Date 20
M T W T F S S

Class Code: 5al5xsx [GCR]

Software / System Development Life Cycle.

↳ what is a system ↳ is an organizational combination of many process activities/processes.

System Study? Domain Knowledge

Feasibility Study? - Is the software something that is needed? Are you improving on something?
- cost-benefit analysis.

System Analysis } Requirement Gathering.
Design } UML Diagrams / Class Diagrams, etc.

Coding v/s Implementation —?

↓ ↓

... actually writing code || Going to a site & deploying it ...

... Testing: Test under different scenarios / unit testing / load, etc
↳ functional / Non-functional? Beta Testing?

Maintenance → Once you have deployed a system, maybe it becomes obsolete over time, then you might have to conduct a different study.

→ System Environments ←

Development
↓
Developer PCs / Servers

Pre-Production → Hardware / Software to be replicated here. First ~~they~~ ~~are~~ ~~done~~ ~~here~~ you can try things out here

Production → Data Related Errors.
Now you have "real" data to work with.

Test

Staging
↳ Have Multiple "Stages"
↓
3 iterations, usually.
→ Multiple Versions of a Software ["x" amount of versions running concurrently]

Mirror → Next Page

Difference b/w Beta Testing & Staging

Actual End-Users Testing is Beta Testing.

Page #

Date 20
M T W T F S S

Mirrors: To debug on client side. (after production)

↝ Basically a clone/copy of your Production. Customer doesn't know when they interact with it.

You make a mirror (for "x" amount of time)
The database is changed from production to mirror. (temporarily).

After the x time has passed, you revert all the changes from the mirror & then apply those changes on the live server. (The more time you spend on it, the more data you need to send back to production).

- There are specialists for this, who make scripts/ debugging plans

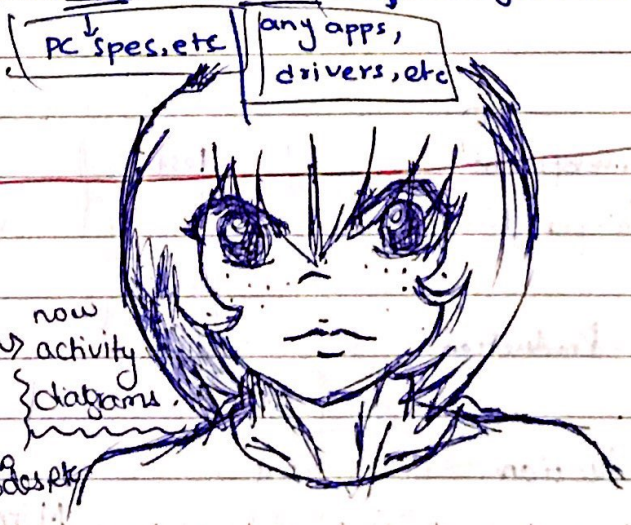
- Needs to be done very carefully. - less users / transactions, etc -

---

Developer, Development Manager, Project Manager, Test Manager, Configuration Manager, Deployment/Implementation Team.

Pre-production v/s Production

An environment before production, so you can test out the errors. [You try to clone the client's hardware & software on your end]



---

what terminology is being used

flow charts
Data flow Diagram (DFD) ↝ now activity diagrams
Data Dictionary
Structured English ↝ pseudo codes etc
Decision Trees

System Run Strategies

Date 20
M T W T F S S

(during implementation phase)

- **Parallel-Run**: old & New system are being run in parallel. Everyone is slowly shifted from one platform to the other.
- **Pilot-Run**: -If new system fails, old one is still running.

Module-wise implementation.

If installed module works as intended, then the next module is implemented.

→ Easier to spot where the error originates from.

(Between the implementation & maintenance phase)

OOAD

Object-Oriented Analysis & Design

Why oo: Design?

→ We're "engineering" a software.

Example: How does an engineer make things?
→ Blueprints? → Models? etc.

So, how does it apply on a software?

→ Different Types of Diagrams are needed here ←

Why do we do it? So we can visualize it & map it onto the real world scenarios.

Normally, you have an abstract model, which slowly becomes more & more concrete.

Requirements Gathering used to be a phase, but now it is an activity.

Date 20
M T W T F S S

⇒ The requirements keep changing /
More & more requirements keep arising.

So how do you decide when to stop gathering more?
In waterfall, etc, you kind of did a sign off, but you can't do that & make sure that the client is happy.

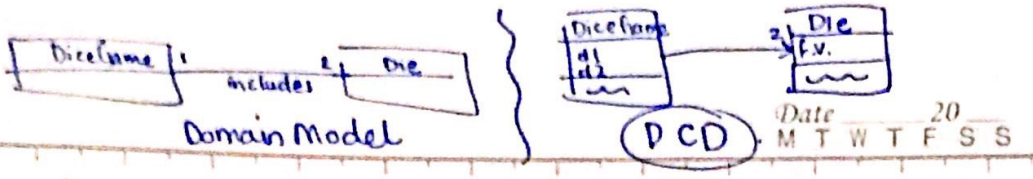
→ UML Unified Modeling language.

Used for specifying, constructing & documenting the artifacts of systems.

Three way to Apply UML

- UML as a Sketch:→ Diagrams → Informal & Incomplete. (hand drawn)
  Just to visualize the problem.
- As a BluePrint:
  - reverse engineering → code to diagram
  - forward engineering. → diagram to code.
- As a Programming Language:
  - Proper & complete execution of a system in UML
  - Done using Interaction & State Diagrams.
  - Still under development.

Agile Modeling → UML as a Sketch ←
high return on investment of time.



Date 20
M T W T F S S

Perspectives to Apply UML.

1- Conceptual
   ↳ Diagrams are representing a situation in the real world → Domain Mode

2- Specification
   ↳ Diagrams describe abstractions or components with specification or interfaces. (no specific classes). in any language.

3- Implementation. → Design Class Diagram.
   ↳ Diagrams describe software implementation in a particular language.

"Class" in different Perspectives

"Boxes" in the diagrams are "classes"

→ In Domain Model → They're called: Domain Concepts / Conceptual Classes.

→ In Design model / DCD, they're called design classes:

- Conceptual Class
- Software Class
- Implementation Class.

Date 20
M T W T F S S

Handling Change

→ Not uncontrolled. -- may cause feature-creep/ scope creep.

→ In each iteration you take a small subset of requirements an then design, implement, test

→ Then you see if the desired outcome was achived or not

So over here -- quick implementation = quick feedback

Instead of making the whole software and then testing, here you can get a feel for it beforehand.

User feedback → "Yes... but"?

→ any other tests like load testing, etc. You make sure that nothing needs to be changed, or if it does need to be changed.

Advantages of Iterative Development?

Time boxing.

↳ what to do if you can't complete it on time?

De-Scope remove tasks/ requirements & include them in future iterations.

⇒ Superimposing Waterfall on Iterative

- Requirement Gathering & Analysis.
- System Design.
- Implementation.
- Integration & Testing.
- Deployment of the System
- Maintenance.

⇒ Why is water fall failure Prone?

- maybe a false key assumption!

Therefore you need feedback & you need to incorporate it ChangeRates from 35-50%.

Iterative & Evolutionary Analysis & Design.

- Pre-Iteration - 1

1 : Have a small requirements workshop
: Identify use cases / features / non functional reqs.
→ HIGH LEVEL
: Pick 10% of the use cases.
: Do proper analysis of the 10% UC
functional / Nonfunction - reqs, etc.
: Plan the iteration #1 ; pick goals, etc
: Do iteration #1
↳ Sketch out the UMLs
↳ Program.
↳ Test
↳ De-Scope if needed.
Do step #1 again ; but now refine the requirements

Date 20
M T W T F S S

UP Phases:

not a requirements phase also neither requirements nor design phase

Inception: approx. Vision, business case, scope

Elaboration: refined vision, iterative implementation
resolution of hig risks.
identification of most reqs & scope.

not implementation phase

Construction: iterative implementation of lower risk & easier elements.
Preparation for deployment.

Transition: Beta Testing, Deployment.

Cover first

WHY DO WE WANT FEEDBACK & ADAPTATION?

[Types of Feedbacks]

- Early Development feedback.
  - Programmer reads/implements specifications
  - Client demos to refine requirements.
- feedback from testing & devs.
  - Refine design & models.
- feedback from team progress
  - refine schedule & estimates.
- feedback from client/market place
  - ~~prioritize~~ prioritize the features.

UP is · Risk-Driven // · Client Driven.

identify & eliminate risks early on. Specially highest risks

} give client actual visual features that they care about.

↓

ARCHITECTURE CENTRIC

↓

building, testing, stabilizing core architecture.

Not having a common architecture is a high risk

Page #

Date 20
M T W T F S S

→ Agile Manifesto ←

Individuals & interactions	over	Process & Tools
Working Software	over	Comprehensive Documentation
Customer Collaboration	over	Contract Negotiation
Responding to change	over	following a Plan.

→ Agile Principles ←

→ online / in book. ←

Agile Modeling Implies:

- Minimal Modeling — this doesn't mean avoid modeling.
- Modeling Purpose: understanding & Communication, not documentation.
- Don't model all of the design, figure it out while you program.
- Use the simplest tools. White board & a camera.
- Model in pairs or triads
- Create Models in parallel.
- Exact UML isn't important.
- Know that models will be inaccurate.
- Developers should do the oo-design themselves. (Otherwise it ends up being like waterfall).

Date 20
M T W T F S S

• Inception • Elaboration • Construction • Transition.

UP is : Use-case Driven
: Object-Oriented
: Iterative & Incremental/evolutionary.
: Risk Driven / Architecture Centric.
:

Inception →
- vision / business case.
- feasible?
  - buy / build?
  - Range of cost (rough)
  - continue?

Phase Deliverables for Inception →
Vision / Business Case.
Use-Case Model.
Supplimentary Specification.
Glossary.
Risk List / Mitigation Plan.
Prototypes & Proofs of Concepts.
Iteration Plan.
Phase Plan & Software development Plan
Development Case.

F U R P S
functional usability Reliability Performance Supportability

+ => Implementation - resource limitations, language / hardware
Interface - constraints imposed by interfacing w/ other systems
Operations - system management.
Packaging.
Legal - licensing.

Date 20
M T W T F S S

Actors
Scenarios
Use Cases. ⇝ text documents.
Use-Case Model → writing text, not drawing diagrams.

what do use cases handle from FURPS+?

Types of actors:
- Primary
- Supporting - some external service.
- Offstage → other stakeholder.

Types of use cases:
- brief - main success scenarion
- casual - ↓ & alternate
- fully dressed.

{

Use Case Name
Scope → SuD
Level → user goal | subfunction.
Primary Actor
Stakeholder & Interests
Preconditions
Success Guarantee
Main Success Scenarion
Extensions → alternate & scenarios.
Special Requirements. → Non-functional.
Technology & Data Variations List
frequency of occurance
Misc → any open issues?

Association ——— can have multiplicity on it too

Implementation - - - ▷ (Like with Interfaces)

Composition ◆—— (`xx = new x();`) x scope within class. (actual object).

Aggregation ◇—— x can exist outer of class (reference of object).

Generalization ——▷ (Inheriance)

Examples:

human, Brain, Heart, Legs.

Bicycle, Person, frame, wheel.

Agg

player & team

Comp

book & chapters

* 0 - many
1...* 1 - many
1...5 → 1-5
5 → 5 exactly

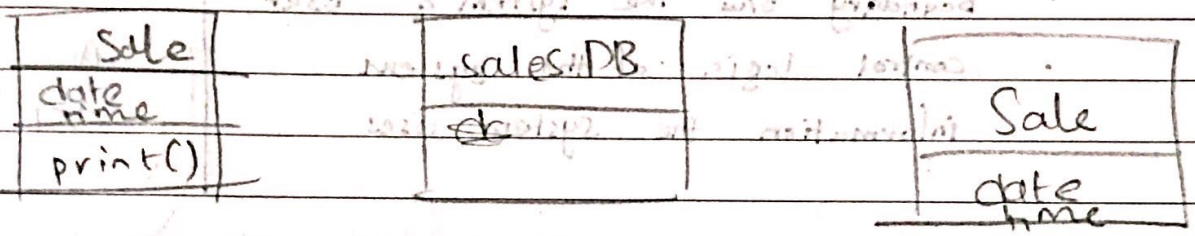
4,5 , (4 or 5)

Date 20
M T W T F S S

Domain Models?

→ visual representation of conceptual classes or real-situation objects in a domain.

Also called as a "visual dictionary".



You make domain models so you can understand the key concepts & vocabulary.

Attribute or Class?

Airport → class?

flight
↳ destination.

Sale
↳ store.

or...

Store → class

Normally "attributes" will be datatypes
↓
not concepts.

Date 20
M T W T F S S

Entity Control & Boundary Classes.
(Key system entities) (Interface)

Identifying Candidate Classes from behavior.

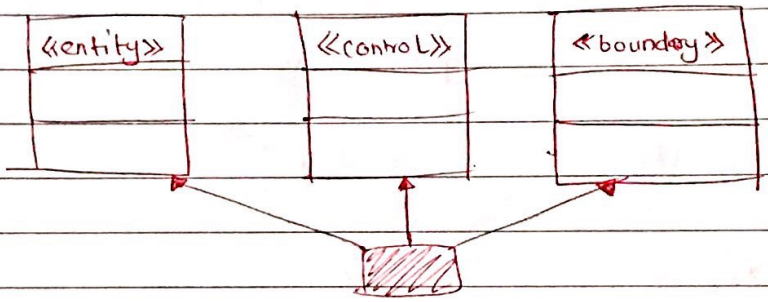
→ we use Stereotypes to represent them « » ← stereotypes.

- Boundary b/w the system & user
- control logic of the system
- information the system uses

- Robustness Classes
- Analysis Classes
- ECB

→ same thing.

It allows us to make a robust model



You first identify these & then convert them to an actual Design Class Diagram -

---

Analysis Classes represent an early conceptual model. for things that have responsibilities & behavior.

They handle

- Primary functional Requirements.
- Interface Requirement.
- Control.

• Boundary ⇒ user interface { system interface.

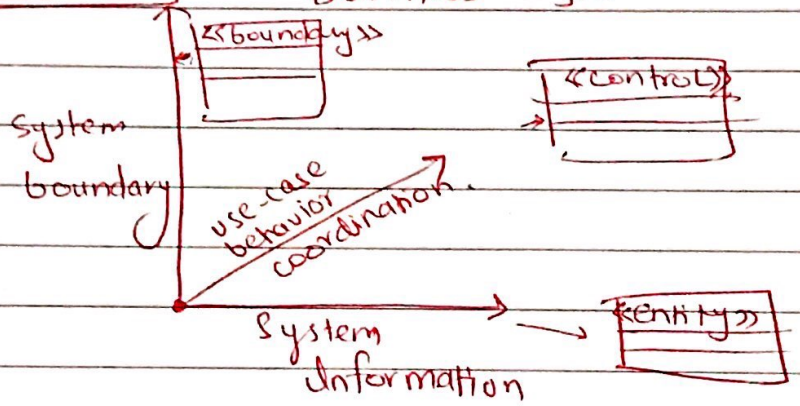
one system talking to → user / system / device } Make a boundary Class.

{Between a system & its user}.

Date _____ 20___
M T W T F S S

Entity Class → Stores data || any persisted data

Control Class → Business logic



Actors can only communicate with boundary classes.

Control can talk to boundary & entity classes