Chapter - 2

Iterative And Incremental Development:

2.1 Phases (inception, elaboration, construction and transation)

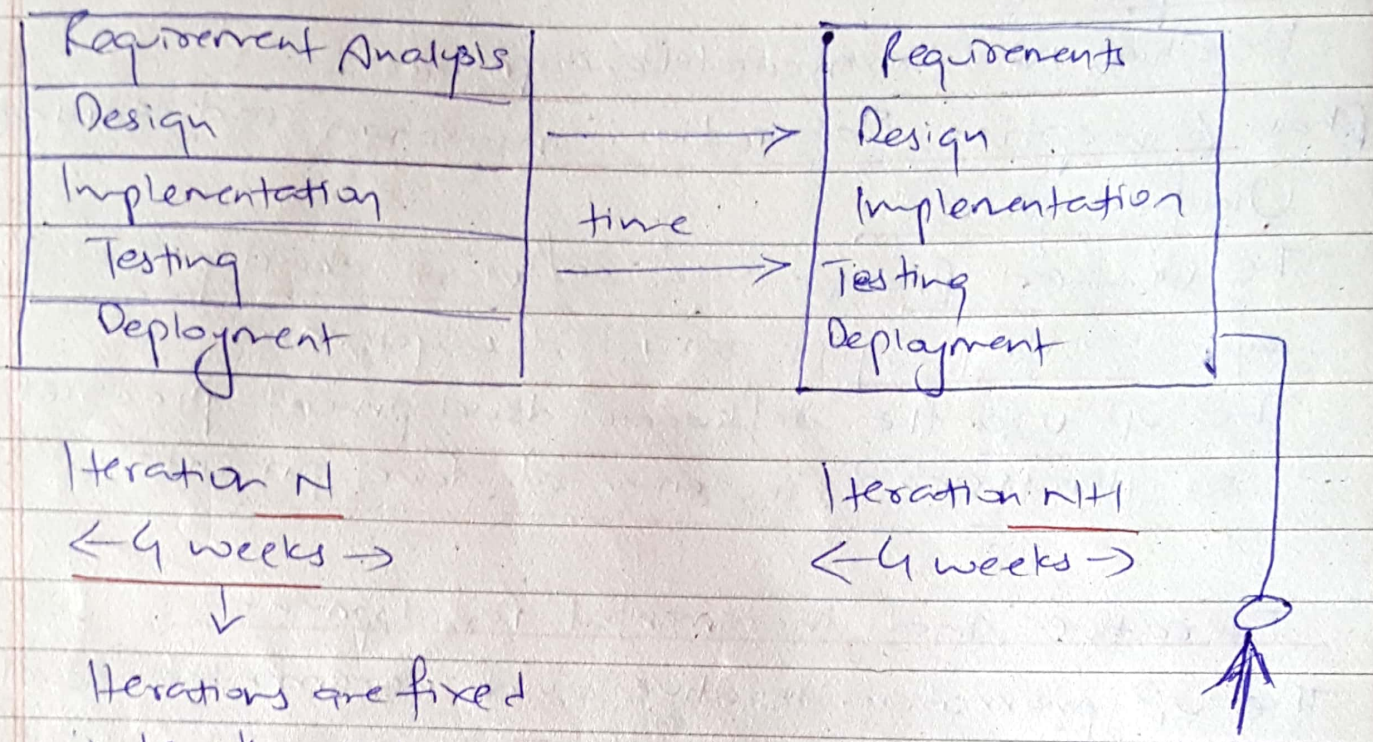
Unified Process:

- The Unified Process is a software development process for building object-oriented system.
- The up uses the software development practices such as iterative and incremental development

Iterative And Incremental Development:

- The up promotes iterative and incremental development approach.
- -> In this approach, development is organized into a series of short-fixed length miniprojects called iterations.
- -> The outcomes of each is a tested, integrated and executable system.
- -> Each iteration includes its own requirement analysis design, implementation, and testing activities.
- -> The iterative lifecycle is based on the successive enlargement and refinement of a system through multiple iterations with cyclic feedback and adaptation as a core drivers to converge upon a suitable system.
- -> The system grows incrementally over time, iteration by iteration and thus this approach is also known as iterative and incremental development.

fig: Iterative and Incremental Development



feedback from iteration N leads to refinement and adaptation of the requirements and design in iteration N+1

The phases of Unified process are as follows:

1) Inception
2) Elaboration
3) Construction
4) Transition

1) Inception:

* The vision of the end product is developed.
* The business case for the project is developed.
* The outline for the most crucial subsystem is designed.

* The most important risky are identified and prioritized
* The elaboration process is planned in detail and the whole project is roughly estimated

2. Elaboration:
* Most of the project use cases are identified in detail and the system architecture is designed.
* The relationship between the architecture and the system itself is developed
* The architecture is expressed as views of all the models of the system, which together represent the whole system. The models include architectural views of the use-case model, the analysis model, the design model, the implementation model and deployment model.
* The most critical use cases are identified and realized
* At the end of elaboration phase, the project manager is in a position to plan the activities and estimate the resources required.

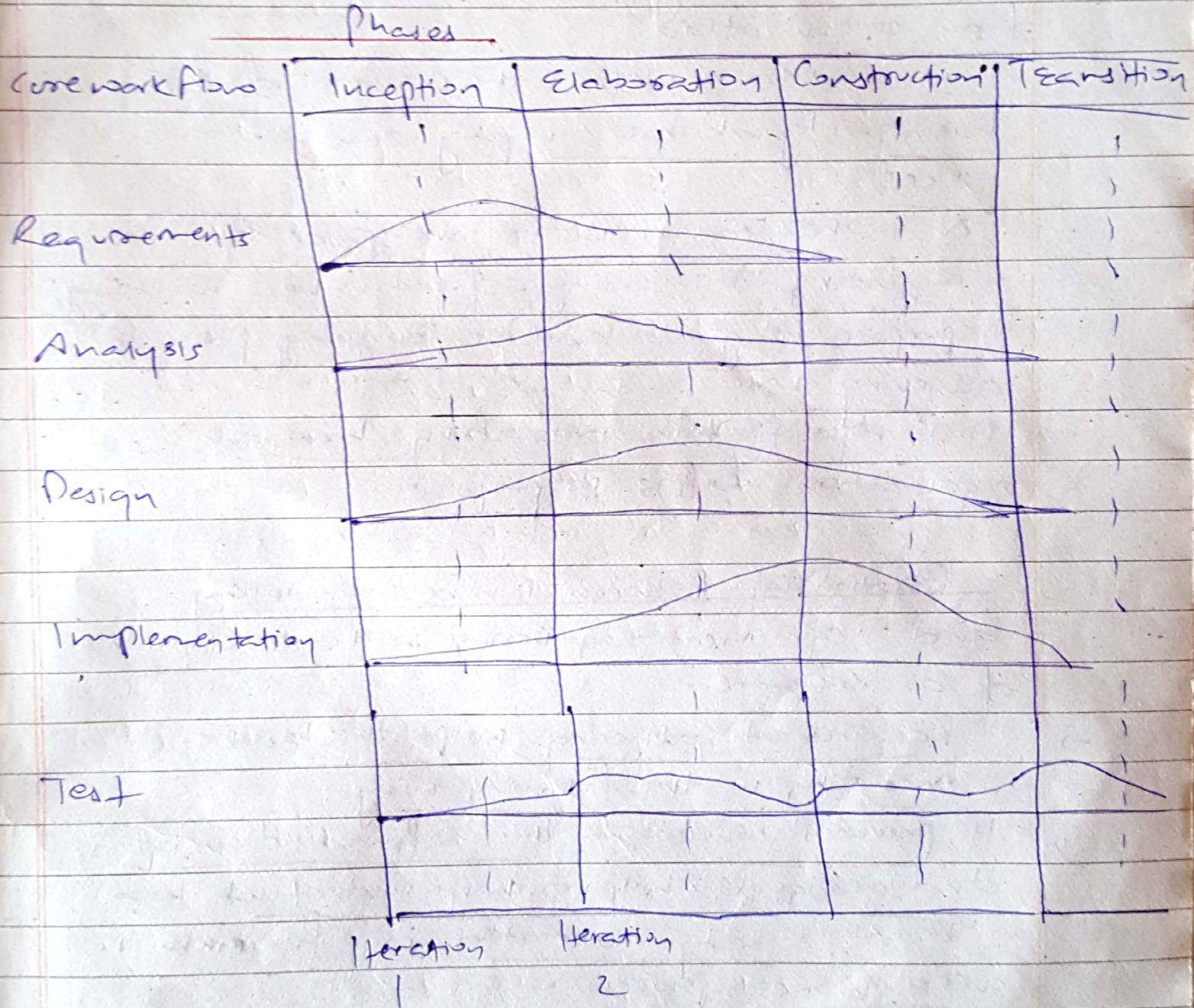
3) Construction:
→ The architecture baseline grows into a system.
→ Here, the project is builded and made ready for transfer to the user
→ Here, the developers may discover the better ways of structuring the system. So, they may suggest minor architectural changes to the architects.

At the end of this phase
The product contains all the use cases that management and customer agreed to develop for this release.
It may not be entirely free of defects, however here, the milestone is
Does the product meet users needs for early design?

Transition phase:
In transition phase product moves into beta release.
In the beta release, a small number of experienced users tries the product and reports defects.
Developers then correct the reported problems and incorporate some of the suggested improvements into a general for the larger user community.
This phase includes the activities such as manufacturing training customer personnel, providing help assistance and ~~corresponding~~ correcting defects found after delivery.

The maintenance team often divides these defects into two categories:
1) those with sufficient effect on operations to justify an immediate data release.

8, those that can be corrected in the next regular release



2.2.1 Business Modeling:

- A business modeling is a technique for understanding the business processes of an organization.
- A business model is the method of doing business by which a company can sustain itself.
- A business model is the conceptual structure supporting the ability of a business, including its purpose, its goals and its ongoing plans for achieving them
- It is specification describing how an organization fulfills its purpose.

Relationship between Business modeling and requirement raqathering in the development of a software:

→ Requirement gathering helps the business modeling in the following ways:

i) It provides context to the software developers and designers, helping them understand how the steps being automated, will fit into the overall business process.

ii) It is the basis for validation that each step adds value, even when there is no time to complete a long reengineering effort.

- के को आधार मा analyze garne [illegible]
- tools for validation - req ko adhar ma valid garne parxa [illegible]
- requirements make people understand by giving verbal list or visual model
- first layer of traceability.

3, People understand things in different ~~[illegible]~~ formats, so while some of the audience will best understand verbal lists of requirements, others will respond to the visual depiction of a process model. So requirement should be properly gathered.

4, It is the first layer of traceability for the software development lifecycle.

In this way, requirements gathering provides extreme support to the business model that will directly affect on the success of business. So, the relationship between Business modeling and requirement gathering in the development of a software is vast.

How to develop a business Model?

1, The business modelers should prepare a business usecase model. ~~[illegible]~~ The business usecase model identifies the 'actor' to the business and the "business usecases" that the actors use.

2, The modelers should develop a business object model. The business model consists of workers, business entities and work units that together realize the business use cases.

3 b) Models development is driven by use case. Explain with suitable ~~example~~ diagram.

⇒ Models development is driven by use case. The analysis model grows incrementally as more and more use cases are analyzed. For each iteration, we select a set of use cases that we realize in the analysis model. we build the system as a structure of ~~ana~~ analysis classes and relationships between those classes. we also describe the collaborations that realize the use-case, i.e usecase realizations. Then, in the next iteration - we identify and select the most important sets of use cases for the first iterations, to build a stable architecture early in the system life cycle.

During analysis and design, we transform the use-case model via an analysis model into a design model. i.e to a structure of analysis classes and use-case realizations. The goal is to realize the use cases cost-effectively so that the system offers suitable performance and can evolve in the future. we create use-case realization which describe how the use case is performed in terms of interacting object in the design model. This models describes, the different parts of implemented system, and how the parts should interact to perform the use cases.

Model
View
controller

In this way, use case helps to make analysis model that later is transformed into design model and hence a complete system model is formed. In this ~~way~~ way we can conclude Model development is driven by use-case.

Example: Dice-game:

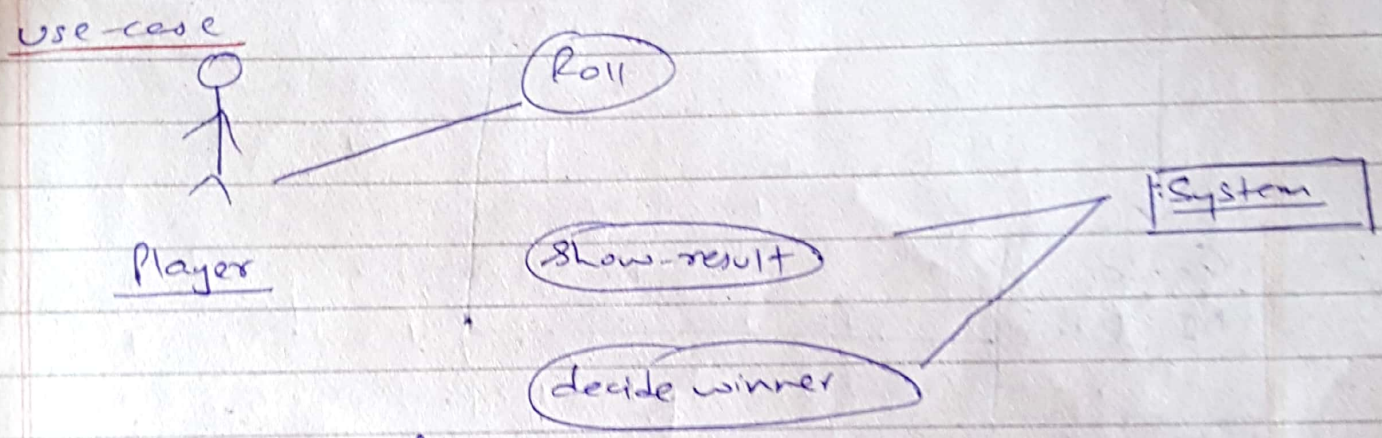
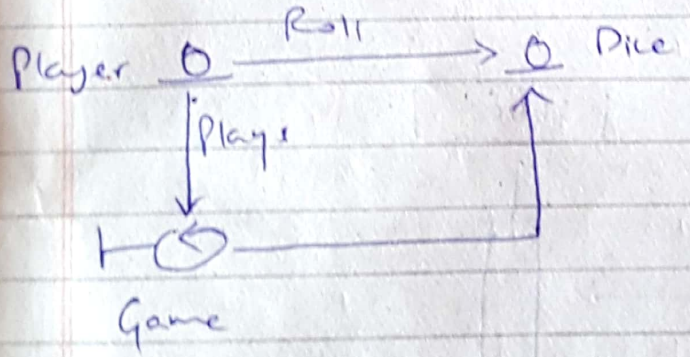
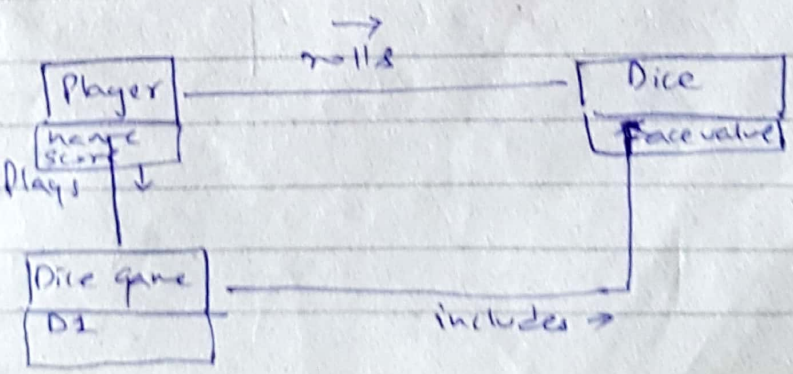


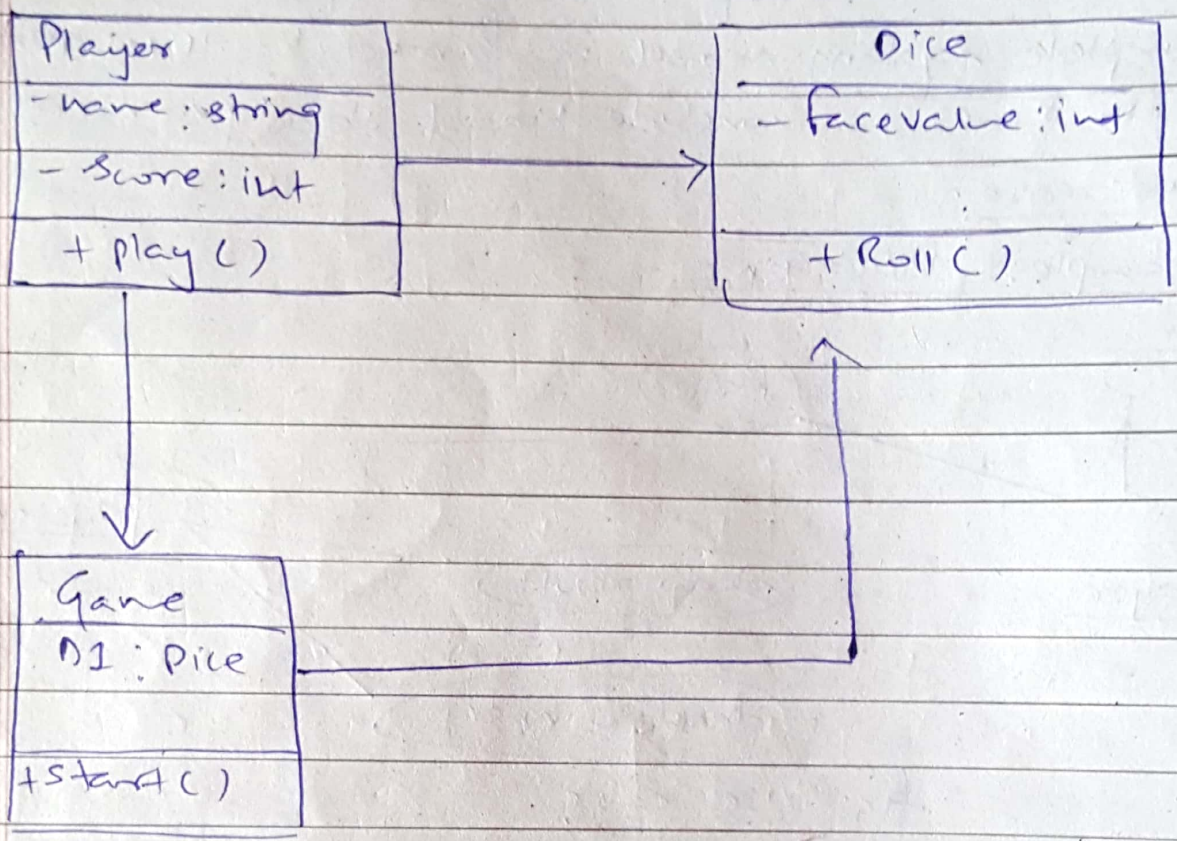
fig: use-case



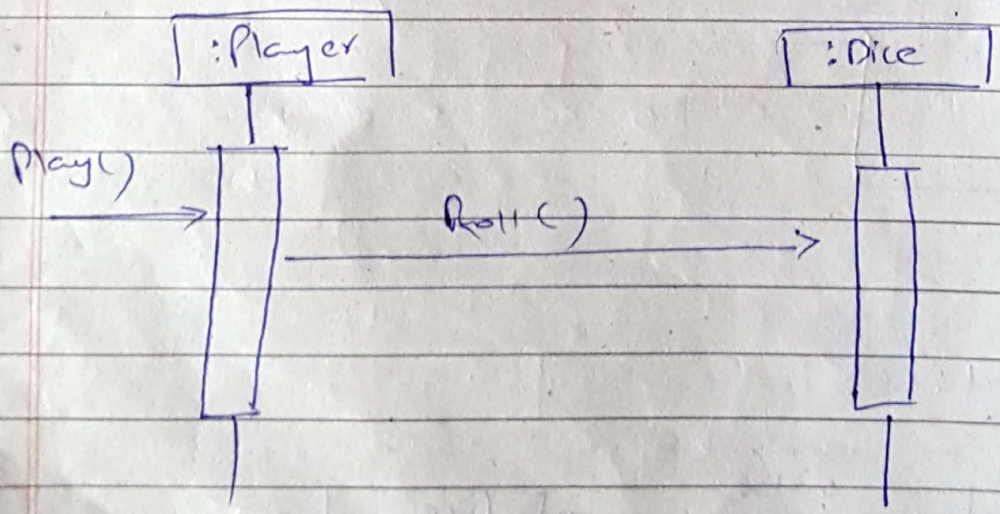
Domain model:



Class Diagram



Interaction Diagram:



Phases of Unified Process:

① Inception ② Elaboration ③ Construction ④ Transition

① Inception:

- The life cycle objectives of project are stated; so needs of every stakeholder are considered.
- Scope and boundry conditions, acceptance criteria and some requirements are established

Activities

1) Formulate the scope of project
   Here, Needs of every stakeholder, scope boundry condition and acceptance criteria are established
2) Plan & prepare the business case
   Define risk mitigation strategy, develop an intial project plan and identify known cost schedule & profitability trade-offs.
3) Synthesize candidate architecture:
   Candidate architecture is picked from various potential architectures.
4) Prepare the project environment

Exit criteria:

→ An initial business case containing at least a clear formulation of product vision - the core requirements interms of functionality, scope, performance, capacity, technology etc.
→ Success criteria
→ An initial risk assessment (important risk are identified
→ Estimation of resources required to complete elaboration phase.

Elaboration Phase:

→ Most critical among 4 phases
→ At the end of this phase, hard engineering is considered complete and project undergoes the decision whether or not to commit to construction & transition phase.
→ Plan approved by project manager and funding authority & resources required for this phase have been allocated.

Entry criteria

→ Define the architecture
Project plan is defined. The process, infrastructure and development environment are described.

→ Validate the architecture
→ Baseline the architecture
To provide a stable basis for the design and implementation effort in the construction phase

Exit criteria:

- A detailed software development plan with an updated risk assessment, a management plan, a staffing plan, a phase plan showing the number and contents of the iteration. Iteration plan and a test plan
→ The development environment and other tools

→ A baseline vision. in the form of a set of evaluation criteria for final product
- A domain analysis model, sufficient to be able to call the corresponding architure 'complete'
- An executable architecture baseline.

Construction:
- Manufacturing process
- Managing resources, controlling operations to optimize cost, schedules and quality.
- This phase is broken into different iterations.

Entry criteria
- Products and Artifacts of previous iteration.

Iteration plan must state iterations specific goals
- Risks being mitigated during iteration
- Defects being fixed during iteration.

Activities
① Develop & Test components
Components required satisfying usecases, scenarios & other functionality for iteration are built. Unit & integration are done on components.
② Manage resources and control process
③ Assess the iteration
Satisfaction of the goal of iteration is determined.

Exit criteria

- The same products and artifacts updated.
- Release description document, which captures the results of an iteration
- Test cases and results of the test conducted on products.
- An iteration plan, detailing next iteration.
- objective measurable evaluation criteria for assessing the results of next iteration.

(4) Transition:

Entry criteria

- Transition the s/w product to the user community
- This phase is entered when baseline is sufficiently mature enough to be put into hands of users.

Activities

1. Test the product deliverable in customer's environment.
2. Fine tune the product based upon customer feedback.
3. Deliver the final product to the end users.
4. Finalize end-user material.

Exit criteria.

-> Update of some of the previous documents as necessary. The plan being replaced by "post-mortem".
Analysis of performance of project relative to its original & revised success criteria.

-> brief inventory of organizations new assets as a result of this cycle.

Workflow:

- Sequence of activities that produces a result of observable value
- In UML, WF can be expressed as sequence diagram, a collaboration or an activity diagram.

Process of making workflow:

1. Identify the different kinds of workers that participate in the process
2. Identify the artifacts that we need during the process for each type of worker.
3. Describe how the process flows through the different workers and how they create, produce and use each other's artifacts.

There are 9 workflows in UP/RUP.
6 — engineering workflow
3 — supporting workflows.

"Engineering workflows"	"Supporting workflows"
- Business Modeling	- Project Management
- Requirement Analysis	- configuration & change Management
- Design	- Environment
- Implementation	
- Test	
- Deployment	

Business Modeling:

- There is a problem with business engineering as software engineers and business engineers don't communicate with each other.
- This leads to the condition where the output from business engineering is not properly used as the input to software development effort and viceversa.
- UP address this by providing a common language and process for both community and showing how to create and maintain direct traceability between business & software models.

In business modeling, we document business process by using business use cases. This assures a common understanding among all stakeholders of what business process need to be supported in the organization.

Many project chose not to do business modeling.

(2) Requirements:

Goal → describe WHAT the system should do and allow the developers and the customers to agree on that description.

→ To achieve this, we elicit, organize and document required functionality and constraints; track and document tradeoffs and decisions.

→ A vision document is created and stakeholder needs are elicited (draw/extract)

→ Actors are identified, representing the users and any other system that may interact with system being developed.

→ use cases are identified representing the behaviour of the system. because use cases are developed according to actors needs. The system is more likely to be relevant to others.

eg:

→ use case model is used during requirement capture, analysis and design and test.

(3) Analysis and Design:

→ goal is to show how the system will be realized in the implementation phase.

→ result in design model and an analysis model.

→ design model serves as an abstraction of source code i.e design model acts as a blueprint of how source code is structured and written.

→ Design model consists of design class structured into design package and design subsystem with well defined interfaces.
→ Contains descriptions of how objects of these design classes collaborate to perform use cases.
→ The design activities are centered around the notation of architecture.
→ The production and validation of this architecture is the main flows of early design iterations.
→ Architectural views are abstractions or simplifications of the entire design.

Implementation:

→ To define organization of code in terms of implementa subsystems organized in layers.
→ To implement classes and object in terms of compon
→ To test the developed component as units.
→ To integrate the results produced by individual implementers into an executable system
→ It describes how you reuse existing components; implement new components with well-defined responsibility making possibility of reuse
eg: subsystems can be created as directory, folder in file system or packages.

Test:

→ Purposes:

1) verify the interaction between objects.
2) verify the proper integration of all components of software.
3) verify all requirements have been correctly implemented
4) To identify/ensures defects are addressed before the ~~deployment~~ deployment of software.
5) UP is an iterative approach; which means you test throughout the project.
6) allows us to find the defects as early as possible which reduce the cost of fixing the defects.
7) Tests are carried out along three quality dimensions: reliability, functionality, application performance and system performance

Deployment:

- covers a wide range of activities including:
  - producing external releases of the software.
  - Packaging the software.
  - Distributing the software.
  - Installing the software.
  - planning and ~~conduct~~ beta test.
  - formal Acceptance.

Supporting workflows:

Project Management

→ Software Project Management : It is the art of balancing competing objectives, managing risks and overcoming constraints to deliver.

→ Our goal in this section is to make the task easier by providing

i) A framework for managing software intensive project.

ii) Practical guideline for planning, staffing, executing and monitoring.

iii) Framework for managing risk.

Configuration And change Management:

→ This workflow provides guidelines for managing multiple variants of evolving software systems.

→ Tracking which version are used in given software builds.

→ Performing builds of individual program or entire releases according to user-defined version, specif

→ So, it is important in iterative process where you may want to be able to do builds as often a daily, something that would become impossi without powerful documentation.

Environment:

→ The purpose of this is to provide software development organization with the software development environment that are needed to support development team.

→ focus on activities to configure the process in context of a project.

→ Step-by-step procedure is provided in describing how you implement process of organization.

→ It contains a development test providing you with guidlines, templates and tools necessary to customize the process.

- Unified Process is a software engineering process, which provides a disciplined approach to assign tasks and responsibilities within a development organization.

- Unified Process is a process product developed by working closely with customers to ensure that process is continuously updated & improved upon to reflect recent experience and are proved to be best practices.

- Unified Process enhances team productivity by providing every team members with easy access to a knowledge base with guidelines and tool for all critical development activities.

- Unified Process activities creates and maintain mo rather than focusing on the production of large am of paper documents: (focus on development and maintenance of model.)

- Unified process is also a guide to effectively use UML. UML allows to communicate requirements, architectures design.

- Unified process is supported by tools which autom large parts of the process.

2.3 Models evolution through the iteration:

1) Requirement Model → Usecase Model
   → Interface Description
   → Problem Domain Model
   (Object, Implication, Problem &

2) Analysis Model → object Domain Model

3) Design Model → class diagram
   → Sequence diagram → Interaction diag
   → collaboration diag

4) Implementation Model → Source code + - -

5) Test model → Test cases
   → Test result
   → Test Project

6) Deployment model → Setup file
   → Environment
   → Data

Comparison of Use-case Model and the Analysis Model

Use-case Model	Analysis model
- Described using the language of the customer.	- Described using language of the developer.
- External view of the system	- Internal view of system
- Structured by use cases; Gives the structure to the external view	- Structured by classes and packages; gives structure to the internal view.
- Used primarily as a contract between the customer and the developers on what the system should and should not do.	- Used primarily by developers to understand how the system should be shaped. i.e designed and implemented
- May contain redundancies, inconsistencies etc among requirements.	- Should not contain redundancies inconsistencies etc among requirements.
- Captures the functionality of the system, including architecturally significant functionality.	- outlines how to realize the the functionalities within the system, including architecturally significant functionality; works as a first cut at design.
- Defines the use cases that are further analyzed in the analysis model	- Define use-case realizations, each one representing the analysis of a use case from the use-case model.

Analysis Model	Design Model
- Conceptual Model, because it is an abstraction of the System and avoids the implementation issues.	- Physical model, because it is a blueprint of the implementation.
- Design-generic (applicable to several designs	- Not generic, but specific for an implementation.
- Three building blocks on classes: <<control>>, <<entity>> <<boundary>>.	- Any number of (physical) building blocks on classes depending on implementation language
- less formal	- more formal
- less expensive to develop	- More expensive to develop
- few layers	- Many layers
- Dynamic (but not much focus on sequence)	- Dynamic (much focus on sequence)
- Outline of the design of the System, including its architecture	- Manifests the design of the system, including the architecture (one of its vie
- Primarily created by "legwork" in workshops.	- Primarily created by "visual programming in round-trip engineer environments.
- May not be maintained throughout the complete Software life cycle.	- Should be maintained throughout the complete Software life cycle
- Defines a structure that is an essential input to Shaping the System - including creating the design model.	- Shapes the system while trying to preserve the struc defined by the analysis mo as much as possible

Implementation Model:

- The implementation model describes how elements in the design model such as design classes are implemented in terms of components such as source code files, executables and so on.
- The implementation model also describes how the components are organized according to the structuring and modularization mechanisms available in the implementation environment and the programming languages in use. Also how the components depend on each other.
- The implementation model defines a hierarchy of implementation subsystem.
- Implementation model includes:
  * Implementation subsystem and their dependencies interfaces and contents.
  * Components, including file and executable components and their dependencies. The components are unit tested
  * The architectural view of the implementation model.

Test Model:

- The test model primarily describes how executable components (such as builds) in the implementation model are tested by integration and system tests.
- The test model can also describe how specific aspects of the system are to be tested.
- The test model is the collection of test cases, test procedures, and test components.

- If test model is large, it may be useful to introduce the packages in the model to manage its size.

++ includes - Test cases:

specify what to test in the system.

- Test procedures:

specify how to perform the test cases.

- Test components:

automate the test procedures.

- Testing also results in a test plan, evaluations of the tests, and defects that can be fed back to the other core workflows, such as design and implementation.

Use-case Driven Process:

* During the requirements workflow, developers can represent requirements as use-cases.
* Project managers then plan the project in terms of the usescases so that developers can work with it
* During analysis and design, developers realize the use case in terms of classes or subsystems.
* Developers then, implement the components.
* components are integrated to realize a set of useca
* finally, testers verify that the system implements the right use cases for the users.
* In other words, use cases bind together all development

Q. What is software quality? Explain its perspectives:

a. In context of software engineering, software quality refers:

i) Software functional quality reflects how well it complies or conforms to a given design, based on functional requirements and specifications.

ii) Software structural quality refers to how it meets non-functional requirements that supports the delivery of the functional requirements such as robustness/ maintainability.

Following are the prerspectives:

1) Transcendental perspective:

Quality is something that can be recognized but not defined. One can say it may be of high quality while others can criticise it. This requires that product developer should keep in touch with their customers to ensure that their specifications are met.

2) Manufacturing perspective:

It refers to the ability to manufacture a product to that specification over and over within accepted tolerances where primary focus of software quality is on the design and development activities.

3) User perspective:

This perspective of quality not only considers the viewpoint of the individuals users but their context as well.

4, Product prespective:

Quality is tied to inherent characteristics of the product like reliability, usability, availability, flexibility, accessibility, maintainability, portability and adaptability.

5, Value-based prespective:

Quality is dependent on the amount a customer is willing to pay for it.

Process engineering:

It focuses on the design, operation, control optimization and intensification of chemical, physical and biological processes.

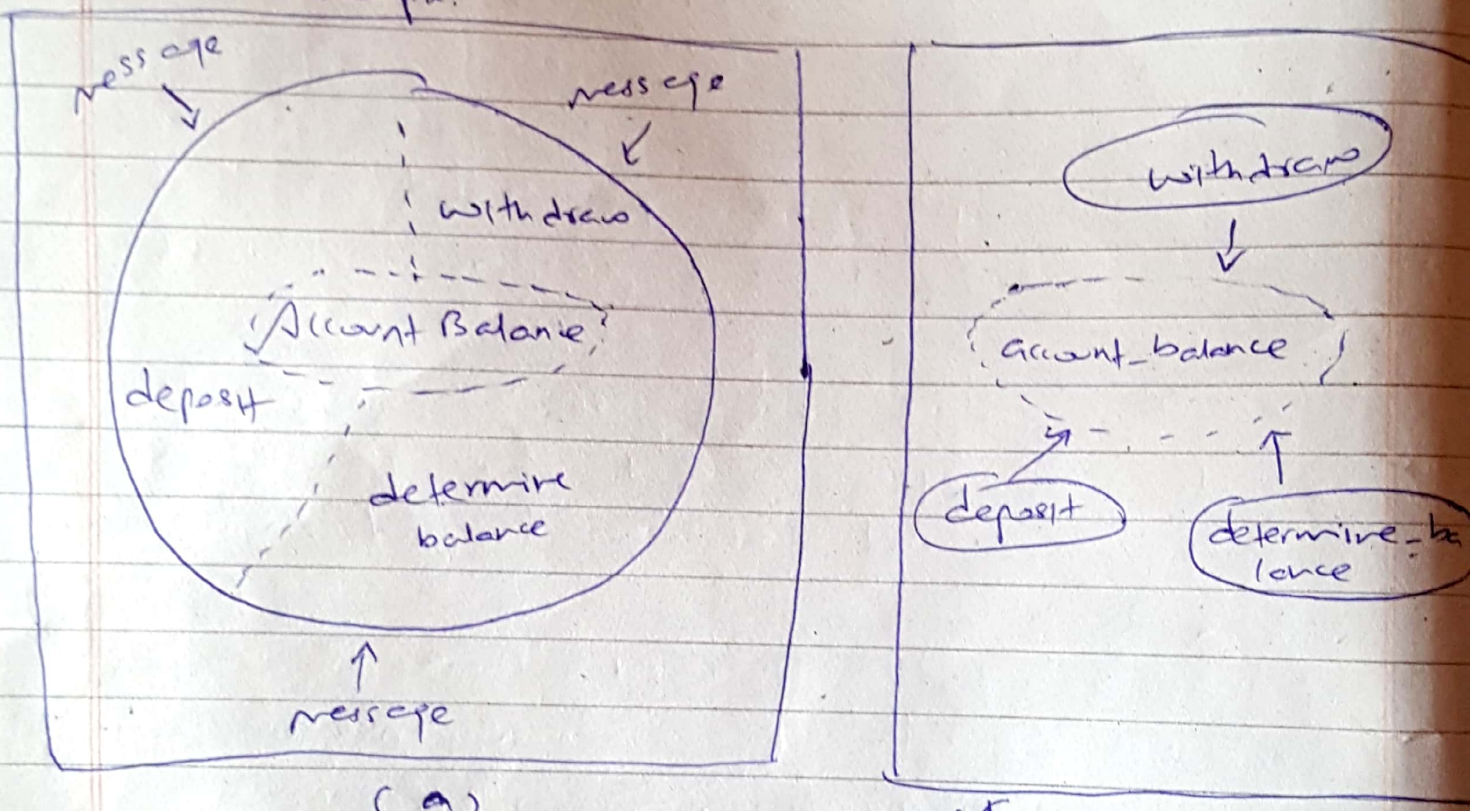
Role of People in iterations:

1, finance the product
2, Schedule
3, develop
4, Manage
5, test
6, use
7, Get benefit from it.

Date
Page

5 Object oriented programming and procedural programming strength and weakness of object-oriented and procedural programming with the help of banking transaction. strength.

example:



(a)
Object-oriented paradigm

(b)
Procedural paradigm

In object orient. view point there will be an object as shown in figure 'a'. The 'attribute' component of the object is the "account balance". The operations that can be performed on that account balance include 'deposit', 'money in the account, 'withdraw', money from the account and 'determine balance'. The "bank account" object combines an "attribute" and three operations performed on that "attribute" into a single artifact.

from procedural viewpoint, a product that deals with banking would have to encorporate an attribute the 'account_balance' and three operations 'deposit', 'withdraw', and 'determine_balance'.

In object oriented approach, object is implemented. Details such as how the attributes of an object are stored are not known from outside the object. It is an implementation of information hiding (Abstraction). This information barrier surrounding the object is denoted by the solid black line in figure 'a'. The dashed line surrounds 'account_Balance' in figure 'b' because all the detail of 'account_balance' are known to the modules in the implementation using the procedural paradigm. And the value of 'account_balance' therefore can be changed by any of them.

The 'deposit' method is within the 'bank account' object and knows how the accountBalance is implemented.

2) During maintenance
In procedural programming, if the way an account_balance is represented is changed (for example, integer to some other type) then every part of that product that has anything to do with an 'account_balance' has to be changed.

In object oriented paradigm, the change need to be made only within the bank account object itself. No other part of the product has knowledge of how an accountBalance is implemented, so no other need to be changed. So, maintainance is quicker and easier.

2. An object has a physical counterpart. example: The bankaccount object in the bank product corresponds to an actual bank account in the bank. This is close correspondence between the objects in a product and their counterparts in the real world should lead to a better-quality software.

3. A product built using procedural programming is implemented as a set of modules, but conceptually it is a single unit. In contrast, when the object oriented paradigm is used correctly, the resulting product consists of number of smaller, largely independent units. So, complexity is reduced

4. OO encourages reuse.

Report

Add New Transaction Type

PP → The changes needed in funcn design is as shown in fig.
↳ tyo transn module मा locally add गर्नु पर्छ which also cause need of modfn मा Transaction module

OO → we would add new opm on every object type.

<u>Conclusion:-</u> data item change गर्न OO sajilo हुन्छ,
process मा change गर्न PO " "

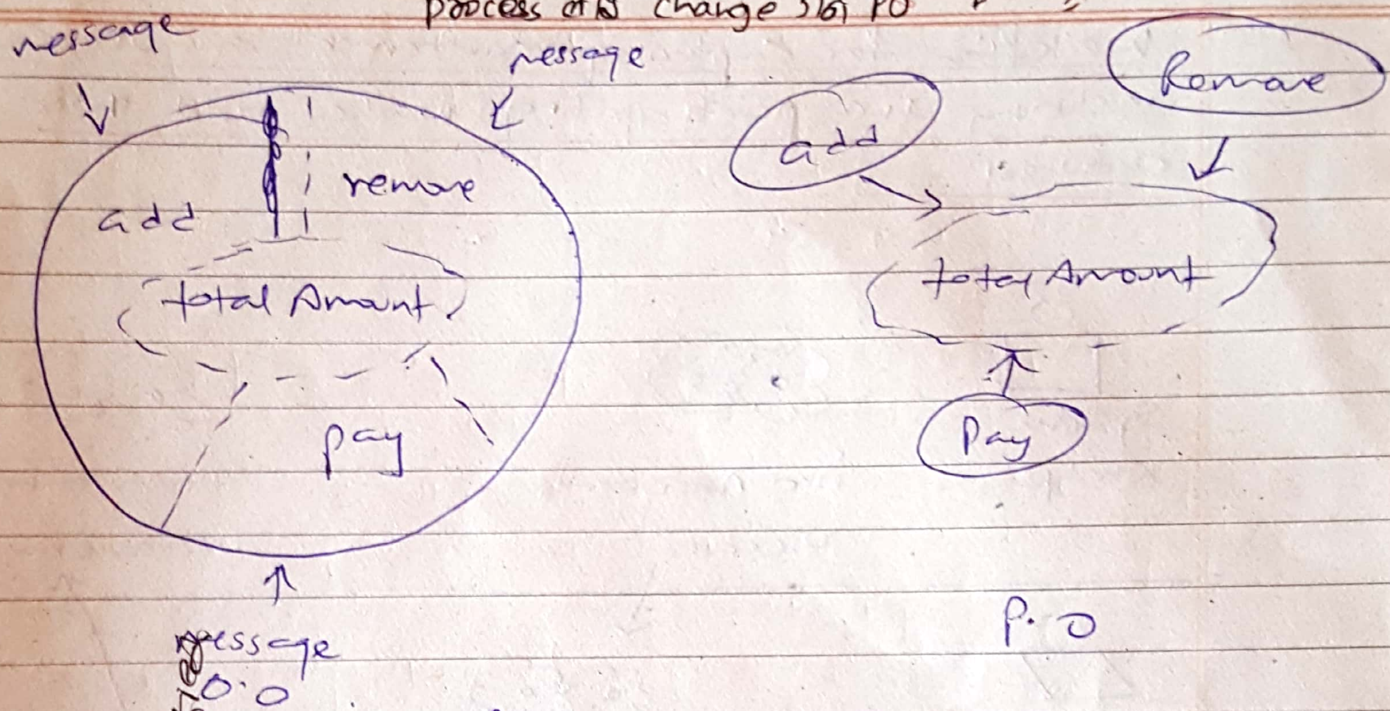


fig: Sale line Item.

Account: passbook saving, bonus saving.

Transaction processing module: Open Deposit withdraw.

Account type: → Account
↑ subclass ↑ ~~---~~ superior class
operations
inherit

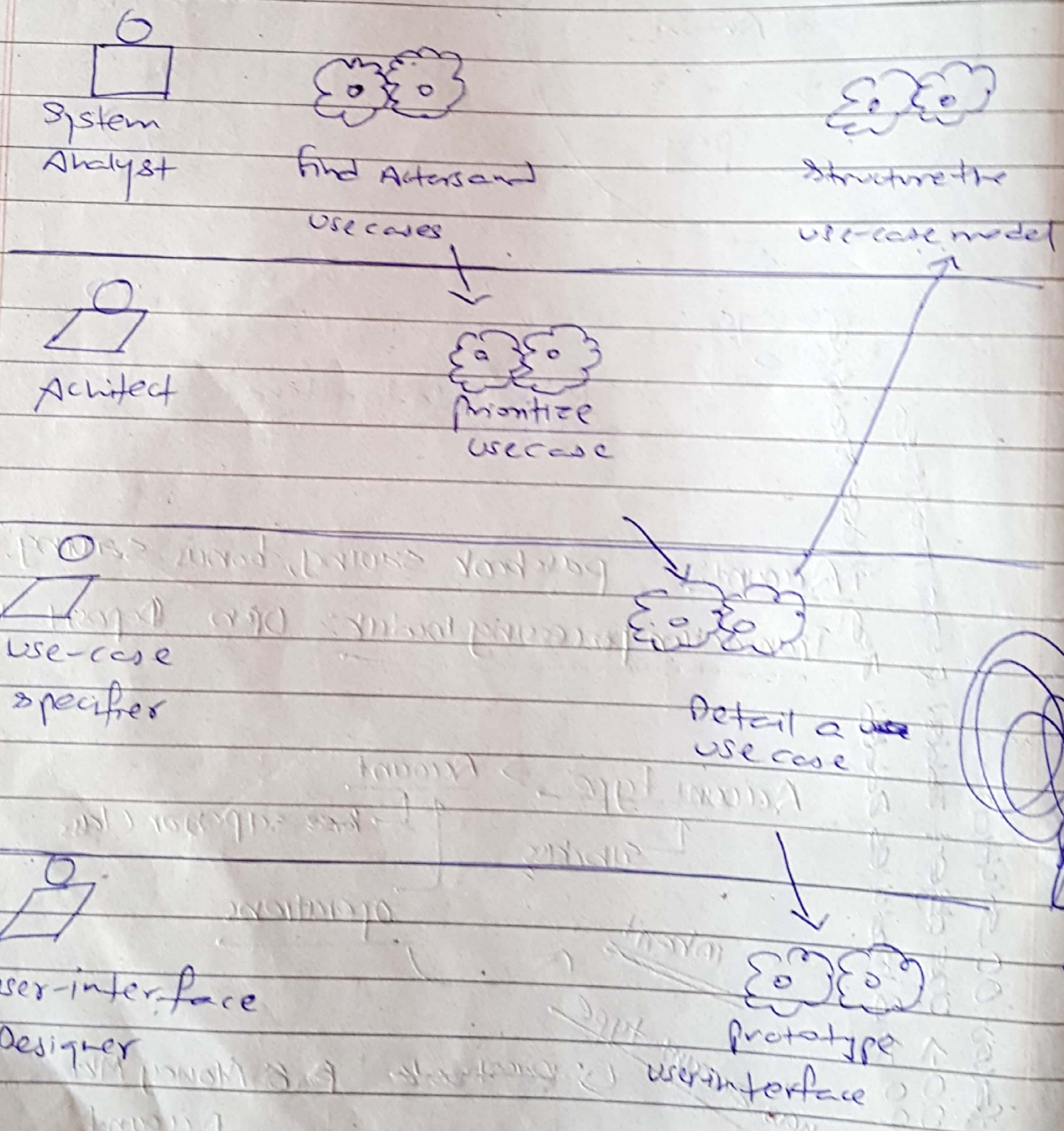
Add new account type
↳ ~~Passbook~~ ~~⊖~~ ⊖ Money Mkt Account

PO → ~~adding new func~~ account type to fun ~~adding~~. add a new a new module in each valid transaction

OO → only the adding of the new subclass account class & ↑

We would follow OO solution:-
OO → one object added & one modified
PO → 3 modules added and ~~three~~ three modified

Workflow for capturing requirements as use cases including the participating workers and their activities.



is used

An activity diagram shows the dynamic behavior of the workers. The diagram uses swim lanes to illustrate which worker performs what activities

first, the system analyst (an individual supported by a team of analysts) performs the Find Actors and Use cases activity to prepare a first version of a use-case model. with actors and use cases identified. The system analyst should ensure that the developing use case model captures the requirements that were input to the workflow, that is the feature list and the domain or business model.

Then the architects (S) will identify the architecturally significant use-cases to provide input to the prioritization of use cases to be developed in the current iteration. The use-case specifiers (several individuals) describe all use cases that are prioritized.

And the user-interface Designer (several individuals) suggests suitable user interfaces for each actor based on the use cases.

Then the system analysts restructures the use-case model by defining generalization between use cases to make it as understandable as possible.

The result of the first iteration through this workflow consists of a first version of the use-case model the use-cases, and any associated user-interface prototypes. The results of further iteration would then consists of new versions of these artifacts where these

artifacts are completed and improved incrementally over the iteration.

System analyst will find actor and use-case activity during the inception phase, but in construction phase the system analyst will mostly make minor changes.

System development is model development:

There are various types of models

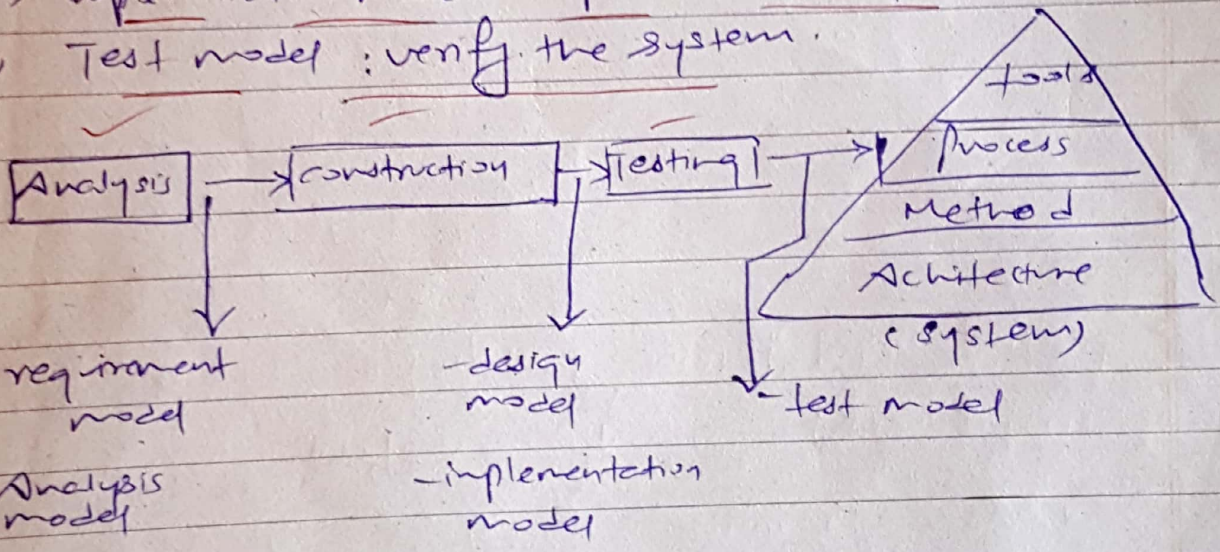
i) Requirenant model : Aims to capture the functional requirenant

ii) Analysis model: Gives the system a robust and changable object structure.

iii) Design model: Adapt and refine the objects structure to the current implementation environment

iv) Implementation model: Implement the System.

v) Test model : verify the system.



- The models are tightly coupled to the architecture.
- Aim of architecture is to find powerful modeling language, notation or modeling technique for each model.
- Modeling technique is described by means of Syntatic, Semantics and Pragmatics
  - Syntatic → grammer
  - Semantics → meaning
  - Pragmatics → heuristics