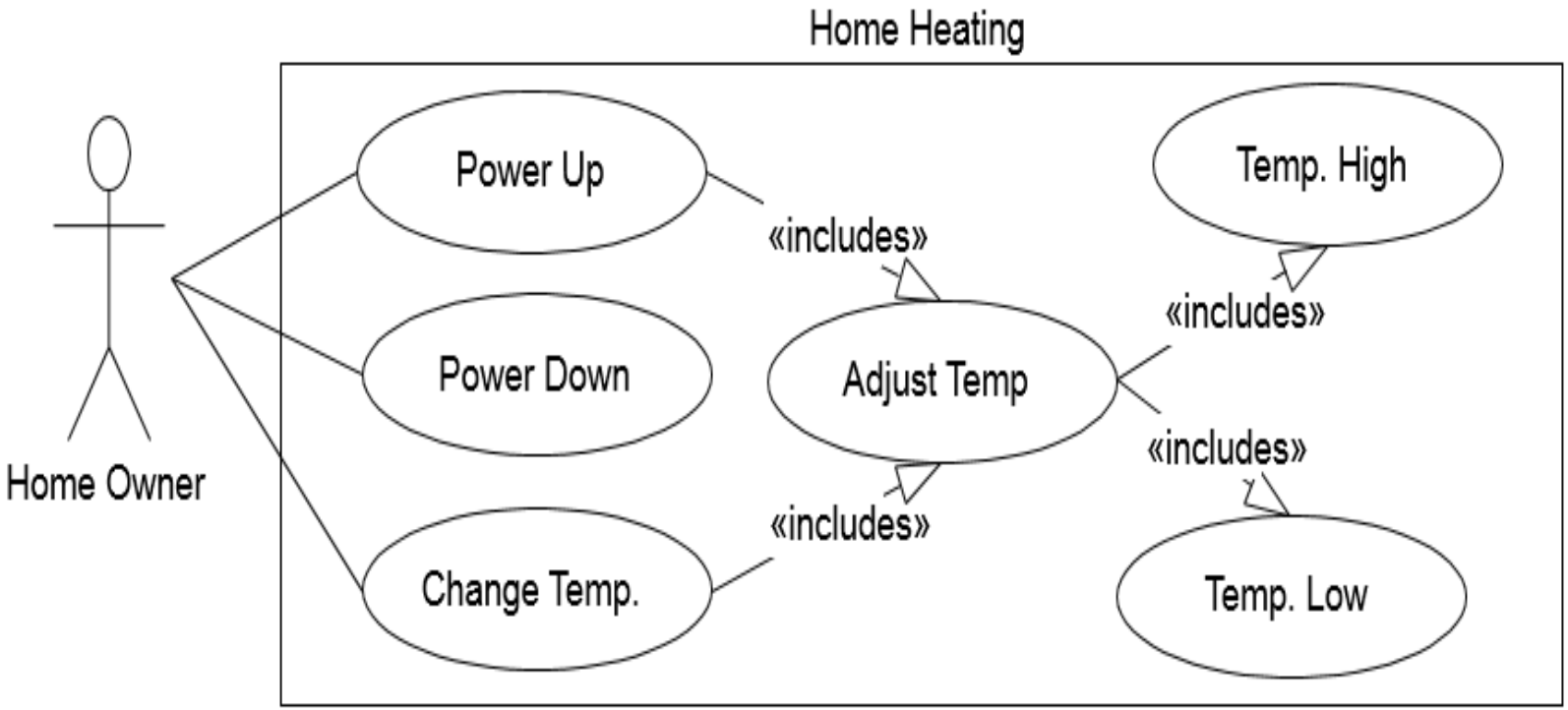




Use-Cases Diagram



Home Heating System



Home Heating System

Use Case Description

Use case:	**Power Up**
Actors:	**Home Owner (initiator)**
Type:	Primary and essential
Description:	The Home Owner turns the power on. **Perform Adjust Temp**. If the temperature in all rooms is above the desired temperature, no actions are taken.
Cross Ref.:	Requirements XX, YY, and ZZ
Use-Cases:	Perform Adjust Temp

Home Heating System

Use case: **Adjust Temp**

Actors: **System (initiator)**

Type: Secondary and essential

Description: Check the temperature in each room. For each room:

Below target: **Perform Temp Low**

Above target: **Perform Temp High**

Cross Ref.: Requirements XX, YY, and ZZ

Use-Cases: Temp Low, Temp High

Home Heating System

Use case: **Temp Low**

Actors: **System (initiator)**

Type: Secondary and essential

Description: Open room valve, start pump if not started.
If water temp falls below threshold,
open fuel value and ignite burner.

Cross Ref.: Requirements XX, YY, and ZZ

Use-Cases: None

Use Case Scenario

Home Assignment distribution and Collection System(HACS)

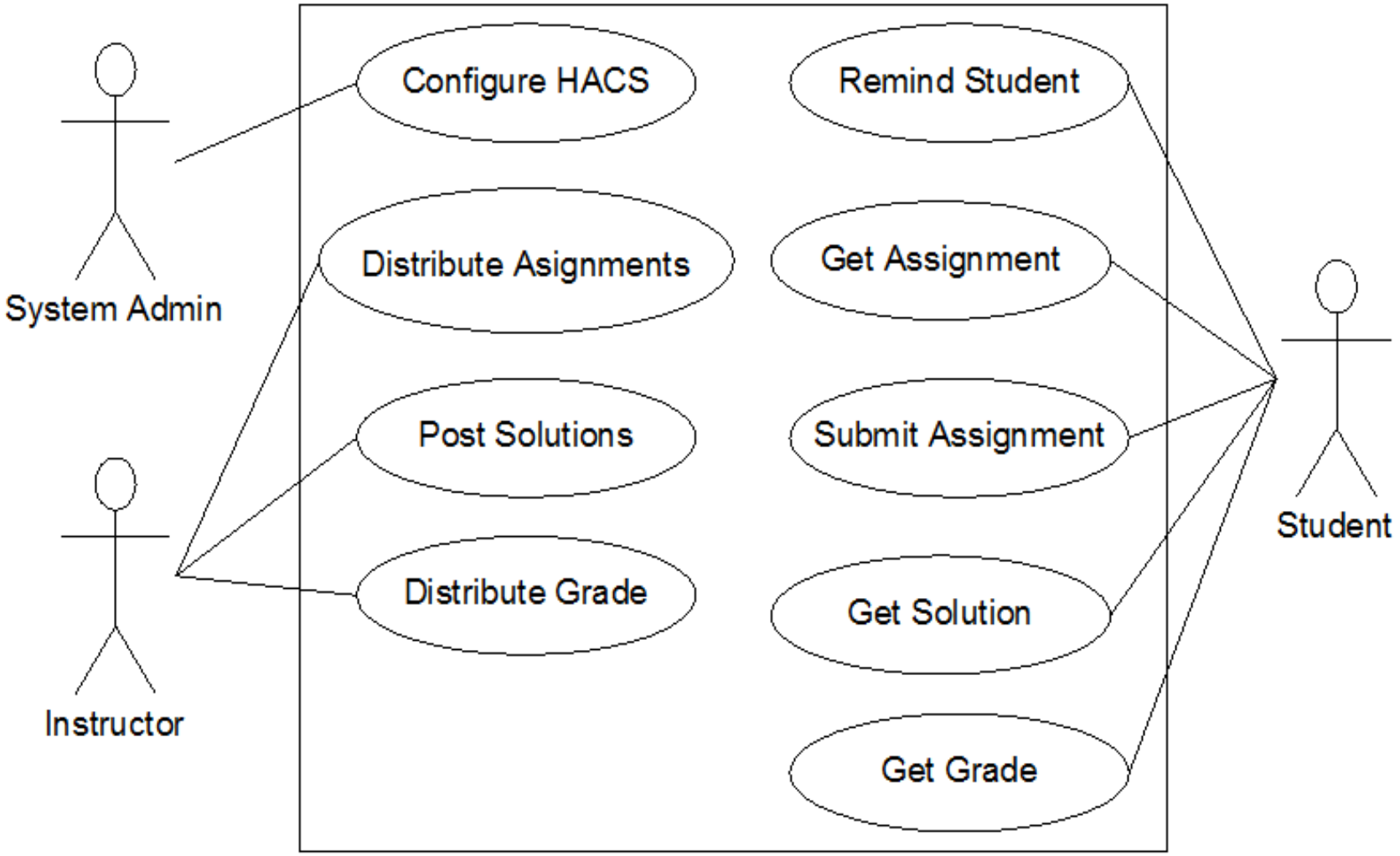
Homework assignment and collection are an integral part of any educational system. Today, this task is performed manually. What we want the homework assignment distribution and collection system to do is to automate this process.

The system will be used by the Instructor/Teacher to distribute the homework assignments, review the students' solutions, distribute suggested solution, and distribute student grades on each assignment.

This system will also help the students by automatically distributing the assignments to the students, provide a facility where the students can submit their solutions, remind the students when an assignment is almost due, remind the students when an assignment is overdue.

Home Assignment distribution and Collection System

Use Case Diagram

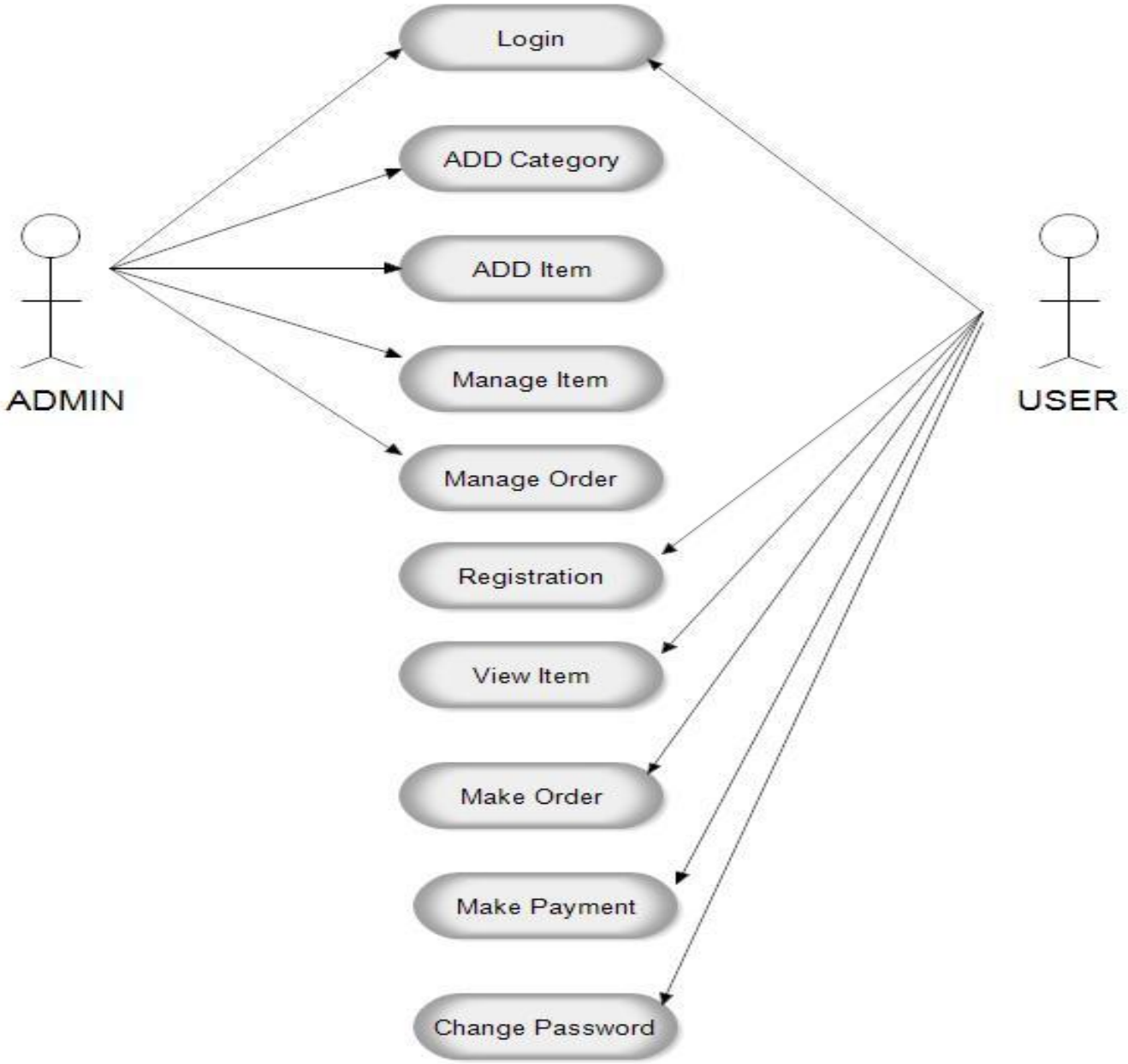


HACS Use Cases Description

Use case: **Distribute Assignments**
Actors: Instructor (initiator)
Type: Primary and essential
Description: The Instructor completes an assignment and submits it to the system. The instructor will also submit the due date and the class the assignment is assigned for.

Cross Ref.: Requirements XX, YY, and ZZ
Use-Cases: *Configure HACS* must be done before any user (Instructor or Student) can use HACS

Use Case Diagram for Online Shopping Website



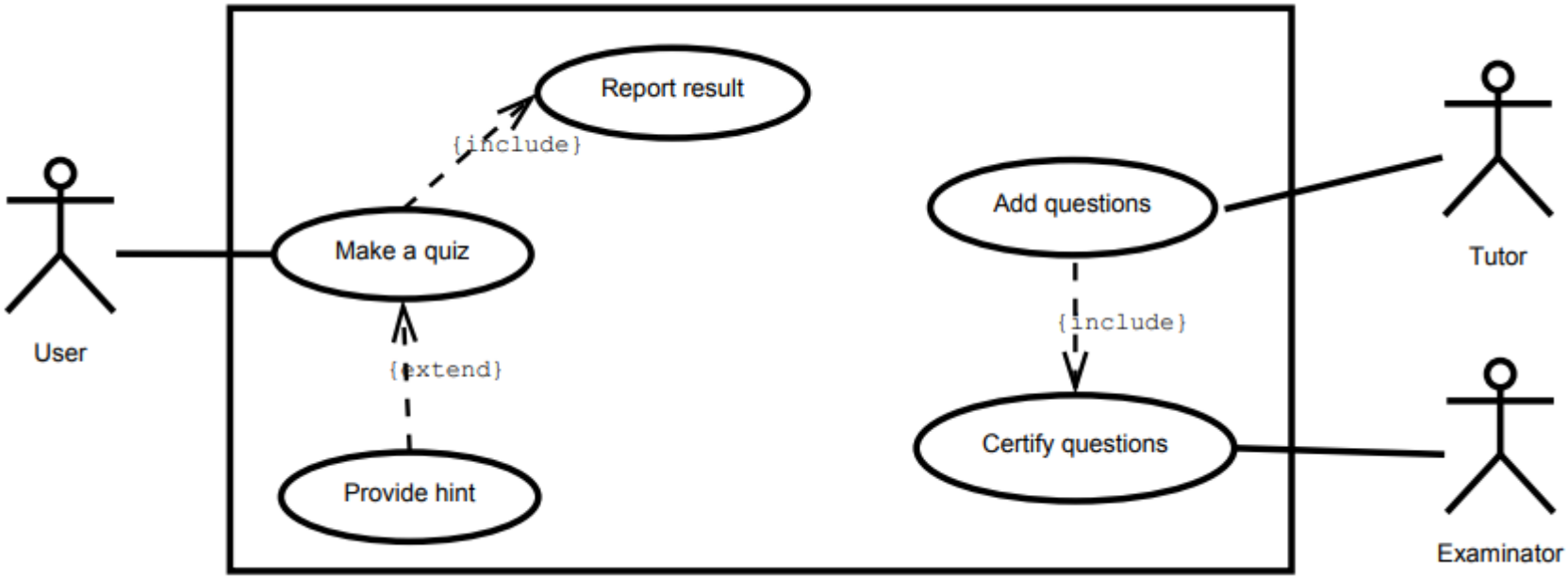
Example: Use Case Scenario

A user can request a quiz for the system. The system picks a set of questions from its database, and compose them together to make a quiz. It rates the user's answers, and gives hints if the user requests it.

In addition to users, we also have tutors who provide questions and hints. And also examinations who must certify questions to make sure they are not too trivial, and that they are sensual.

Make a use case diagram to model this system. Work out some of your use cases. Since we don't have real stake holders here, you are free to fill in details you think is sensual for this example.

Use Case Diagram



Use Case Description

Use case: Make quiz.
Primary actor: User
Secondary actors: -
Pre-condition: The system has at least 10 questions.
Post-condition: -
Main flow:

1. The use-case is activated when the user requests it.
2. The user specifies the difficulty level.
3. The system selects 10 questions, and offers them as a quiz to the user.
4. The system starts a timer.
5. For every question:
   5a. The user selects an answer, or skip. [Extension point]
6. If the user is done with the quiz, or the timer runs out, the quiz is concluded, and [include use case 'Report result'].

References:

1. **Software Engineering A practitioner's Approach** by Roger S. Pressman, 7th edition, McGraw Hill, 2010.
2. **Software Engineering by Ian Sommerville,** 9th edition, Addison-Wesley, 2011
3. **FUNCTIONAL VS NON-FUNCTIONAL REQUIREMENTS: MAIN DIFFERENCES & EXAMPLES** https://theappsolutions.com/blog/development/functional-vs-non-functional-requirements/

Week-5-Lesson-1
Activity Diagram and Sequence Diagram

Abdus Sattar
Assistant Professor
Department of Computer Science and Engineering
Daffodil International University
Email: abdus.cse@diu.edu.bd

Daffodil International University

Topics Covered

- Activity diagrams
- Elements of activity diagram
- Example of activity diagram
- Sequence diagram
- Elements of Sequence diagram
- Example of Sequence diagram

Activity Diagram

- **Activity Diagrams** to illustrate the flow of control in a system and refer to the steps involved in the execution of a use case.
- An activity diagram focuses on condition of flow and the sequence in which it happens.
- Activity diagrams describe the workflow behavior of a system.

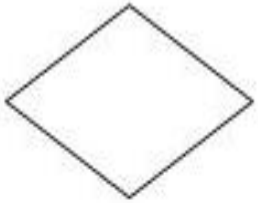
Elements of Activity Diagram

Activity Initial Node

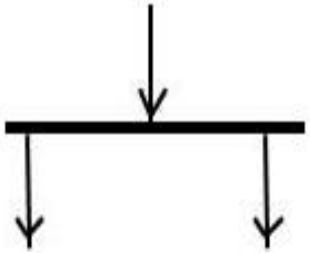
Activity End Node

Transition

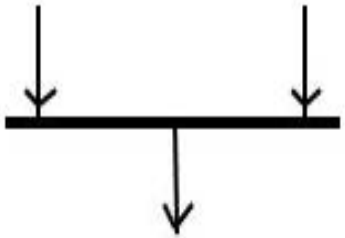
Action



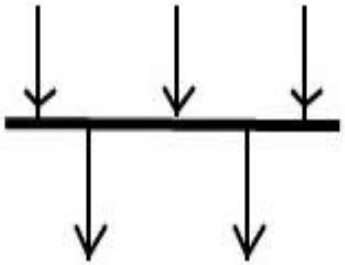
Decision



Fork

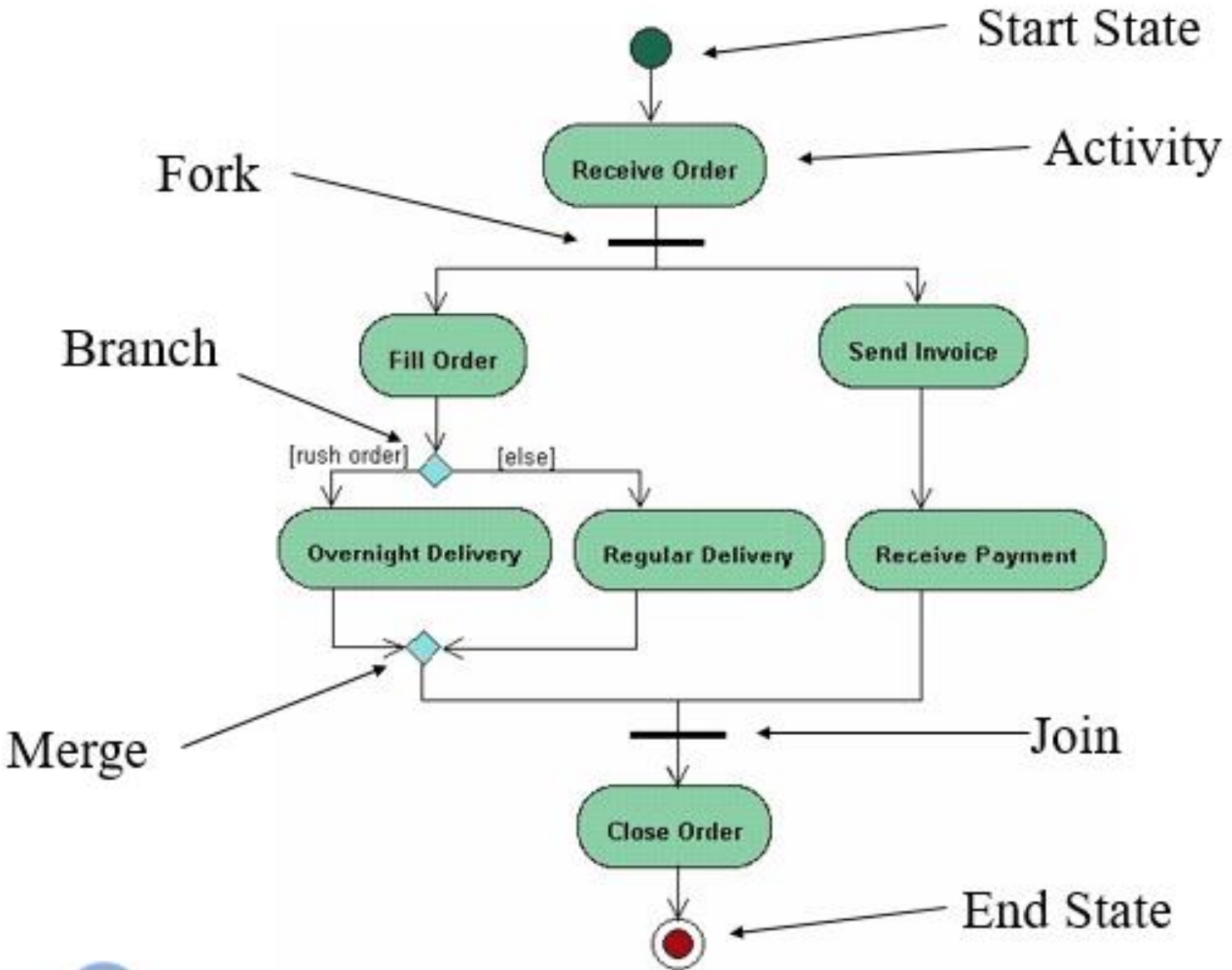


Join

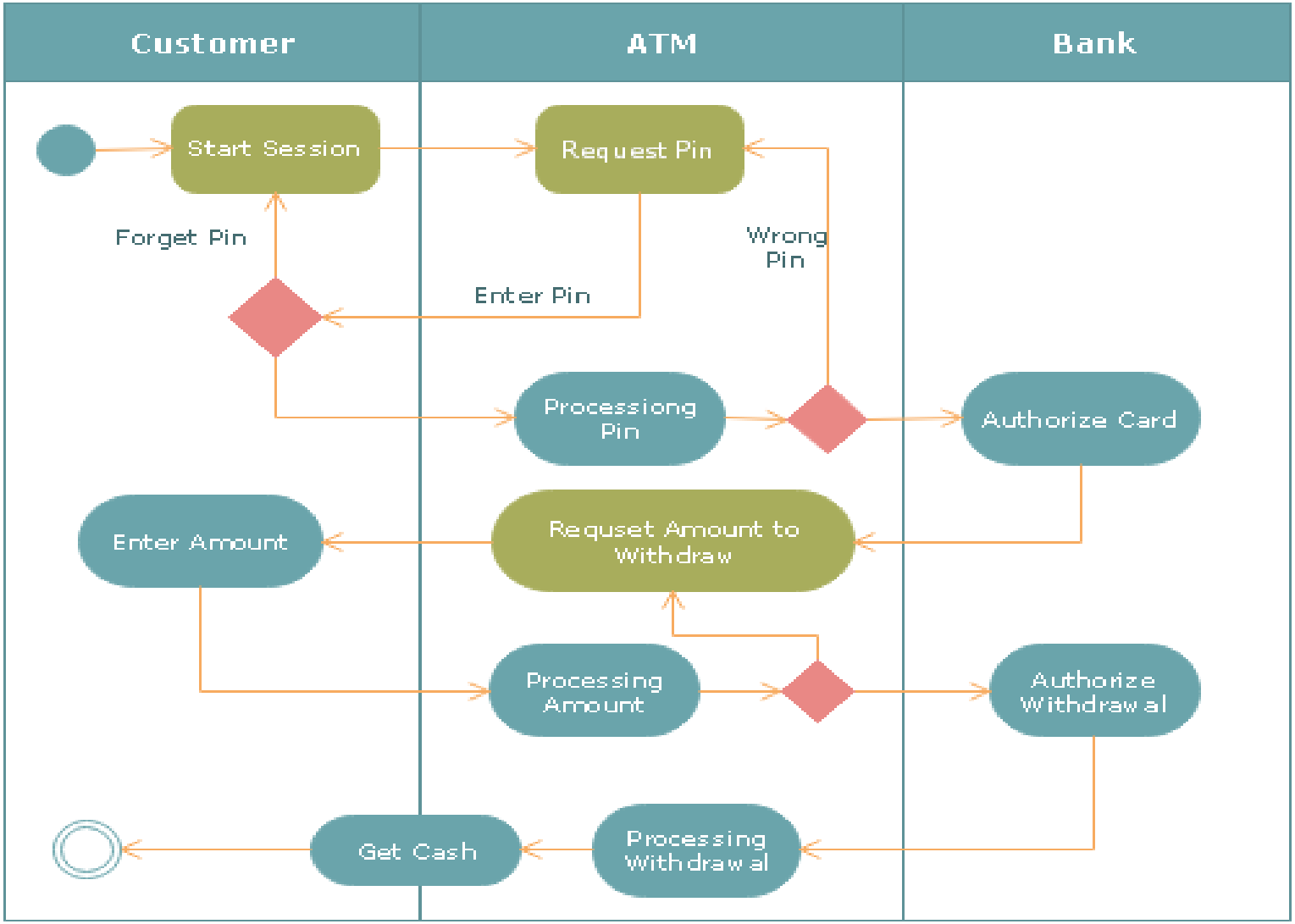


Rendezvous

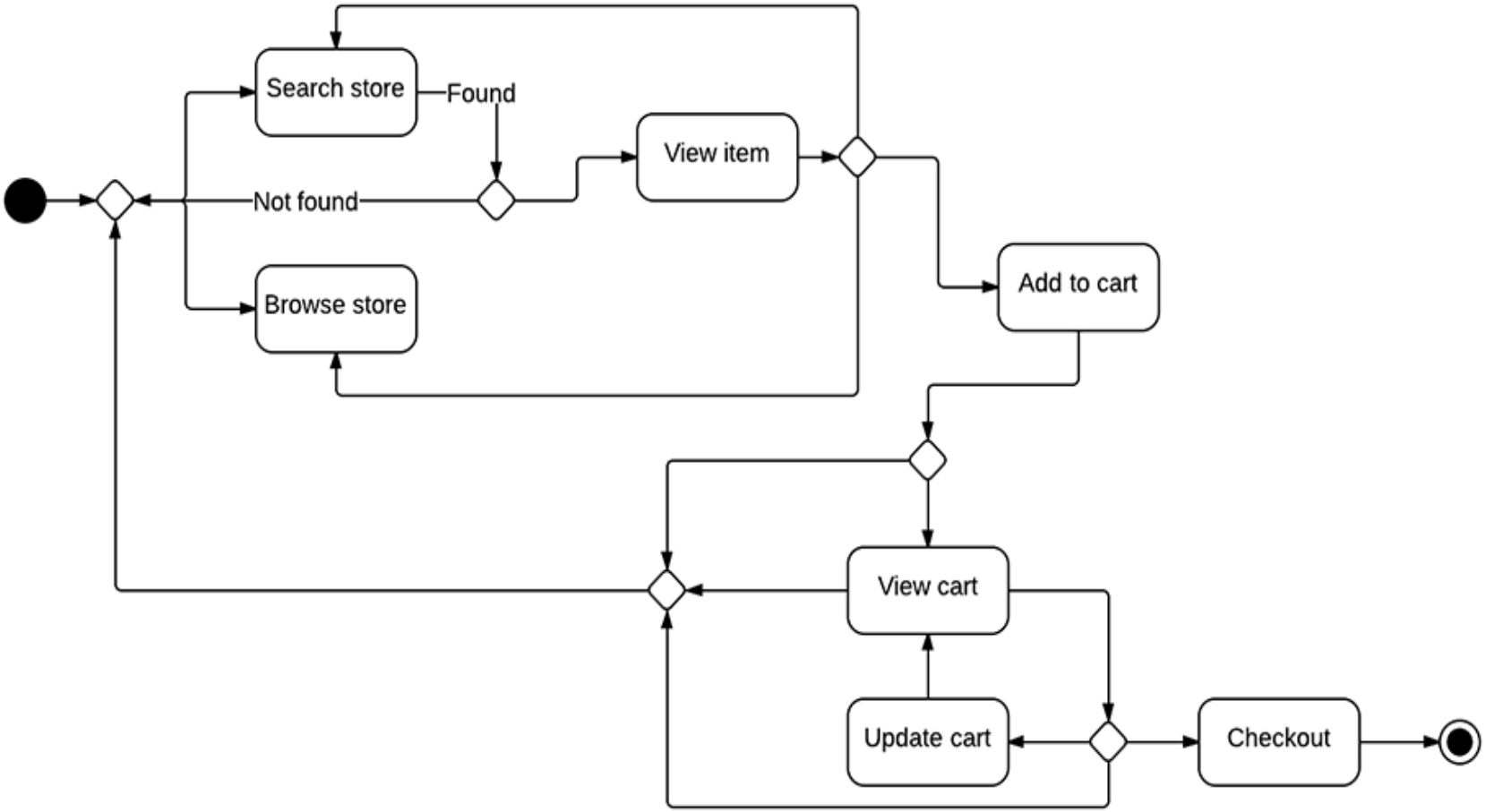
Activity Diagram - Examples



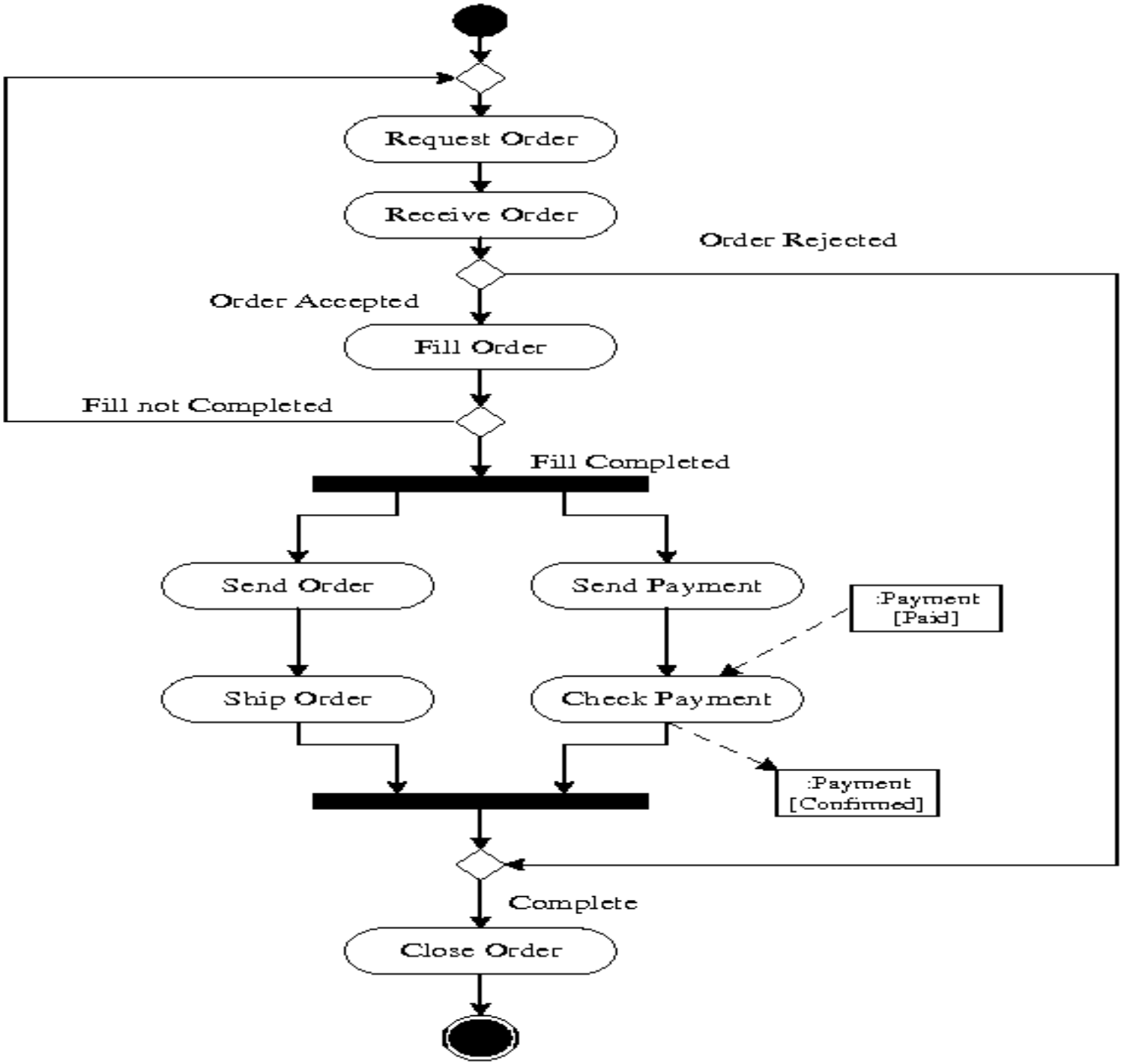
ATM Withdrawal Activity Diagram



Activity Diagram (Search Store)



Activity Diagram for order system



Sequence Diagram

- A **sequence diagram** describes an interaction among a set of objects participated in a collaboration (or scenario), arranged in a chronological order;
- It shows the objects participating in the interaction by their "lifelines" and the messages that they send to each other.
- Sequence diagrams can be useful references for businesses and other organizations.

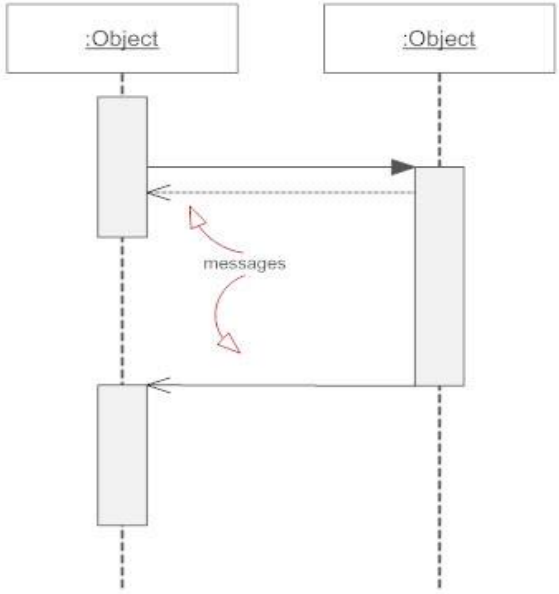
Basic Sequence Diagram Notations

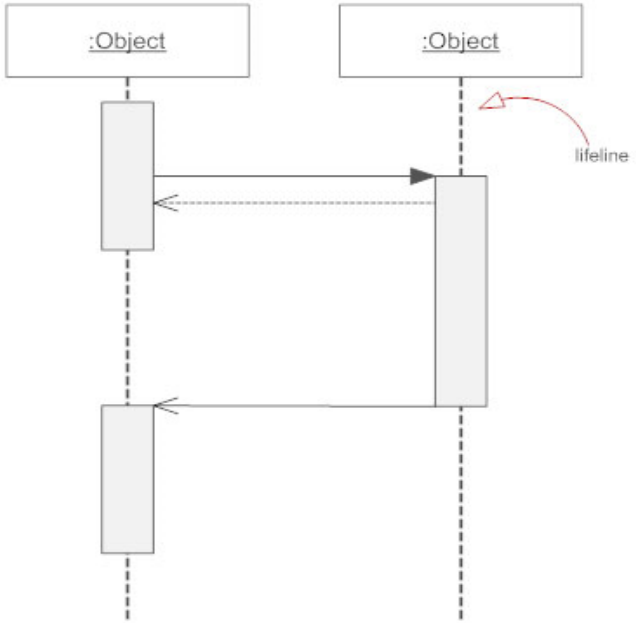
Object symbol

Activation or Execution Occurrence

Actor

:User



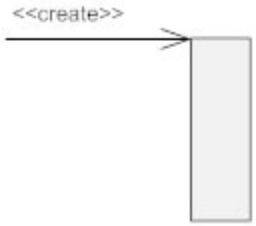


Types of Messages in Sequence Diagrams

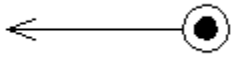
Simple, also used for asynchronous

Synchronous

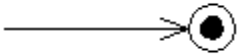
Reply or return message



Create message



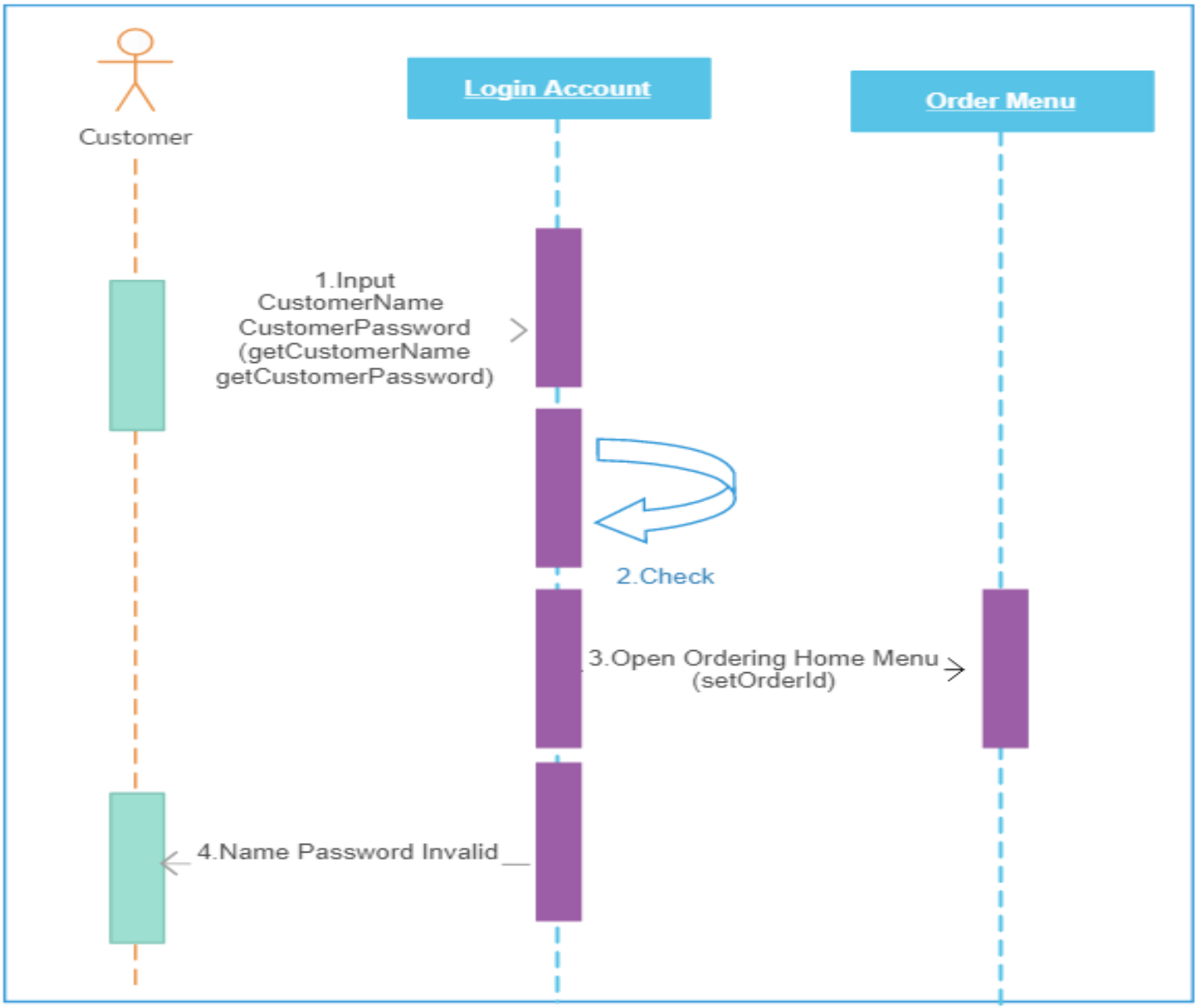
Found message



Lost message

Delete message

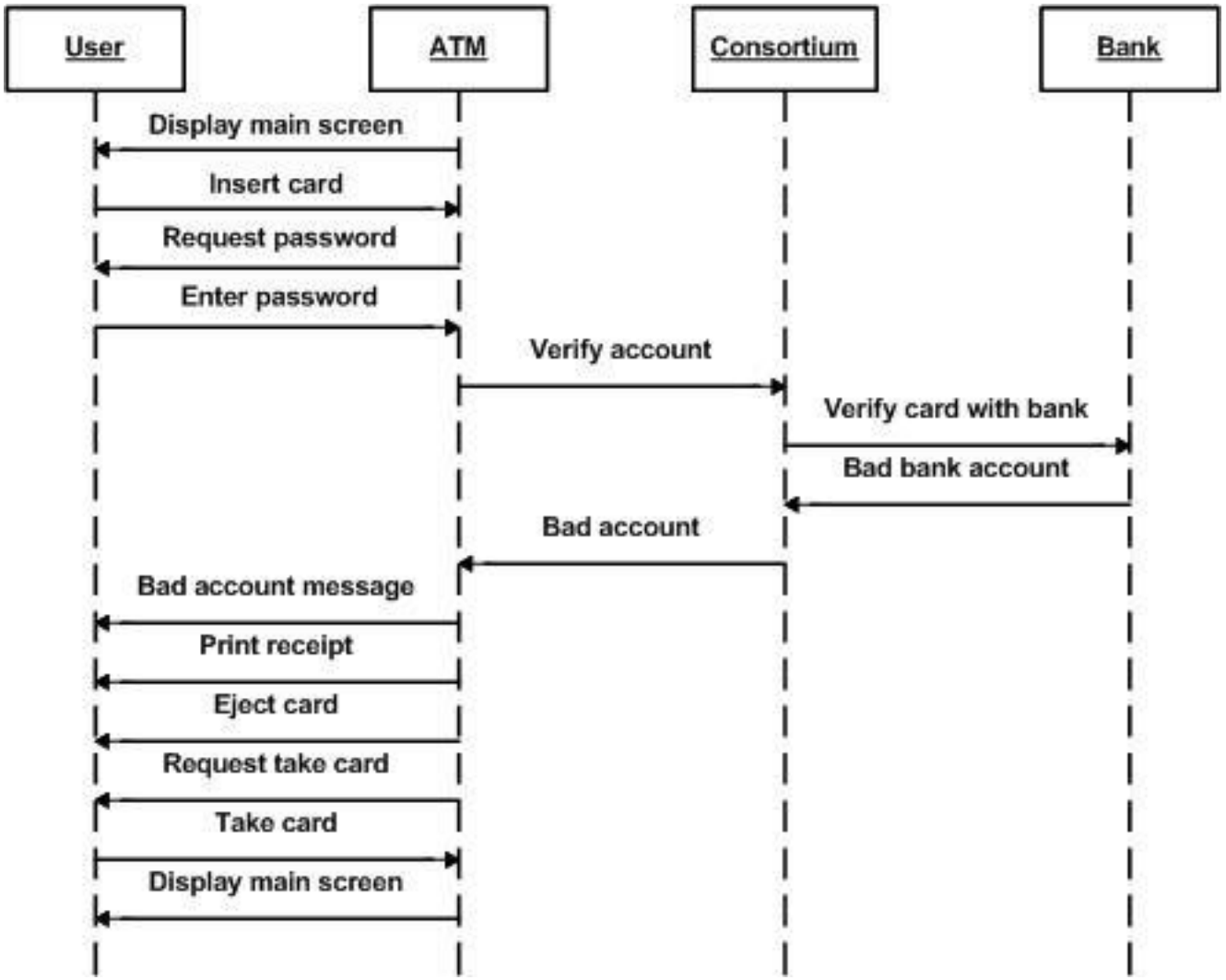
Sequence Diagram for Login



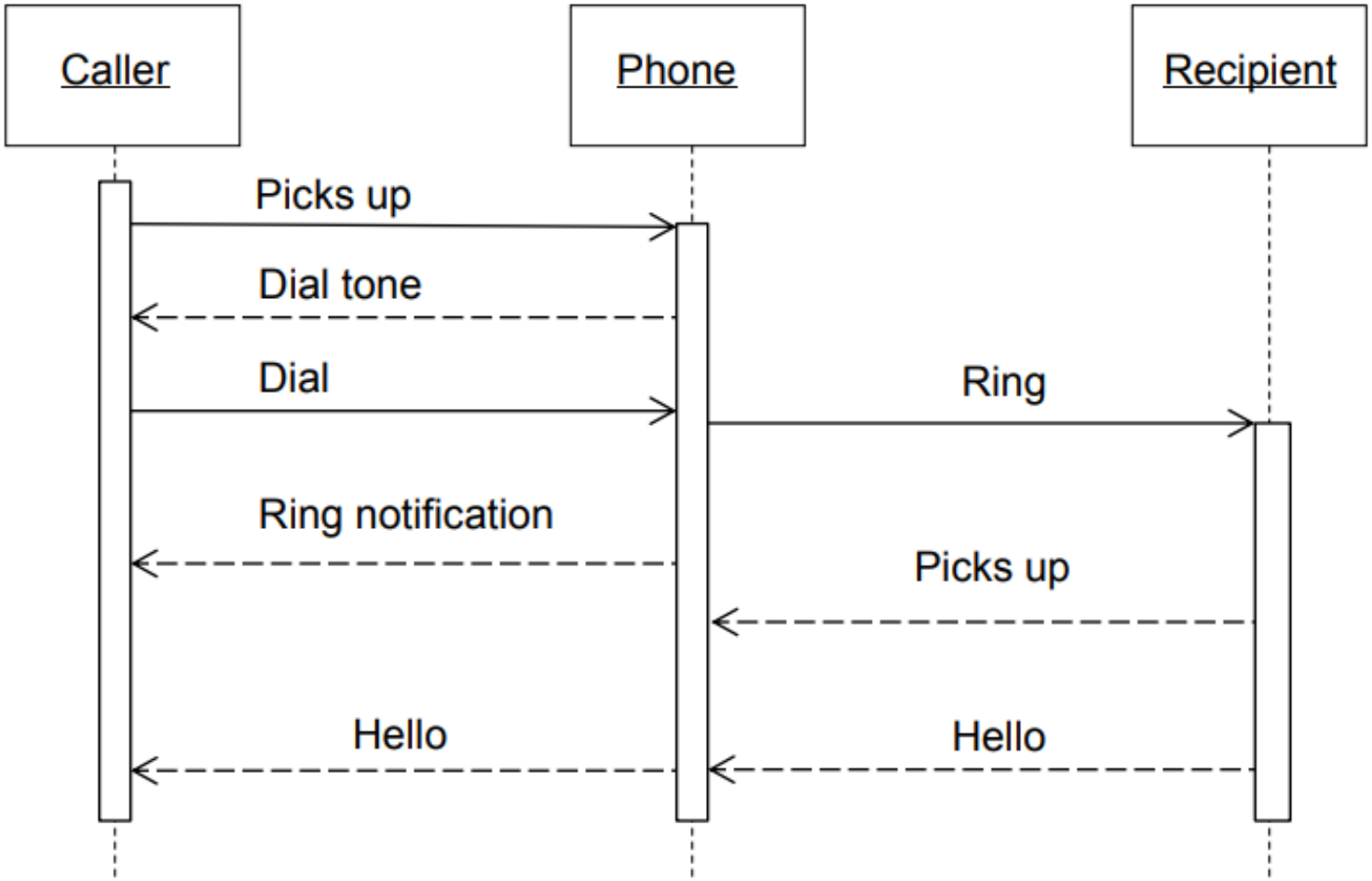
Sequence Diagram of an ATM system

An ATM allows patrons to access their bank accounts through a completely automated process. You can examine the steps of this process in a manageable way by drawing or viewing a sequence diagram. The example below outlines the sequential order of the interactions in the ATM system. Just click to edit the template, and customize the sequence diagram so it suits your own needs.

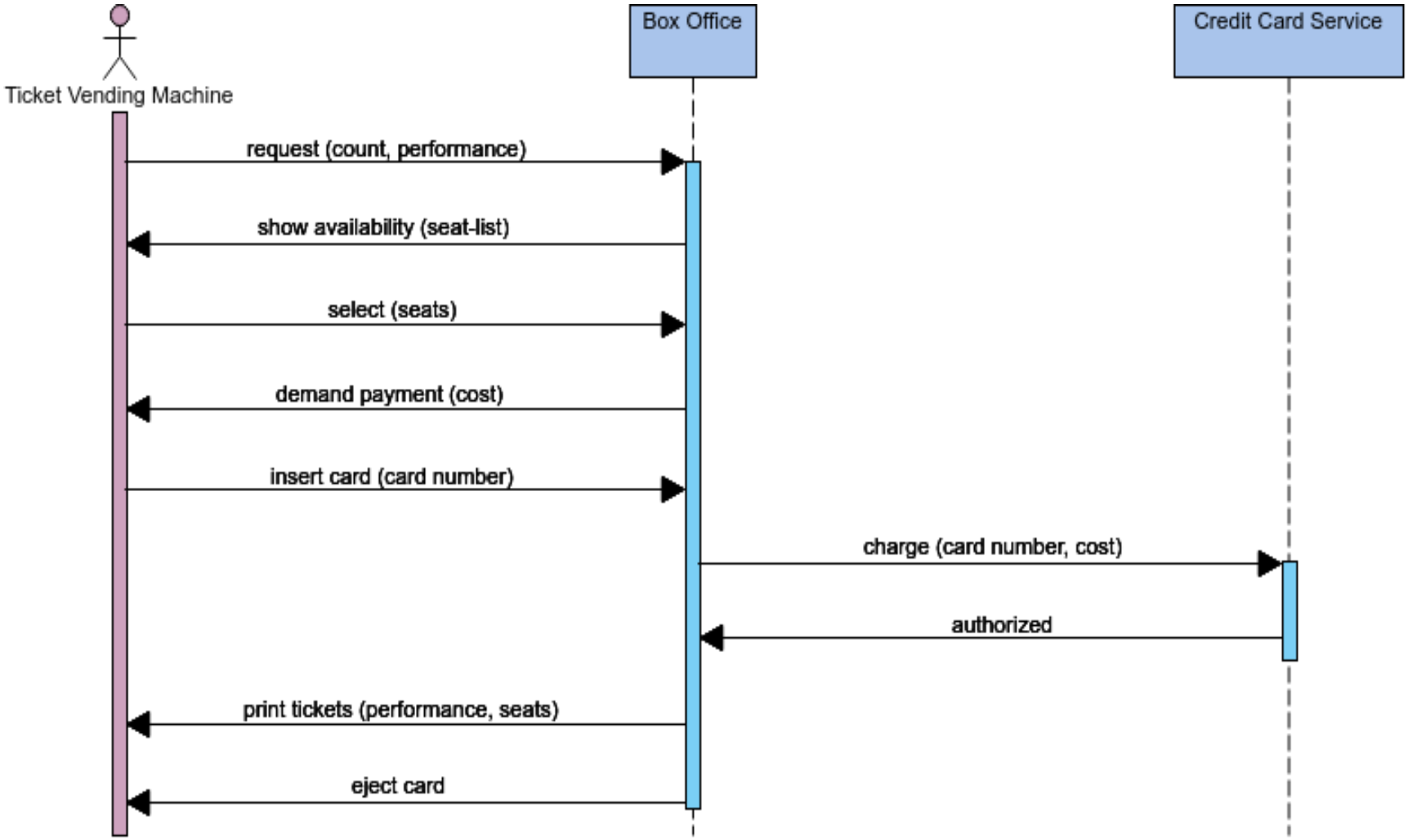
Sequence Diagram of an ATM system



Sequence Diagram (make a phone call)



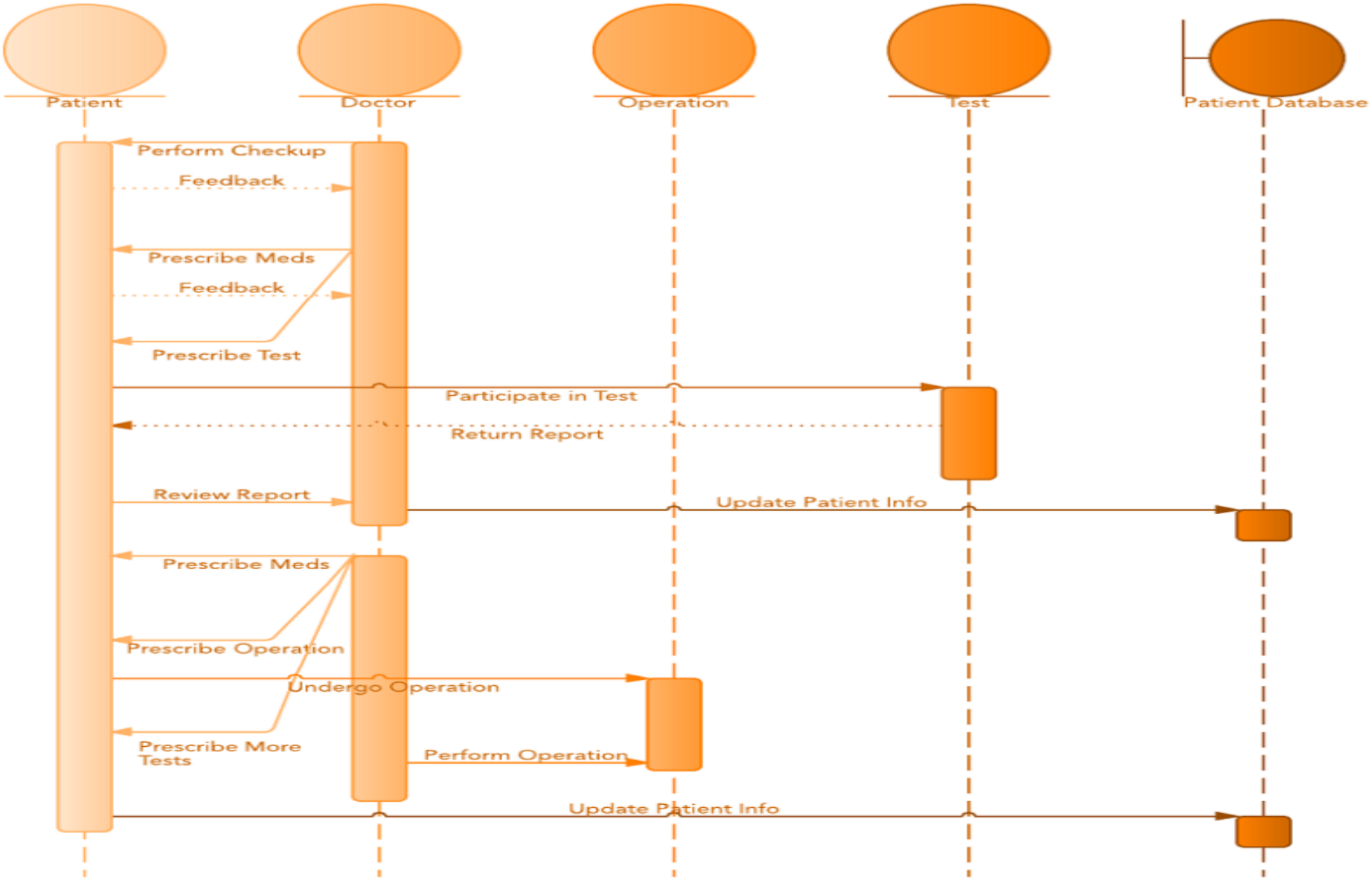
Sequence Diagram Example: Buy Tickets



Sequence diagram of a hospital management system

Technology has completely transformed the field of medicine, as it has with most industries. A hospital information system, also known as a hospital information system, helps doctors, administrators, and hospital staff managing all of the activities and information collected at a hospital, including checkups, prescriptions, appointments, and information on the patients and their caretakers. The diagram below provides a simple view of how the primary processes operate with each other over time. You can use Laucidchart to reshape the diagram any way you choose and to share it with your colleagues or collaborators.

Sequence diagram of a hospital management system



References:

1. **Software Engineering A practitioner's Approach**

   by Roger S. Pressman, 7th edition, McGraw Hill, 2010.

2. **Software Engineering by Ian Sommerville,**

   9th edition, Addison-Wesley, 2011