CHAPTER 2.
LITERATURE REVIEW

2.1 Software Engineering

Software Engineering is one kind of type engineering that have some several aspects which included a process or method to do and tools to build high quality computer-based system/ computer-software with a specified time (Pressman & Maxim, 2015, p. 14).

According to Pressman, Software Engineering have 4 layers:

1. Quality Focus Layer

   Software Engineering should be focus on quality of software, especially in organizational customer quality requirements, developer quality requirements, users' requirements, etc. best practice of quality focus layer is when a software that meet the all of the requirements needed or meet its specification.

2. Process Layer

   Process layer is a foundation of Software Engineering. Process Layer become a connector all of the other layers which could increasing rate of finishing the software on time that has been specified before with proper time spent.

3. Methods Layer

   Methods Layer act as technical guidance of Software Engineering. This layer covers several main aspects which are: requirements analysis, design modeling, program constriction or coding, testing, and maintenance phase of software development.

4. Tools Layer

   Tools Layer acts as supporter which provides functionality to process and method layers. Some combination of tools could become referred to Computer-Aided Software Engineering (CASE). CASE combines software, hardware, and database for development.

2.2 Software Process Framework

Software process is one kind of the most important things to develop software. Process could be described as collection of activities and action that perform some work to create something (Pressman & Maxim, 2015, p. 16). Process frameworks establish a foundation to create software from nothing to kind of finished product. Process framework for software engineering could divide 5 main activities:

1. Communication

   This step is critical aspect of starting a software process because in this step communication is one to understand between customer (individuals or stake holder) what is the requirements to help define all of the feature and function of a software.
2. Planning

   Planning could become a guideline for creator to map all of possibility technical task to be conducted, what risk will be happened, and all of the resource that are needed to develop into next step.
3. Modeling

   Creating a basic sketch of the software, taking a big picture to implement a function on software and its better when the model or sketch is more detail.
4. Construction

   Implementing of modeling activity as a real program with its all functionality.
5. Deployment

   The software is ready to deliver to customer although the program is partially finished or completely finished. After deployment, the software could be revised or getting feedback by customer.

2.3 Software Development Life Cycle (SDLC)

Software Development Life Cycle (SDLC) is used in software engineering to describe all of process of software development. Applying activities and defines process flow with some different model to implement.

2.3.1 Spiral Model

Spiral model is originally proposed by Barry Boehm which is an evolutionary of iterative prototyping. Spiral model provides for fast development to enhance complete version of the software (Pressman & Maxim, 2015, p. 47). Adaptation is one of the keys for spiral model. It can be adapted throughout entire development process. Spiral model focus on risk-driven process model that guide multi-stakeholder to engineer software system. One cyclic approach for growing system could decreasing degree of risk. Spiral model also part of evolutionary process models, which is produce an increasingly more complete development version of software for each iteration.

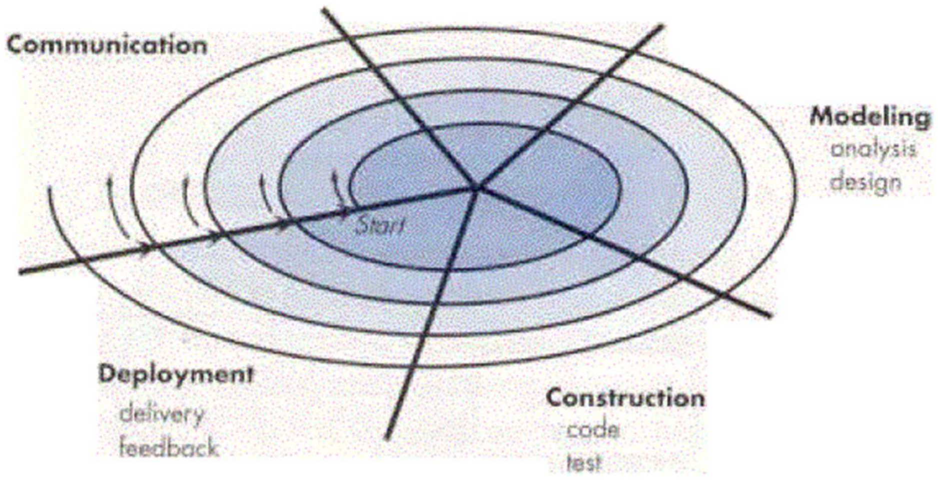


Figure 2.1. Spiral Model

(Source: Software Engineering, Pressman & Maxim, 2015, p.47)

There are 4 main process of Spiral Model:

1. Communication

   Communication is the beginning to identify the requirements. In this process, requirement gathering is the most important part, identify all aspect of requirements. From business sight, what the main point of developing a project and determine the cost, time, and resources for each iteration.

2. Modelling

   Analysis all information that gathered from communication process. Considering risk process and determine alternative for better solution. Modelling also considers about design. Creating the design that help to build a software such as: prototyping, which help to give a big picture of software would be.

3. Construction

   Implementation of modelling process that create a bunch of code which perform as it modeled. All of functionality has to be based on design that has been created before. Construction also consists of test which test, verify, and validate the code. Test is important to make sure that the code is working well on usage.

4. Deployment

   Deployment process is when all of functionality that gathered from communication to construction is deliver to the client/customer. In this case, feedback from customer is the most important part to revise or construct for the next cycle of iteration.

Spiral model starts its process clockwise direction, start from communication process framework and beginning at the center. First circuit of spiral model give earlier stage of result development such as: product specification which is planning result to adjust cost and schedule based on customer. After the process, it will adjust all of the number of iterations required to complete develop a software. Earlier stage of iterative spiral model could be much like a prototype and the next iteration could become more complete module version of the system that will be engineered.

Spiral model can adapt throughout the life of the computer software. First circuit represent a concept of development software system project. Each iteration makes perfect of the module inside the software system. Continuous iteration of spiral model could be a process to enhancement of software system of product enhancement project.

The most flexible and it realistic model of SDLC because it takes from the iterative model combined by structured development. This model considering all of the risk analysis that could be happen. Software development pass through four stage over and over, make a spiral condition until the process is finished.

There’s several advantage using spiral model as SDLC because it adapts easily to evolve of process progress. Developer and customer will react better to risk of evolutionary level. Spiral model using prototyping as risk reduction and also maintain the systematic of development step by step approach. It could act as consideration of technical risk, so the problem of software system development could be predicted and not become problematic.

Beside its advantage, the other hand, spiral model also has disadvantage especially relation with customer. If, the project development is in contract situation, it demands risk assessment expertise and these experts will consider for success of development. Major risk will occur if expertise fails to determine the risk. Another thing, if development is fixed-budget. It’s hard to determine the cost because each circuit is completed, project cost should be revised and revisited. (John, Jackson, & Burd, 2012)

2.4 Database Management System

Database Management System is a software system that helps people to define, create, maintain, and control all of access to database (Connolly and Begg, 2010, p.66). DBMS is a software that connecting user’ database with application program. It also helps the programmer access and use the same data while managing data integration. DBMS provide several features to use:

1. Data Definition Language (DDL)

   DDL gives an ability to user for creating variable and its data type. It also helps to design structure of database that will be use for the next step.
2. Data Manipulation Language (DML)

   DML gives some ability to user to perform CRUD. CRUD stands for Create, Read, Update, and Delete data from database.
3. Data Control Language (DCL)

   DCL gives ability to user for dealing about permissions and other controls of database system
4. Transaction Control Language (TCL)

   DCL gives ability to user for dealing transaction within database.

2.5 Entity-Relationship Diagram (ERD)

Entity-Relationship Diagram (ERD) is a model analysis of a data entities, attributes, and the relationships among the data entities that needs to store information in data storage (Satzinger, 2012, p. 97). Actually, ERD similar to Unified Modeling Language (UML) but have their own functionality.

ERD has main point of usages which are to determine database design where the requirements that have gathered implement to ERD focusing what are database likely to be displayed. ERD also could help analyzing the problem more convenience because of we know the basic database or the database design of the program.

According to Satzinger, ERD has several important components to know. Rectangle represent data entities and lines between rectangles show the relationship between data entities. Symbols on a line have a specific meaning to describe what the relationship between entities is.

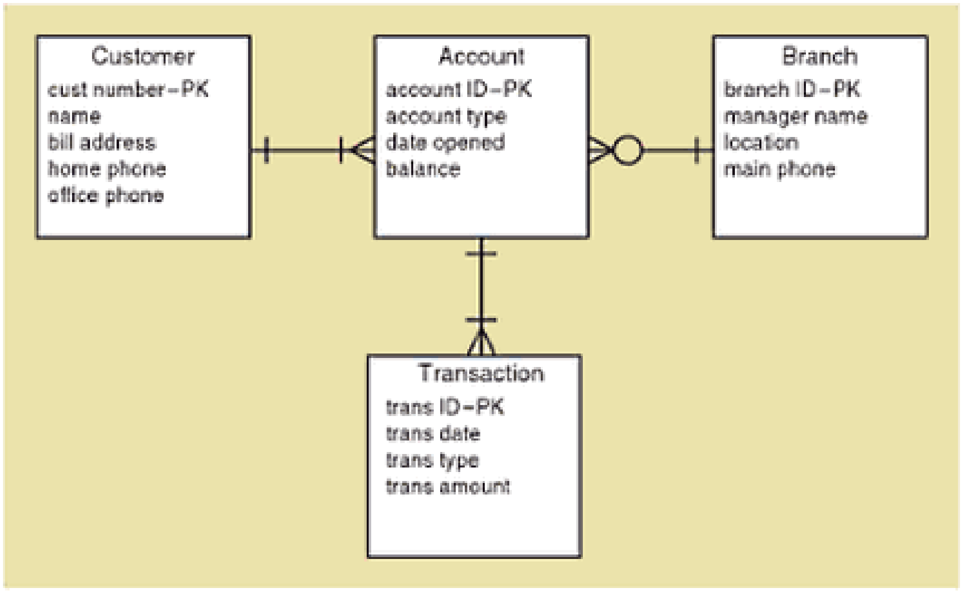


Figure 2.2 ERD for bank with many branches

(Source: Systems Analysis and Design, Satzinger, 2012, p. 100)

Rectangle that represent a data entity have some properties on it. Data entity is an object that are break down into smallest part. Each property or attribute is used to represent data entity. Example from above, Customer data entity. Customer has several properties such as: cust number, name, bill address, home phone, office phone. Beside cust number, there’s a symbol of PK, which means Primary key. Each data entities should have a Primary key for determine its uniqueness. Primary key is critical concept to have for ERD, without it, ERD will become complicated and confusing because it makes hard to determine which is foreign key and at the end, relational database concept will not work.

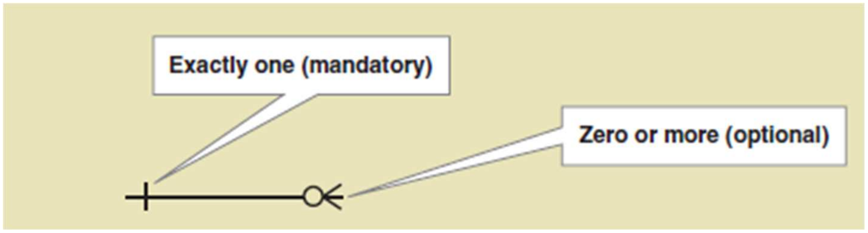


Figure 2.3 Cardinality symbols of ERD relationships

(Source: Systems Analysis and Design, Satzinger, 2012, p. 99)

This is a symbol of relationship for each rectangle. A left line with a strip on it represents exactly one or have to be one in a relationship. A right line with circle and crow’s feet represent zero or more relationship, which is it could be nothing, one, or more than one of data entities.

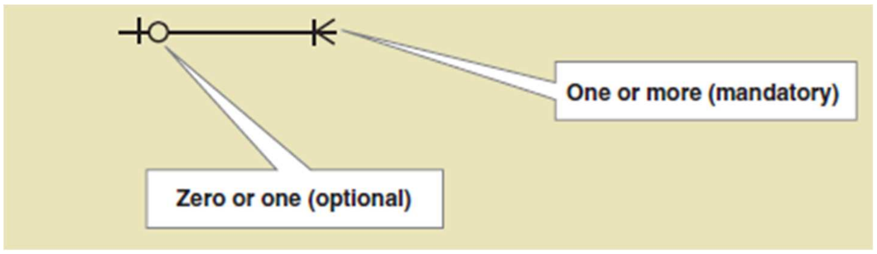


Figure 2.4 Cardinality symbols of ERD relationships 2

(Source: Systems Analysis and Design, Satzinger, 2012, p.99)

For this left line with strip and circle on it symbolize that zero or one relationship, which is could be nothing at all or one relationship of data entities. A right side of line with a strip and crow’s feet on it represent one or more relationship of data entities, zero or nothing is not allowed for data entities relationship.

2.6 Unified Modeling Language (UML)

UML is based on Object Oriented concepts that doesn’t designed for one particular Object-Oriented Programming Language. UML can be used for all software development from various condition such as: complexity, real time, volume, etc. (Seidl, Scholz, Huemer, & Kappel, 2015, p. 14).

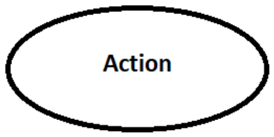
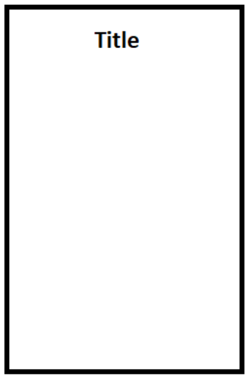
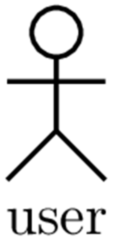
In UML, a model is represented into diagram. These diagrams will show users use the functionality and showing the structure of the system development without specifying detail concrete implementation (Seidl, Scholz, Huemer, & Kappel, 2015, p. 15).

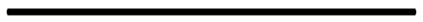
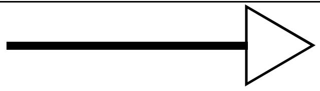
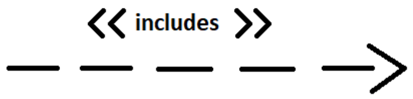
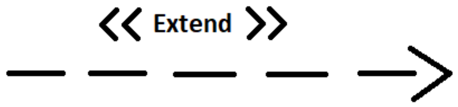
In general, UML define to two major kind of UML diagram: Structure diagram and Behavior diagram. Structure diagram explain static structure, which describe all of elements is related with each other and represent meaningful concept of the system. The other hand, behavior diagram explains the details of behavior which affect how states of object changer over time.

2.6.1 Use Case

Use case is a fundamental concept of Object-Oriented Programming that defines all of possible usage scenarios. Describing in simple method that cover who using the system, what is being described on system, and what action that user will be able to do with the system. In shorthand, use case will describe all of the functionality of the system developed.

Table 2.1 Notation elements for use case diagram

No	Symbols	Description
1		A use case describing what is the function is expected to perform. It described as ellipse with simple and clear action description in it.
2		Automation boundary is the boundary that separate between use case and actor. Separate who will operate the system and what system ability to do with actor action. Automaton boundary described as vertically rectangle with title. Title describes the name of system.
3		Stick figure represent an actor or user in the use case diagram. Actor or user represent the actual person who using the system. A stick figure should be named to know who is exactly is the user of the system, could be customer, administrator, or etc. User or actor should always outside from automation boundary of

No	Symbols	Description
		the system. An actor isn't always a human, but non-human also such as e-mail server.
4		A line is the most common relationship that represent or connecting between user or actor with the system on specific action or case.
5		An arrow line symbolizes as generalization or specialization. Could be used between actors or between use case. Sub actor or sub use case is inherit from its parent, and has another ability to do / perform something special.
6		A dashed arrow with <<includes>> is a relationship between cases, which is original case is not complete without included one. <<includes>> relationship. It defines that from A use case is should be executed to perform B use case
7		A dashed arrow with << Extend >> is a relationship between cases that 1 use case is optionally dependently on another use case. Which means extended use case is could be done or not done at all.

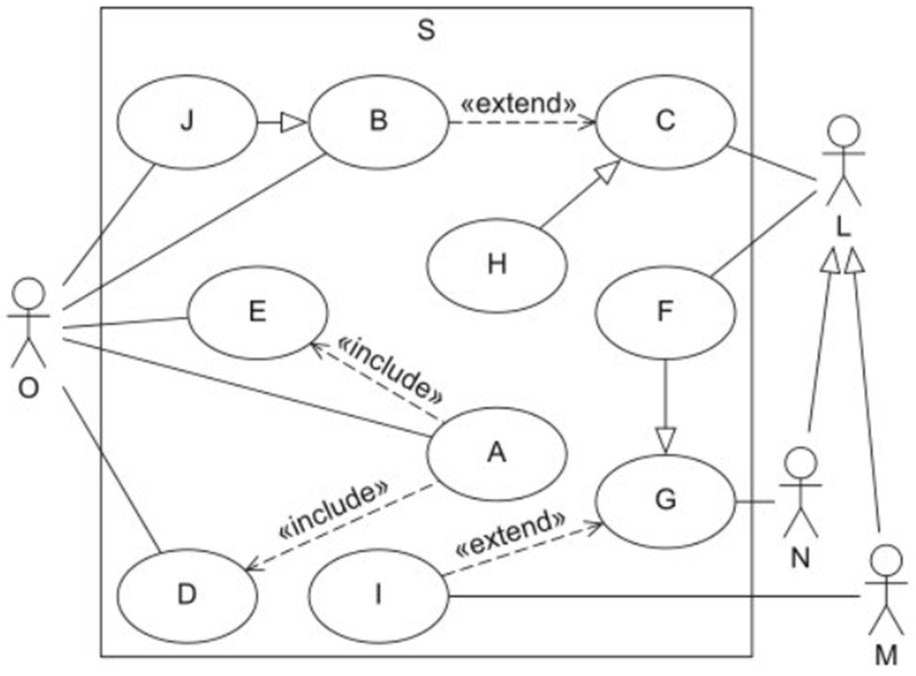


Figure 2.5 Relationship in use case diagram

(Source: UML@Classroom, Seidl, Scholz, Huemer, & Kappel, 2015, p.33)

Explanation:

1. Use case A includes use cases of E and D. Actor O perform 3 uses case. If actor O perform use case A, that means use cases E and D should be executed too.
2. Use case H inherits from use case C and actor L should be perform use case H by doing use case C. Actor N and M has ability to do the same as L does because of N and M are the generalization of L which is has same ability to perform C.
3. Use case J is inherit from use case B. Actor O could directly access to perform B and J use cases. In this case, two actors in the role O are involved on execution of J.
4. The use case F inherits from the use case G. As a result of the inheritance relationship, an actor N is involved in the execution of use case F. For F, an association with the actor L is also modeled directly. Therefore, an actor N and, due to the inheritance relationship of the actors L, N, and M, either an actor L or an actor M or an additional

actor N is involved in the execution of F. Use case I extends use case F because use case F inherits from use case G and as I extends use case G, relationship will be passed on to F.

5. Use case I extends use case F because use case F inherits from use case G and as I extends use case G, relationship will be passed on to F.
6. Use case J extends the use case H because relationship of inheritance from B to J and from C to H.

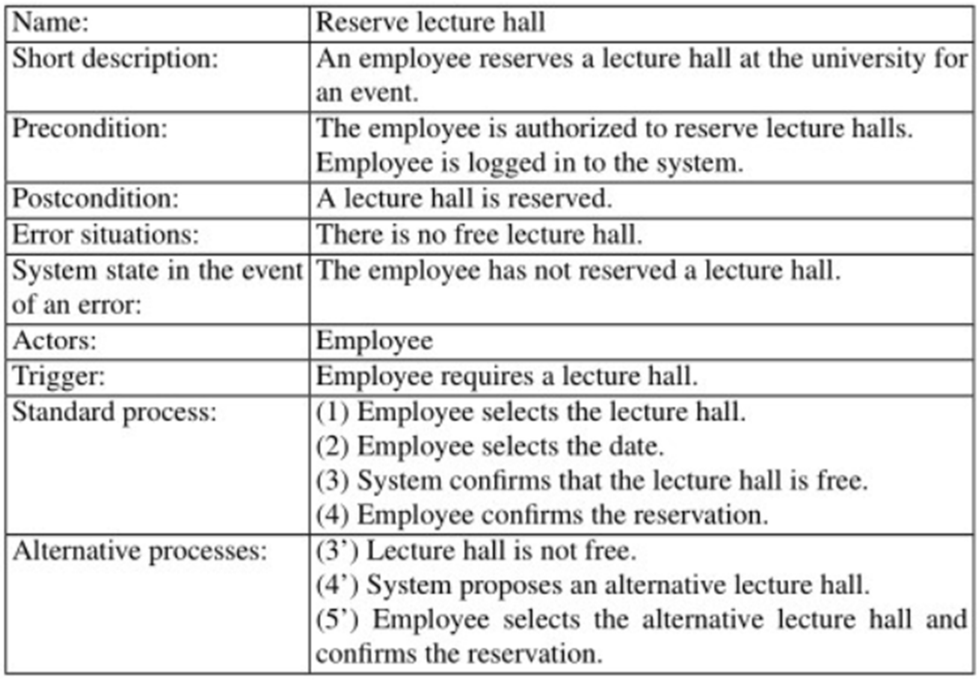
2.6.2 Use Case Description

Use case description is a use case that ensure to describe use cases as clear and concise as possible. It also helps to improve comprehension of system and clearly described statement for user's requirements.

Alstair Cockburn presents structured that has to be there on use case description:

1. Name
2. Short description
3. Precondition → prerequisite for successful execution
4. Postcondition → system state after successful execution
5. Error situation → error relevant to problem domain
6. System state on occurrence of an error
7. Actor that communicate with the use case
8. Trigger → event that start the use case
9. Standard process → individual step to be taken
10. Alternative process → deviation from standard process

Table 2.2 Use case description for Reserve lecture hall



Name:	Reserve lecture hall
Short description:	An employee reserves a lecture hall at the university for an event.
Precondition:	The employee is authorized to reserve lecture halls. Employee is logged in to the system.
Postcondition:	A lecture hall is reserved.
Error situations:	There is no free lecture hall.
System state in the event of an error:	The employee has not reserved a lecture hall.
Actors:	Employee
Trigger:	Employee requires a lecture hall.
Standard process:	(1) Employee selects the lecture hall. (2) Employee selects the date. (3) System confirms that the lecture hall is free. (4) Employee confirms the reservation.
Alternative processes:	(3') Lecture hall is not free. (4') System proposes an alternative lecture hall. (5') Employee selects the alternative lecture hall and confirms the reservation.

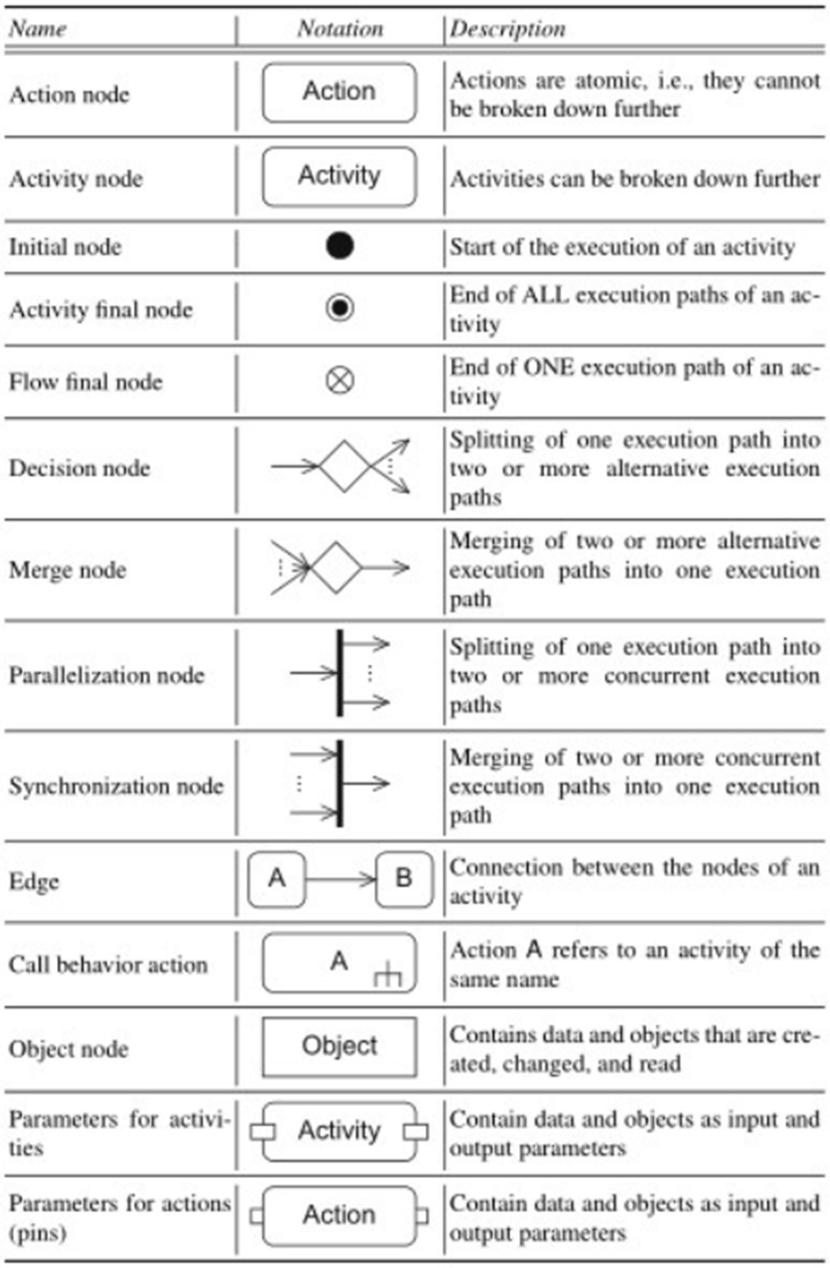
(Source: UML@Classroom, Seidl, Scholz, Huemer, & Kappel, 2015, p.36)

This description shows of the use case Reverse lecture hall in a student administration system. All of condition on this use case is considered. Starting from the process, alternative process, error situation, and postcondition occur. For example, it could be possible to reserve a lecture hall where an event is already taking place—this makes sense if the event is an exam that could be held in the lecture hall along with another exam, meaning that fewer exam supervisors are required (Seidl, Scholz, Huemer, & Kappel, 2015, p. 36).

2.6.3 Activity Diagram

Activity diagram is a diagram that describe a procedural processing of a system. Which means, explain all about control flow and data flow step by step that required to implement an activity (Seidl, Scholz, Huemer, & Kappel, 2015, p. 141).

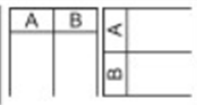
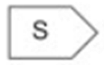
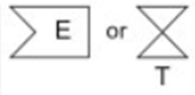
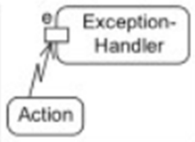
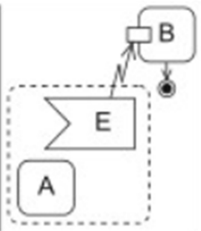
Table 2.3 Notation elements for activity diagram



Name	*Notation*	*Description*
Action node	Action	Actions are atomic, i.e., they cannot be broken down further
Activity node	Activity	Activities can be broken down further
Initial node	●	Start of the execution of an activity
Activity final node	◉	End of ALL execution paths of an activity
Flow final node	⊗	End of ONE execution path of an activity
Decision node	◇	Splitting of one execution path into two or more alternative execution paths
Merge node	◇	Merging of two or more alternative execution paths into one execution path
Parallelization node		Splitting of one execution path into two or more concurrent execution paths
Synchronization node		Merging of two or more concurrent execution paths into one execution path
Edge	A → B	Connection between the nodes of an activity
Call behavior action	A	Action A refers to an activity of the same name
Object node	Object	Contains data and objects that are created, changed, and read
Parameters for activities	Activity	Contain data and objects as input and output parameters
Parameters for actions (pins)	Action	Contain data and objects as input and output parameters

(Source: UML@Classroom, Seidl, Scholz, Huemer, & Kappel, 2015, p.165)

Table 2.4 Notation elements for activity diagram part 2

Name	*Notation*	*Description*
Partition		Grouping of nodes and edges within an activity
Send signal action		Transmission of a signal to a receiver
Asynchronous accept (time) event action		Wait for an event E or a time event T
Exception handler		Exception handler is executed instead of the action in the event of an error e
Interruptible activity region		Flow continues on a different path if event E is detected

(Source: UML@Classroom, Seidl, Scholz, Huemer, & Kappel, 2015, p.166)

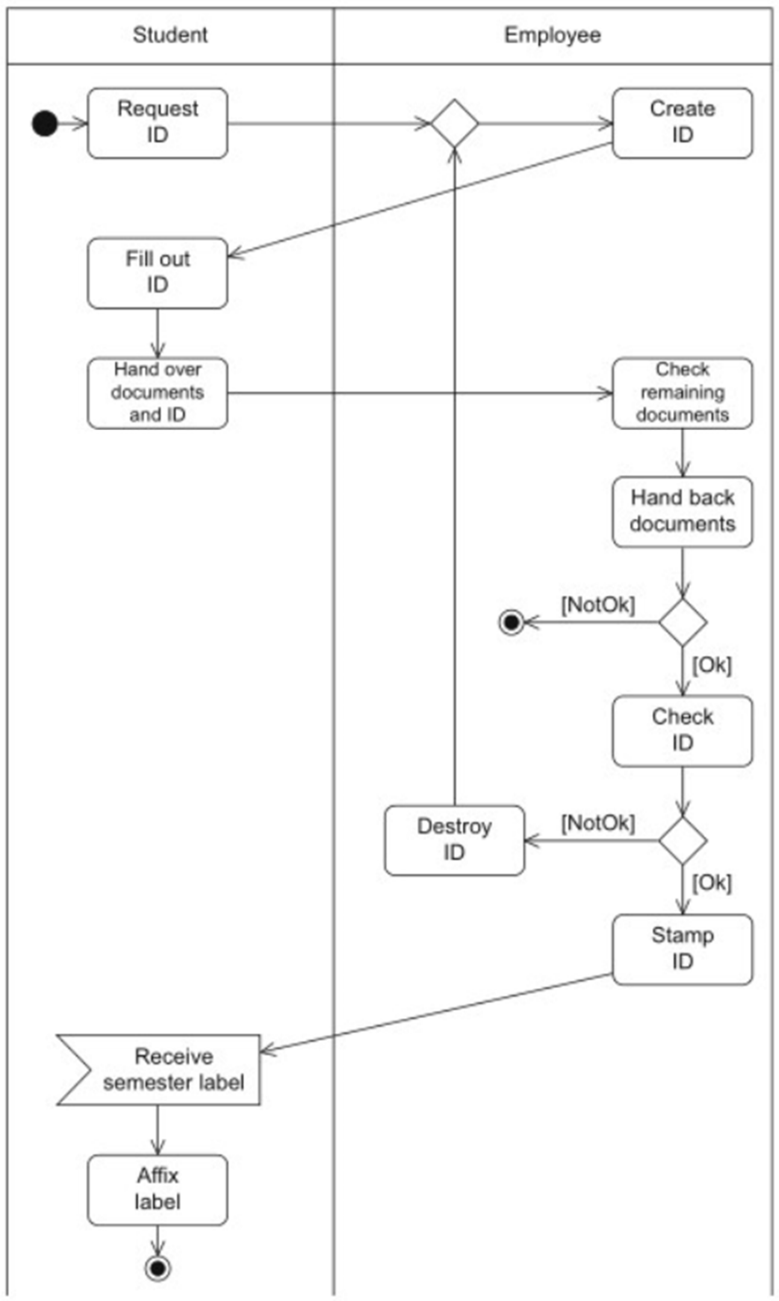


Figure 2.6 Activity diagram "Issue student ID"

(Source: UML@Classroom, Seidl, Scholz, Huemer, & Kappel, 2015, p.163)

This is activity diagram for Issue Student ID use case. From the beginning of initial state symbol, Student start to request ID, then Employee creates ID for student. Student fill out the ID and giving all of the document required and the ID to employee. Employee will check the remaining documents and hand over back the document. If document is not appropriate, the activity will be ended, but if it's ok, employee will check the ID. The ID will be check either ok or not ok. If it not ok, ID will be destroyed and re-create the ID, otherwise employee will stamp the ID. Then student will receive semester label and affix the label. After affix the label, it will end the process of activity diagram.

2.6.4 Class Diagram

Class diagram is static structure that describes the elements of the system and its relationships between them (Seidl, Scholz, Huemer, & Kappel, 2015, p. 49). Class diagram could help creating conceptual view of the system. In class diagram, there's several aspects that is important to recognize. An object that has several attributes and methods become a class and relationships between classes.

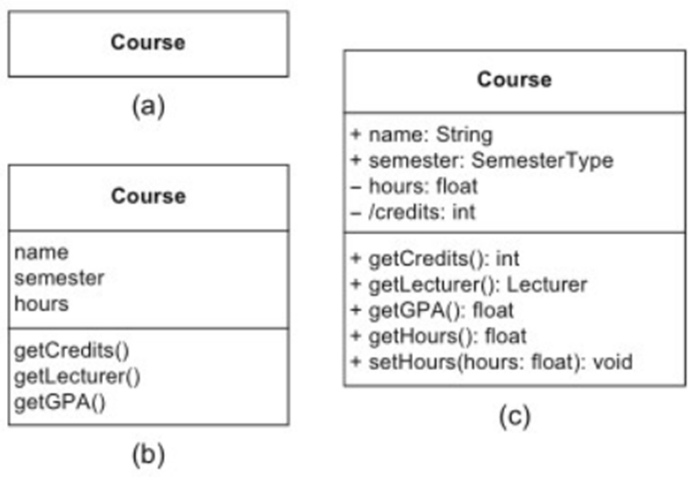


Figure 2.7 Representation of a class and its characteristics
(Source: UML@Classroom, Seidl, Scholz, Huemer, & Kappel, 2015, p.53)

Table 2.5 Visibilities

Name	Symbol	Description
public	+	Access by objects of any classes permitted
private	−	Access only within the object itself permitted
protected	#	Access by objects of the same class and its subclasses permitted
package	∼	Access by objects whose classes are in the same package permitted

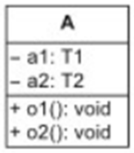
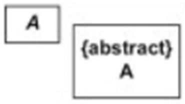
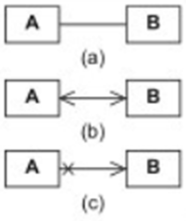
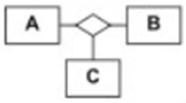
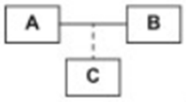
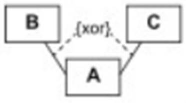
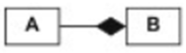
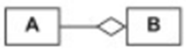
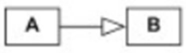
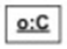
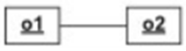
(Source: UML@Classroom, Seidl, Scholz, Huemer, & Kappel, 2015, p.59)

From figure 2.7 section A**,** defines an object that become a title of each class. Section B shows each class has several attributes or properties of and methods that defines of Course class. Section C shows the complete of one class diagram, which defines all of attributes, method, data type, return type, and access modifier.

Associations is relationships between classes model. Communication between classes occur defined as several concept that will be describes below.

Table 2.6 Notation elementes of the class diagram

(Source: UML@Classroom, Seidl, Scholz, Huemer, & Kappel, 2015, p.84)

Name	*Notation*	*Description*
Class		Description of the structure and behavior of a set of objects
Abstract class		Class that cannot be instantiated
Association		Relationship between classes: navigability unspecified (a), navigable in both directions (b), not navigable in one direction (c)
N-ary association		Relationship between *N* (in this case 3) classes
Association class		More detailed description of an association
xor relationship		An object of A is in a relationship with an object of B or with an object of C but not with both
Strong aggregation = composition		Existence-dependent parts-whole relationship (A is part of B; if B is deleted, related instances of A are also deleted)
Shared aggregation		Parts-whole relationship (A is part of B; if B is deleted, related instances of A need not be deleted)
Generalization		Inheritance relationship (A inherits from B)
Object		Instance of a class
Link		Relationship between objects

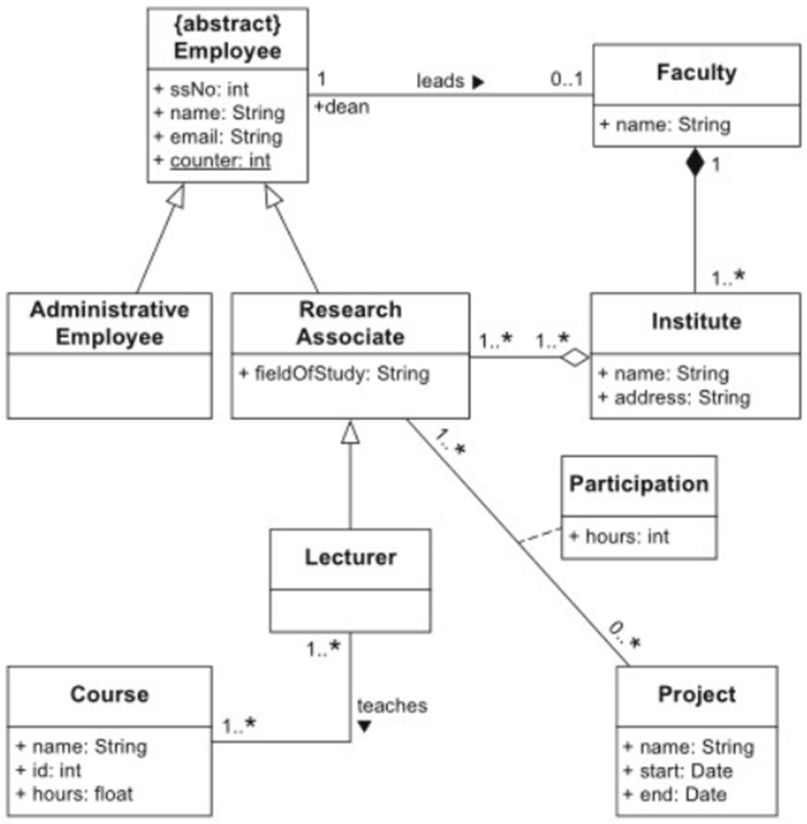


Figure 2.8 Class diagram of the information system of a university
(Source: UML@Classroom, Seidl, Scholz, Huemer, & Kappel, 2015, p.80)

This is a class diagram for information system of university could be described as:

1. Administrative employee class and Research Associate class inherit from abstract class of Employee.
2. Employee abstract class leads zero or one faculty, otherwise faculty is leaded by 1 employee.
3. Research associate class is part of institute that if institute was deleted, related instances of Research Associate doesn’t have to be deleted.
4. An Institute class is part of faculty, which mean if there’s no faculty class, Institute class will be deleted also because of its composition.
5. Lecturer class inherit from Research Associate class. Lecturer teaches one or more courses and course will be taught by 1 or more lecturer.

6. Research Associate class has zero or more Project class, otherwise a project has 1 or more Research Associate class.
7. There's association class called Participation class between Research Associate class and Project class which more detail description of an association class.

2.6.5 Sequence Diagram

Sequence diagram describes kind of interaction between objects with full specific task. It will describe like a readable flow or chronological order of information or message. Sequence diagram contains several main points: lifeline that shows time occurrence to interact or communicate with other, types of massage (synchronous, asynchronous, response massage, and create massage). Chronological information usually starts from top to bottom, but with help of controller structure, it will enable to sequence diagram to control the interaction.

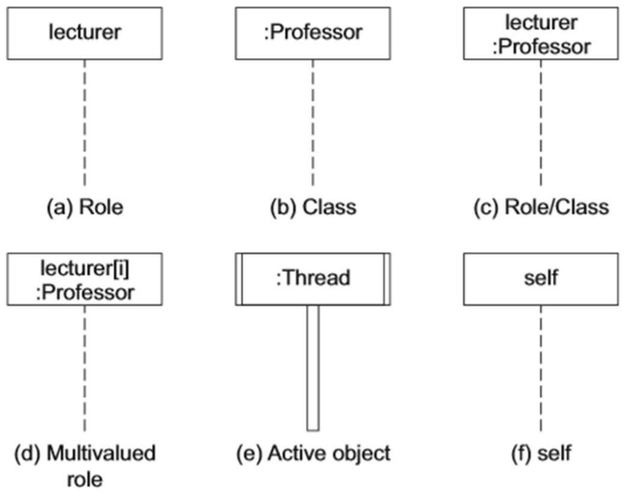


Figure 2.9 Type of lifelines

(Source: UML@Classroom, Seidl, Scholz, Huemer, & Kappel, 2015, p.109)

Each rectangle has their meaning. In general, rectangle plays their important role as an actor or class name. Expression indicates the name of the role and the class of the object is connected with the lifeline. Semicolon on the name inside the rectangle can be omitted if class isn't admitted.

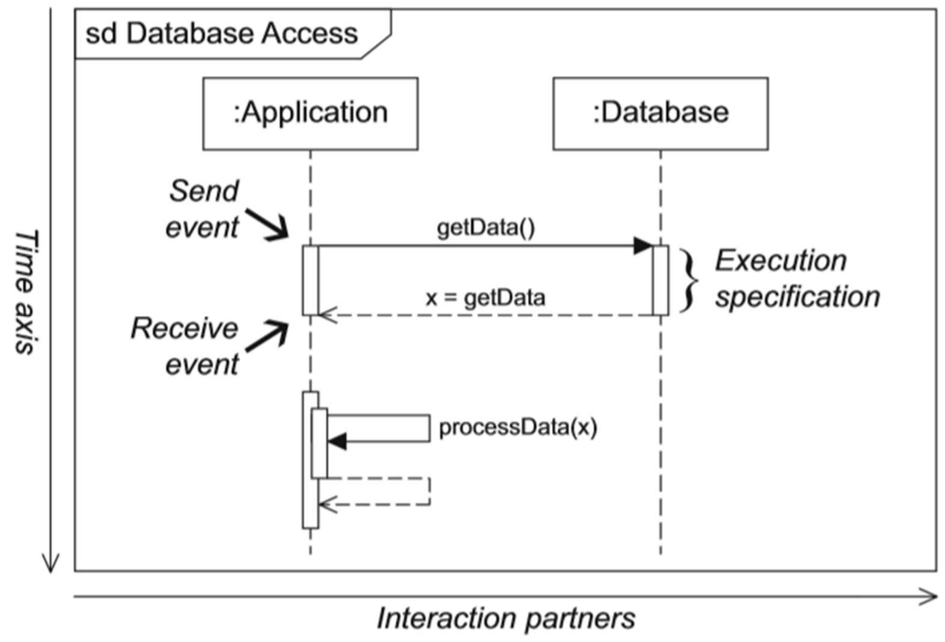
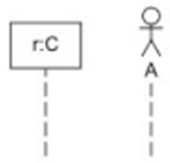
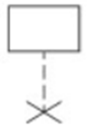
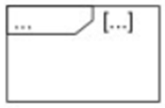
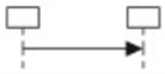
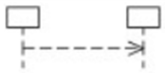
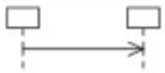
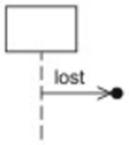
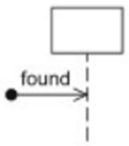


Figure 2.10 Structure of a sequence diagram

(Source: UML@Classroom, Seidl, Scholz, Huemer, & Kappel, 2015, p.110)

Sequence diagram database access has 2 class or role: Application and Database. Application and Database have their lifeline with object activation. Active object is occurred when application or database is interacting each other, whether it send an event or receive an event.

Table 2.7 Notation elements for the sequence diagram

Name	*Notation*	*Description*
Lifeline		Interaction partners involved in the communication
Destruction event		Time at which an interaction partner ceases to exist
Combined fragment		Control constructs
Synchronous message		Sender waits for a response message
Response message		Response to a synchronous message
Asynchronous mes-sage		Sender continues its own work after sending the asynchronous message
Lost message		Message to an unknown receiver
Found message		Message from an unknown sender

(Source: UML@Classroom, Seidl, Scholz, Huemer, & Kappel, 2015, p.140)

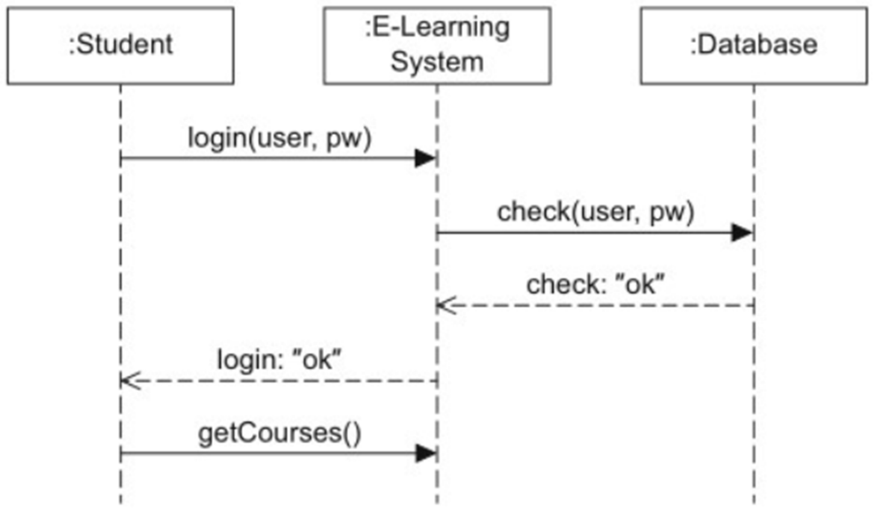


Figure 2.11 Sequence diagram

(Source: UML@Classroom, Seidl, Scholz, Huemer, & Kappel, 2015, p.137)

This is an example of login process sequence diagram. There are 3 interaction partners where involved in the interaction: student, E-learning system, and the database. When student want to login into the system, student send a massage or input their credential into the E-learning system. The system will check the credential by querying database. If the data exist it will response verified access to the system and giving a response to user if their credential is verified by system. Student will be able to do into the next step. In this case, the sequence diagram isn’t considered when student input the wrong credential.

2.7 Eight Golden Rules

Eight Golden rules is a standard rule that created by Shneiderman to describe about planning or analysis great interactive design of software program. According Shneiderman, A system that has interactive design could help productivity of user, frustation free UI, and user-friendly. Which is, these components help user to understand better about the software an knowing more deeply about it. These are eight golden rules:

1. Strive for consistency

   Consistency is the important part for user to know each page or several group of pages that don’t make user confuse to use. For example, the display, color choices, font usage, location of attributes that connecting page to page.

2. Cater to universal usability

   Interface designer has to know about all of user who use this program, calculating all of the possibility who will be use this program from background, age, beginner, expert, and others. After knowing all of that aspect, interface designer has ability to develop better design, for example give a multiple choice of language inside the program, giving shortcut for people who expert in this software field.

3. Offer informative feedback

   Informative feedback happens when a user interacting with the program, and the program is responding to user action. Respond from the program could be a message such as: success, failed, warning, etc. But, in several cases, responds also could be as interchange of page. These responds give user ability to know when his/her action is accepted by program.

4. Design dialogs to yield closure

   Creating step-by-step action to achieve the target. Similar to offer informative feedback but it gives an ending target that user has accomplished. Grouping beginning action, mid process, and ending action has been organized easily to be aware by user.

5. Prevent error

   Design all possibility of program that won’t meet error, bug, or crash. Error could be disaster for a program, because it doesn’t work well what it should. Also, when user which either beginner or expert doing something wrong in the process to program, designer has to know how to prevent error by giving a clue to user for example. Email field for example, it should contain “@” symbol or has to be confirmed, if user doesn’t put the correct format or account, program has to respond that user has made a mistake.

6. Permit easy reversal of actions

   Reversal of action could be described as when user want to cancel to do something, program has ability to back to previous action, similar to “undo” function. For example, user create new transaction of buying product, at several time, user doesn’t want to buy and want to cancel the transaction. So, program has to have ability to cancel the transaction.

7. Support internal locus of control

   A system that has been created which has ability to customize by user. User has access to control anything in that program. So, user experience valuation could be increased based on customizable program.

8. Reduce short-term memory load

   Giving user to use program easier when user giving input to system. So, input from user could be stored in program, and also give a menu that’s not too complex, because it will give user confusion to use the program.

2.8 Five Measurable Human Factors

There’s intercommunication between user characteristic with the interface design. Five measurable human factors could be introduced as early stage of design interface because it helps to prevent design or system failure. Five measurable human factors consist of:

1. Time to learn

   Interface design that could be immediately known by user. How long user learn to use and doing some action to do 1 main task.

2. Speed of performance

   Describe how long it takes to do set of tasks.

3. Rate of user error

   Measure how many or which type error that user frequently do a mistake. A good system will have low rate of user error, conclude that user could interact with program easily understand.

4. Retention of knowledge over time

   Describing how well user remembering in several time frame, from day-to-day, week-to-week, even month-to-month. Retention depends on how many time users spend to use the program and frequency of user open that program and using it. When user often to use it, it’ll be easier to use the program.

5. Subjective satisfaction

   Defining how user likes using the system. This aspect could be conducted as interview to user to know how user satisfaction of the system can be scale of satisfaction or free form (user describe how their feel to use this system).

2.9 Object Oriented Programming (OOP)

Object Oriented Programming (OOP) is designed for focusing on object interact with another object and share information and communicate each other. OOP is different with procedural programming, which procedural programming is focus on writing logic step-by-step flow of execution. Set of data on OOP is used while developer want to manipulating it or usage they want to use, logic statement of OOP isn't used for manipulating the data.

There're several ways to defined Object Oriented Programming. OOP is focus on improving analytics on object, the way to program on OOP is look out everything as an object. In contrast, procedural is focus on step-by-step execution. OOP is a newer technology to help implement feature that hard to develop on procedural programming. It also flexible to be outlined and a modification on an object doesn't give huge impact to another.

Some benefits to use Object Oriented Programming (OOP) rather than Procedural Oriented Programming (POP):

- OOP is more realistic because of developer could work and imagine all of real problem defined into an object.
- Decreasing number of redundant code and increasing extensibility for class.
- Understanding software is easier rather than POP.
- Data hiding concept of OOP help programmer to develop secure programmable code which can't access or altered by another parts of program.
- Interaction or communication between object is easy rather than POP

Object Oriented Programming provide great design pattern to develop with. There are 4 concepts of OOP based on Urdhwareshe:

- Object

  To develop a program which could be web or desktop programming, we need to code properly. In OOP, code properly means everything is seen as an object. Object contains private data that can be accessed by authorized method or operations.
- Class

  Definitions and methods are presented on a Class. Class consists of distinct object with attributes that can be shared of its information and certain function which user could defined their method.

- Inheritance

  Derived base class attributes and method to subclasses. Reusability is powerful on inheritance because reusing base class attribute which is is great idea to reduce logic error even more reducing execution time. Also, it helps to reduce program complexity.
- Polymorphism

  Polymorphism gives ability to programmer to ask the same operation, variable, or object to different actual things by defining different type forms.

2.10 Web Application

Web application is a web that easily accessed by internet protocol and opened with web browser. Web application has a content such as: text information, color, picture, furthermore videos (Reina, 2010, p. 682). Amazon online store is one of the examples of web application, user can do anything, such as searching products, comparing price, also buy the products online.

2.11 RESTful

Representational State Transfer (REST) is an action to create web service. It based on web architecture (Flanders, 2009, p. 5). REST using HTTP protocol to communicate with data. REST client will access to REST server resources which the resources will be differentiate with Universal Resource Identifiers (URIs). There's several type of REST result such as: text, XML, or JSON. REST usually used to interexchange data between system, and its database.

HTTP protocol that is used on REST API:

- GET

  Reading the resources from REST server
- POST

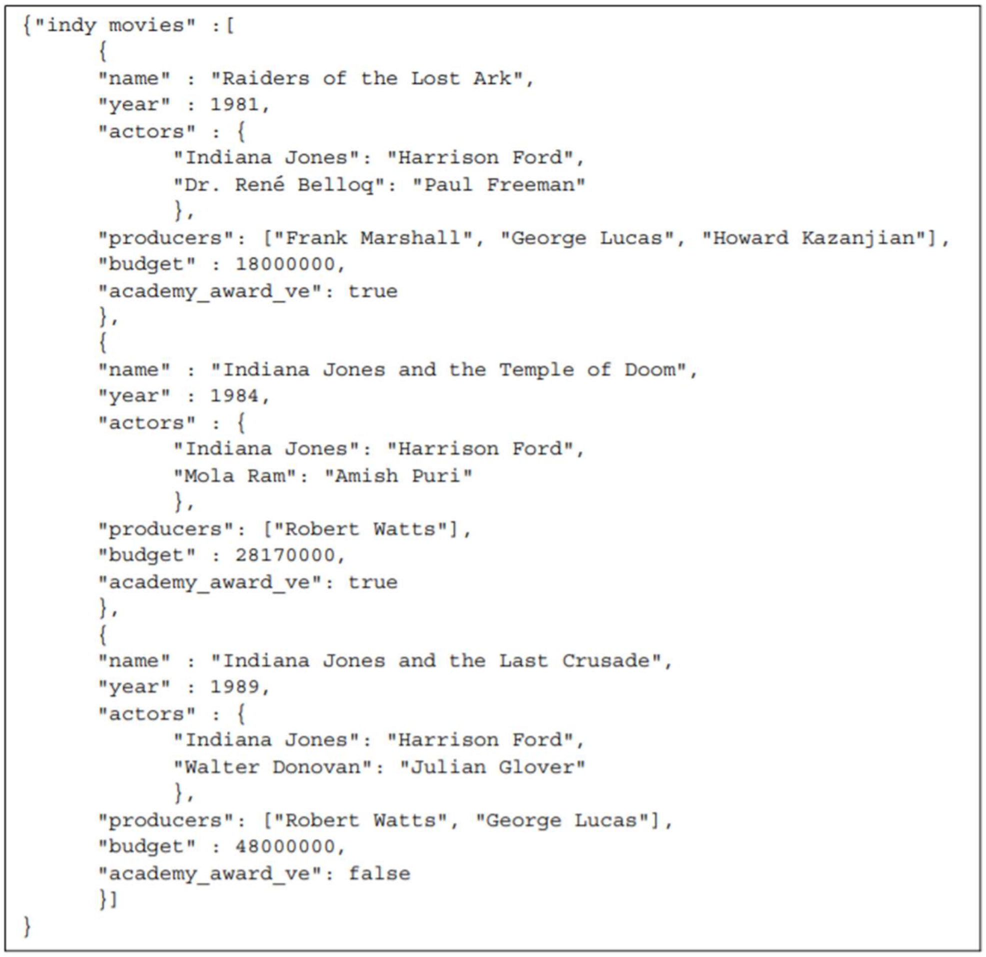
  Creating new data into REST server
- PUT

  Updating existing data on REST server
- DELETE

  Deleting existing data on REST server

2.12 JSON Data

JavaScript Object Notation or JSON is a medium for a storage and changeable data which is easily to read by person. JSON has similar functionality like XML to represent the data (Munzert, Rubba, Meißner, & Nyhuis, 2015, p. 68). JSON is an independent language that created based on JavaScript programming language. JSON data could be decomposed with a lot of programming language. JSON cover an object by "{" and "}" symbols.



```
{"indy movies" :[
      {
      "name" : "Raiders of the Lost Ark",
      "year" : 1981,
      "actors" : {
            "Indiana Jones": "Harrison Ford",
            "Dr. René Belloq": "Paul Freeman"
            },
      "producers": ["Frank Marshall", "George Lucas", "Howard Kazanjian"],
      "budget" : 18000000,
      "academy_award_ve": true
      },
      {
      "name" : "Indiana Jones and the Temple of Doom",
      "year" : 1984,
      "actors" : {
            "Indiana Jones": "Harrison Ford",
            "Mola Ram": "Amish Puri"
            },
      "producers": ["Robert Watts"],
      "budget" : 28170000,
      "academy_award_ve": true
      },
      {
      "name" : "Indiana Jones and the Last Crusade",
      "year" : 1989,
      "actors" : {
            "Indiana Jones": "Harrison Ford",
            "Walter Donovan": "Julian Glover"
            },
      "producers": ["Robert Watts", "George Lucas"],
      "budget" : 48000000,
      "academy_award_ve": false
      }]
}
```

Figure 2.12 JSON result

(Source: Designing an MVC Model for Rapid Web Application Development, 2014, p. 69)

2.13 Model-View-Controller (MVC)

MVC is a concept to produce efficient way to developing a program by separating user interface with its controller underlying information. There are 3 different layers which are: model, view, and controller. Maintenance of an application will be implemented faster because each component of layers can be developed and updated separately without giving a huge impact to another.

Model in MVC represent real data or an association of its business process that is used in an application. There are 2 types of Model:

1. Passive model

   A controller manipulates exclusive model, which means controller change or update model and inform to view if model has been changed and has to be refreshed.

2. Active Model

   When a model is changing its state without controller. It happens when the other object change data and view will be changed too.

View in MVC is getting data from model and represent the model as user interface. View is an output or result that will be understandable by user to operate with.Controller in MVC represent getting and translating input and request on model or view. Controller has responsibility to call method that will change the model and will be presented into view.

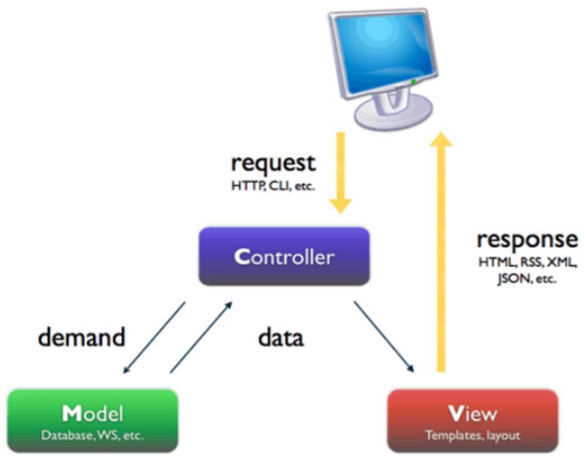


Figure 2.13 Model-view-controller scheme

(Source: Designing an MVC Model for Rapid Web Application Development, 2014, p. 1174)

Model View Controller easily implemented as Figure 2.13. Firstly, requesting HTTP from controller. Controller as a bridge between model and view, it asking a model to know the database. When database or model achieved, controller is working on it and displaying as a view which view could be seen by user.

2.14 Android

Android is operating system for mobile which developed by Google. Android was released to public as commercial on September 23, 2018. This is the first version of android is Android 1.0. Android operating system is based on Linux kernel and the other open source software for touchscreen devices. Android is developed written in Android software development kit (SDK) which is programming in java programming language in common and combined with C/C++. In 2017, Google announced support Android development with Kotlin programming language.

Briefly, Android is developed by 4 people: Andy Rubin, Rich Miner, Nick Sears, and Chris White in California. Andy Rubin is one who thinks to start developing the OS for digital cameras. Likely, they think of future which is of potential developing on smarter mobile devices. Difficulty facing their teams to develop Android until in July 2005, Google acquired Android Inc. $50 million. So, they are joined Google as part of acquisition and make Android become greater. Android was introduced first time on an HTC-made T-Mobile device. Which that is a stepping stone for Android to develop significantly faster based on their version.

2.14.1 Android Version

- Android 1.0

  This version is the first version which released on September 23,2008 introduced Android market (Google Play Store), a widget on home screen, and notifications, etc.
- Android 1.5 (Cupcakes)

  This version has several new features such as: upload photo to Picasa, upload videos to YouTube, support third-party virtual keyboard with text prediction, etc.

- Android 1.6 (Donut)

  The big update of this version is: quick search box that allows user to search easily than ever, support for CDMA network, adjusting with display sizes on Android devices.
- Android 2.0 (Eclair)

  Introducing Google Maps navigation, HTML5 browser, lock screen, etc.
- Android 2.2 (Froyo)

  Introducing USB tethering and Wi-Fi hotspot functionality, PIN lock screen, etc.
- Android 2.3(Gingerbread)

  Introducing support for front-camera, enhanced text input on virtual keyboard, download manager which gives ability to user to access downloaded file easily.
- Android 3.0 (Honeycomb)

  Honeycomb focus on tablet user. Support multi-core processors, optimized holographic user interface for tablet, allow to encrypt all user data.
- Android 4.0 (Ice Cream Sandwich)

  Updating for virtual button and refined the interface. Also adding small feature as data usage analysis, face unlock, sharing content with NFC.
- Android 4.1 (Jelly Bean)

  Introducing Google Now which give information as personal assistance, actionable or expandable notification, and account switching (multiple user on 1 device).
- Android 4.4 (KitKat)

  Introducing Voice: Ok Google which is to do specific action based on what user said. Improving the design based on what user do or wanted.
- Android 5.0 (Lollipop)

  Introducing material design, which makes user to navigate their device easily. Ability to do multiscreen on different devices and allow user to see the notification on lock screen.
- Android 6.0 (Marshmallow)

Updating personal assistance without leaving what are user doing right now, give user permission to share with application, and optimize battery life.

- Android 7.0 (Nougat)

  Multi-windows support and floating window, updating notification that will be presented from screen to screen
- Android 8.0 (Oreo)

  Introducing a modular architecture for hardware makers to adapt easier and faster with update version. Giving ability to user to do multi-tasking features.
- Android 9.0 (Pie)

  Replacing single pill for 3 button setups on bottom screen of Android. Redesign and refined feature of Android itself.

2.14.2 Android Architecture

Android operating system is divided into 5 section with 4 main layers.

- Linux Kernel

  This layer contains all important aspect of hardware and handle networking very well on device drivers.
- Libraries

  Encompasses java-based library that to help develop a mobile apps completely work.

  On this layer, there’s Android Runtime which Java Virtual Machine to optimized Android, so help developer to write application on java programming language.
- Application Framework

  Provide higher level service from java classes. Giving ability to developer to use the service of this application.
- Application

  A product that has been developed and installed.

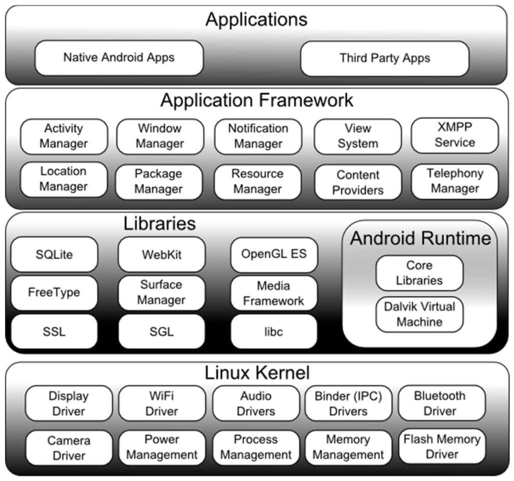


Figure 2.14 Android operating system architecture

(Source: Android Developers)

2.14.3 Android Activity Lifecycle

Activity in Android is a fundamental part platform on application model. Activity represents a single screen in Android. This activity will handle of User Interface components or widgets. An activity will go through several number of states.

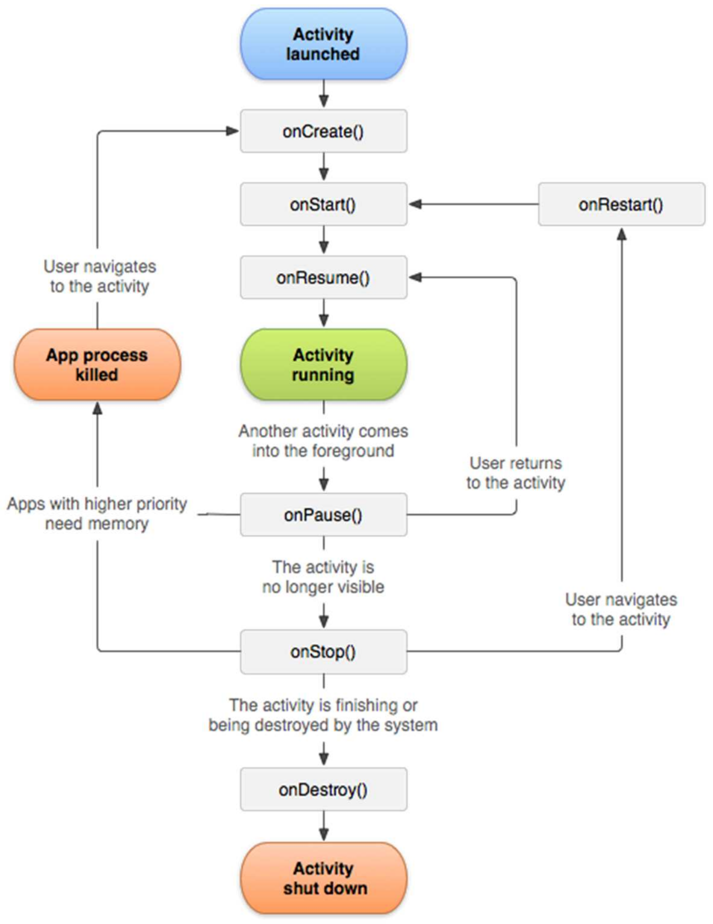

Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Figure 2.15 Android activity lifecycle

(Source: Android Developers)

onCreate() method is called first time when activity was launched. It will handle creating views and binding data.

onStart() method is called after onCreate() method or after an activity was restarted. This state, activity will be show into foreground and visible to user.

onResume() method is called when user interacting with current activity. Current activity will be the top of stack activity, and receive all user input.

onPause() method is called when the activity isn’t main focus anymore. The activity isn’t the top stack of another activities. It happens when the activity isn’t visible to user, or partially visible but isn’t the main focus.

onStop() method is called after onPause() method which the activity isn’t longer visible to user. It may happen when the activity is being destroyed, existing activity entering resumed state, or new activity is starting.

onRestart() method is called when onStop() method isn’t going to be destroy but prior to start state. It happens when user want to navigate to the activity.

onDestroy() method is the final state of activity lifecycle. It will be called if the activity is finish or being destroy by the syst