Software Quality Engineering

- programs - documentation - data.

Classification

- Platform -system s/w
- nature of s/w (what does system do) or embedded s/w
- generic/bespoke → custom made for someone specific = flex
  ↳ meant for oper market. Anyone can use it.
- open / closed sources
- free / proprietary

⊙System: collection of related components ~~which~~ serve a common goal
eg. biological system: plants & animals.
Mechanical: bicycle.

- Computer based systems:
  ↳ special type: software intensive comp based system
    ↳ depends upon the role of software in that system

1) Nature of software:
   Its logical or conceptual entity which cannot be perceived by 5 senses.
2) Size of software: (scale of software itself makes it complicated) quality assurance
   We have to use techniques like abstraction & decomposition to make it handled properly.
3) Complexity:
   Infinite no. of input combinations
4) Contractual: (already decided delivery date)

software supplier —— contract —— software Consumer

5) Developed in a Team: [also adds to complexity]
Sometimes onshore, offshore settings.

6) Interaction:
One s/w may need to interact with another s/w
eg. Flex talks to admission system.

7) Change:
↳ change in requirements
↳ even team working on software may change
↳ people also leave the project midway (eg: he/she got Fired)
↳ we need to use a disciplined approach in this situation called ENGINEERING.

Engineering: systematic, disciplined

How s/w is engineered?

1) Inception: idea of s/w is conceived. Note down unmet needs and automated solutions. Planning done by project managers. Contract is made where high level requirements are mentioned.
2) RE: RE activities like elicitation happen here.
3) Analysis: modelling of problem space.
4) Design: Modelling of solution space
high level design: architecture
low level design: component level design.
5) Implementation
6) Testing
7) Deployment: worry about installation operations
8) Maintenance: based upon type of problem
bugs = corrective, Newfeature: Editive

adapting to new feature : adaptive.

9) Retirement: s/w no longer needed.

← Quality →

conformous to requirements

* compromising quality can lead to ~~spectacular~~ spectacular failures.

Lec 3

Software Quality degree of ~~conformous~~ conformance to all implicit & explicit & technical & non technical requirements.

- explicit req are documented eg SRS or user stories
- implicit not documented eg. following good s/w development practices. &
- quality of s/w may suffer even if implicit req are not followed.
- Technical : — FRs
  — NFRs. constraints
  ] Both are important.
- Non-Technical (managerial)
  - budget
  - schedule
  ] (within budget & time should a s/w complete)
- More degree of conformance better quality.

SOFTWARE QUALITY PROBLEMS

- Most imp ingredient for s/w development is People!
- In Inception: project-manger may not estimate budget & risks & tasks properly. May not draw out req. properly.

- RE: req. can be incorrect, ambiguous, missing. SRS incomplete. req. are extra.
- Design Phase: failed to consider imp req. Algos or designs incorrect. Issues in interfaces.
- Programming: not checking for errors eg. Arrays out of bound. not using meaningful variable names.
- Testing: Test cases missing, plans missing. Not all cases executed. All defects not reported. No regression testing done.
- Delivery. User manual not complete. Training not done

- potential for problems in all phases becz humans are stoopid. They make mistakes. :)

How To Fix

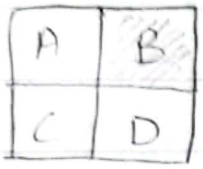
- Prevention or detection or minimize the effects.
  ↳ or remove

Classification Of Problems

1- Errors (made by humans)
   ↳ result of incorrect human action
   ↳ selecting wrong datastructure.
   ↳ incorrect variable initialization
2- Faults (state of s/w) ≡ bugs ≡ defect
   ↳ becz of error state becomes faulty state.
   ↳ some errors cause faults. not all.
   ↳ You can't have a fault without an error.

3- Failure: (event)
↳ not all faults become failures. Some do.

Example: lets say s/w have 4 modules & B is faulty
- But lets say end users never use B so it doesn't become failure.
eg. bugs exsist in MS Word but we don't know.



- failure is a triggered or an activated fault.

4- Incident: some failure become incidents.
↳ failure becomes incident when someone notices it.
eg. the s/w corrupts database but if noone looks at it, it is not incident.
eg. Flex may not calculate cgpa properly but you never noticed it cuz you never compared it with your calculated cgpa. Inshoort, andha bharosa karna on flex & never verifying cgpa. So this is not an incident unless someone notice it.
* a severe failure is an ACCIDENT but the term severe is ambiguous / vague.
* The developers are more concerned about erroos & faults.
* Consumers are concerned about failures & incidents.

- If there are errors try to prevent, detect, remove or contain them in one module.

Software Quality Assurance

Slw Quality Assurance is an umbrella activity that consists of acts of planned, disciplined and systematic actions to provide adequate confidence during development and maintenance to conform to all implicit & explicit, technical & not technical requirements.

Slw Quality Control

↳ its part of SQA.
↳ it comes into action when product is ready but not shipped.
↳ prevents poor quality products from being shipped
↳ Involves Testing.

- SQA : process oriented. → its proactive.
- SQC : product oriented.
  ↳ its reactive.
- SQA starts from Inception & ends at retirement.
- SQC is testing.

Lec # 4

- We can't measure quality without, SRS.

McCalls QF Model (1970s)

- Product Operation Category (has 5 quality factors)
- " Revision (has 3)
- " Transition. (has 3)

- Total factors = 11.

* Product Operation: (day to day usage)
  - Correctness
  - Reliability
  - Efficiency
  - Integrity
  - Usability.

* Product Revision: (changes made to s/w)
  - Maintainability
  - Flexibility
  - Testability

* Product Transition
  - Portability
  - Reusability
  - Interoperability.

1) Correctness - huge / largest dimension / quality.
   - contains all func. req.
   - All outputs produced by func are part of it.
     - eg. displaying transcript, audios playing.
   - it displays outputs accurately.
   - all outputs needs to be complete
   - up to date honi chahiye outputs
   - availability: defined in quantity. eg. Reaction time is 2 secs.
   - adhere to standards. eg. programming & design.

2) Reliability (deals with failures)
↳ system that fails alot is not reliable.
↳ Down time & fail times comes under here.
↳ Reliability is more imp for critical systems like that of hospital. Not imp for entertainment system eg. games.

* NF req. are sometimes called 'ilities'. ends with ility eg. reliability

3) Efficiency:
↳ if a s/w system uses less hardware resources, its more efficient. resources like battery, CPU. This is imp for embedded systems.

4) Integrity (security)
↳ Authentication: verify if person is who he is claiming to be.
↳ authorization: deals with permissions to use of features

5) Usability: (ease of usage)
↳ Training: How much time is required to train a person to use s/w.
↳ Operational: how much time s/w takes to perform a task.
↳ UI/UX focus on these 2 things.
↳ If info on frontend is in all caps. readability reduces. Words shapes are reduced. Adults look at word shape to read a word. Slows down reading.

6) Maintainability (Maintenance & Evolution)
↳ changes, revisions, modifications.
↳ sometimes we spend more time on maintenance than s/w itself.
↳ Corrective maintenance: Fixing bugs; defect identification & removal
↳ Adaptive: Adapt s/w to new settings, users, environment.
↳ Perfective: Improvements eg. improving efficiency of algorithms.
↳ Additive: Adding new features. Also called functionality enhancing maintenance.
↳ s/w evolves over time.

* According to MCCall, Maintainability deals with corrective maintenance, Flexibility with adaptive, perfective and additive.
* Maintainability & Flexibility deals with CHANGE.
↳ Following coding conventions help with Main & flex.

7) Testability (complete it)
Deals with ease of Test. (Intermediate outputs, self-diagnostic features, safety standards)
Example: Troubleshoot Feature, Power-on self test Bios, etc.
* Power-on self protocol ran by BIOS installed in motherboard to check hardware working fine before OS.

Lecture 5

Product transition deals with you as s/w.

8) Portability

The s/w should work on other hardware or operating systems. Diff browsers.

- Java has an advantage.
- Having portable s/w increases customers.

9) Reusability

↳ 'With': Developing your s/w with reusability

↳ 'For': " " " for reusability.

↳ With is related to past. developing s/w using past s/w components.

↳ for is future. Developing so future s/w can work.

↳ save time, fewer resources. Good quality. saves energy and cost.

↳ ecommerce s/w me order & billing modules reuse kar sakte.

↳ we got libraries which gives us components.

↳ requires careful designs (if used in future)

↳ design patterns enable reuse.

↳ codes need to be well documented.

↳ Developing something with mindset to reuse it requires effort but saves effort & time in future.

10) Interoperability
↳ A lot of s/w cannot work in isolation.
↳ list-down all other s/w your s/w talks to.
↳ ABC ⟷ XYZ
↳ They have to exchange info in a certain format.
↳ eg. flex talks to administration system.
↳ In SRS Interoperability has to mention who flex needs to talk to.

- also called classical Quality factor
MC Call's other name.

QF Model	Year	# of factors	
MCCall	1977	11	
Evans & Marcinaik	1987	12	+5
Deutsch & Willis	1988	15	-1

* New models overlap with MCCall & combined they added 5 new qualities. And Testability factor was missing in newer model.
11) Verifiability, Manageability, Survivability, Safety and Expandability.
- Expandability is similar to flexibility
- Survivability " " " Reliable.
* So technically 3 among these 5 were added becz 2 are similar to before.

Verifiability & Expanability were proposed by both.

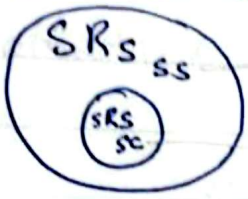
1) Verifiability
   - ↳ Your program should be formally verifiable.
   - ↳ Natural lang. are considered informal.
   - ↳ UML = semi formal
   - ↳ formal: OCL, z notation. .. etc.
   - ↳ Programshould be written/documented using these formal languages.

2) Safety:
   - ↳ should have safety hazards.
   - ↳ if temp rises above 1000°C, the system should have alarm or should shut down.
   - ↳ Threats vs hazards.
     - ↓ cybersecurity
   - ↳ safety req. prevents hazards. Security prevent threat

3) Manageability
   - ↳ have administration tools. eg. Git.

* 14 distinct quality factors are provided by all 3 models.
* not all these factors are relevant to all projects. You atleast have a checklist with you where you can pick relevant qualities for your project.

We got 2 SRS. one for supplier & 1 for consumer



Supplier SRS is superset of consumer SRS

Qualities that are concern for suppliers:
- Reusability (reuse req. or components). Appear in SRS of supplier.
- Verifiability (part of SRS_{SS})
- Portability (")(Can be part of consumer side too)

Lec #6
SQA System Components

① Pre-project components (specific)
- Goal is to improve quality of preprietary steps even before the project starts.
- **Contract review**: SS & SC sign a contract.
  ↳ make sure comitments are documented.
  ↳ " " both sides are capable of fulfilling their parts
- **Plan**: (both development & quality)
  ↳ Dev plan include your work breakdown structure. Prepare resource list & allocate it. Estimate time. Risk plan (for future)
  ↳ List quality goals. List QA activities. Diff. testing that has to be performed. Who & when will those activities be performed
- quality suffer if s/w delivered late or not within budget

⊙ Life cycle component (specific)
- split into 2 stages. Dev & maintainance stage.

Ves Inc RE A D I T D M R
b Dev maintenants.

1) - Reviews: focus on documentations made through the development. eg. Review test cases.
↳ Types of reviews:
- FDR, Inspection, Walkthrough.
formal informal.
↳ These are team based reviews.
↳ Inspec & walkthrough are peer review.
↳ FDR: authorative & grant approval.

2) - Testing: related to code.
- We will learn about stages later.
- Integration, unit & system testing.
- Has testing techniques: whitebox & blackbox testing.
- Manual vs Automated testing.
- Autonomous testing: agents doing it.
- How to design test-cases (will learn)

- Maintenance:
↳ maintenance types

- Expert opinions
↳ when you ~~don't know~~ can't setup a review team you hire experts.
↳ when there is disagreements blw seniors.

Sub1 — SS1 — SC
⋮ ⋮
Subn SSn

sub: sub contractors.
called external participants.

↳ You hire to cut cost. Take advantage of diff time zones
eg. hiring indians from america is cheaper.
↳ basically Outsourcing.

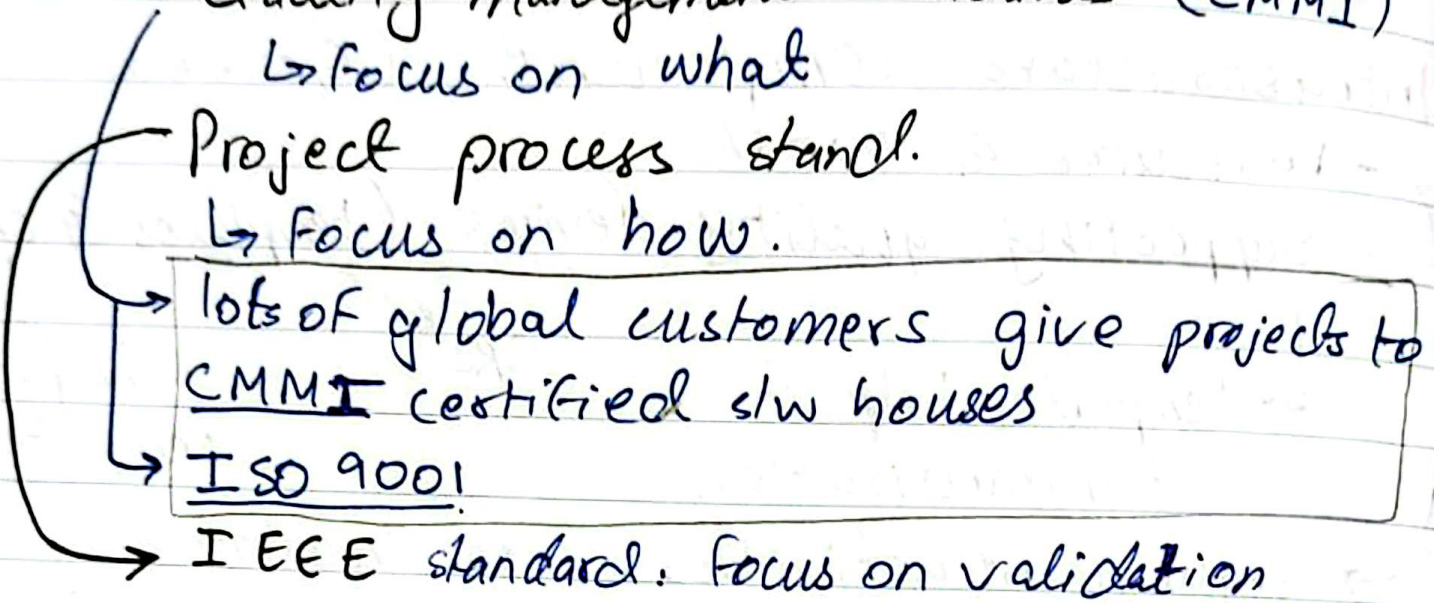
⊙ Infrastructure component (General)
- Procedure & work
- Supporting quality devices (templates & check lists)
  ↳ templates helps in standardization
  SRS
  help in completeness
  ↳ sequence of activities done in certain way.
- s/w consrderation management.
  ↳ manage change at diff points.
- Staff training
  Train & retrain your staff.
- Corrective and preventative actions

⊙ Management component. (specific)
- progress control
  continuous monitoring & taking action if not going according to plan
- Quality matrix
- Cost : we have to worry about quality cost.
  ↳ failure cost
  ↳ cost of performing
  -with performing more QA activities, cost of failures ↓.

We have to manage total cost.
- There's a certain limit to perform QA activities

⑥ Standards -related components (General)

- Standards & certification
- organization related ↑
- Quality management standards (CMMI)
  ↳ focus on what
- Project process stand.
  ↳ focus on how.



⑥ Human components (generic)

- management have diff tiers of managements
  ↳ senior manag: r budget-making policies
  ↳ middle & lower
- SQA units make sures QA activities are performed.

* look at diagram in book *

Tailoring these according to our own organization ↔

↳ Diff s/w houses has their own SQA systems.
↳ SQA systems has to be tailored.

Factors of Tailoring

1) Organizational :- size of organization. More SQA components and efforts.
   - Range of projects developed. Bigger range more SQA.
   - Organizational complexity: more external participants more SQA components

2) Project - how complex project is.
- more complex, more components.
- size of project makes it complex.
- Degree of reuse: Higher degree of reuse, lesser the components needed.

3) Staff / personnel
- experience: more experienced, lower comp.
- qualification: " qualification " "
  ↳ related to degrees.
- Degree of acquaintaince: how well team members know each other
  less acquintance, higher components.

Lec # 7
Integration of SQA activities

Software Processes
- it is an approach to s/w development & maintenance
- Framework where s/w dev & main. activities takes place
- Also called life cycle / roadmap

Processes are classified into
1) Traditional
2) Agile. → change friendly
] two diff. mindsets of s/w dev & main..

↳ also called conventional, heavy weight ... etc.] no need to memorize.

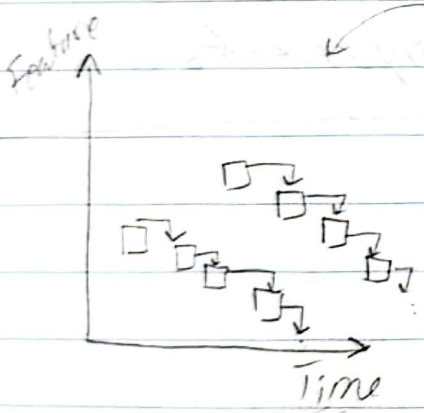
1) Traditional
   a) Evolutionary → accomodate change
   b) Non-Evolutionary → least change friendly
      ↳ waterfall. (oldest one): large in size project with stable requirements. s/w delivered late.

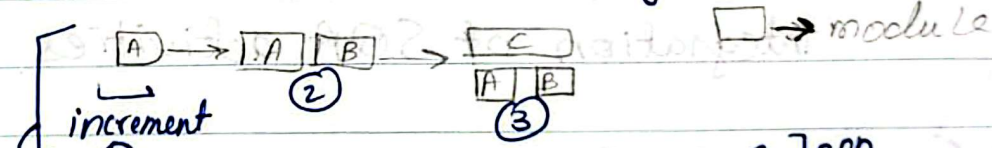
Difference b/w process & models:

abstraction of a model.

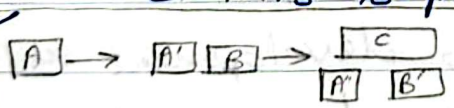

] model of waterfall process.

↳ Incremental (non evol.)
- s/w delivered earlier.
- Still following waterfall.
- Customer can start using it earlier.





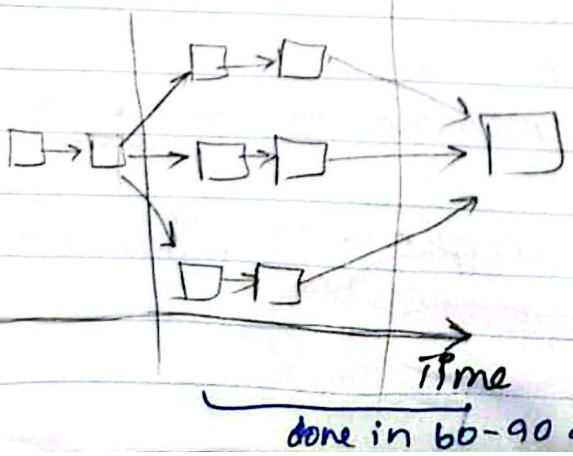
① This is pure incremental] non evol.



] incremental + iterative] evolutionary

↳ change is happening here

↳ RAD (non-evol)



- assumes work can be split easily.
- needs lots of manpower
- strong coupling makes RAD difficult to use

(2) Evolutionary:

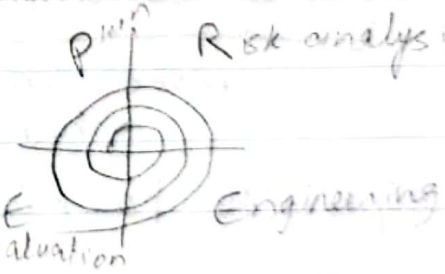
↳ Prototyping

- meant to change
- used when req are uncertain
- customers don't have clear ideas
  ↳ They don't know what they want.
- Customer side suffers from
  IKIWISI : I know it when I see it.
  ↳ show me prototype & i'll know what I need.
- For small medium sized projects.
- nightmare for s/w managers.
- Cause management related problems.
- Plans change continuously
- Customer highly involved. Probability of satisfying the customer is high.
- The design is not much solid.
- Throwaway vs build upon prototypes.

not meant to be delivered. ↘ difficult to change later.

↳ Spiral

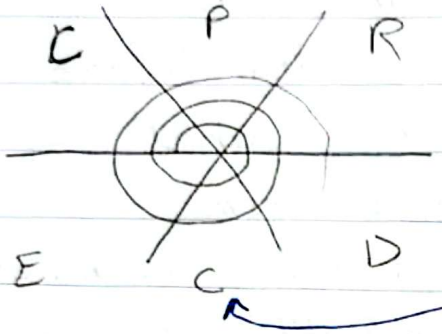
- It works for all types of projects.
- also called risk driven process.
- Engineering could be making SRS, designing, prototyping etc. Not only coding. Making any process inside it.



- First spiral could be 1st increment & so on.
- Evaluation is getting feedback from customer.

- downside: requires risk analysis expertise
not easy to manage these.

Advance spiral ~~process~~ Process



↙ called win win ~~good~~ process
C: communication
P: planning.
D: design.
C: construction

2) Agile Manifesto → 4 clauses.

- Individuals & Interactions vs Processes & Tools
more imp than →

- Working s/w vs comprehensive Doc
more imp than →

- Customer Collaboration vs Contract Negotiation
more imp than →

- Responding to change vs following a plan
more imp than →

↳ means that RHS is imp too but LHS is more imp.

models:
- XP
- Scrum

- increases simplicity: maximizing the work not done.
- " imp of customer satisfaction by welcoming
- Sustainable development: no overtime change.
  ↳ team can work achieve 😊

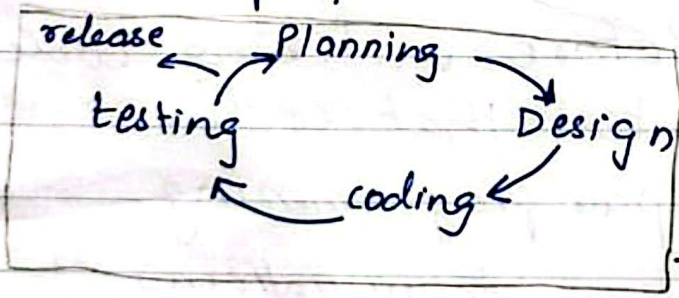
*Traditional says process is king & people should adapt to it.
*Agile says people are king & process should adapt.

Lec # 8
Agile Processes

1) XP
2) Scrum

ss-sc

1) Extreme programming.
   ↳ It proposes 4 steps/stages.



Planning:
- Both ss & sc participates.
- User stories are used to extract req

"As a student I want to register in a course so I can study" ↖ user story.

- Both func & non-func can be written with user stories
- XP has planning game. For each user story one side (SC) provides value and one side (SS) provides cost. Cost means how much effort is poured to produce user stories.

- Highly valuable, easy to implement user story has low cost.
- once all user stories gathered, we can plan which to develop in next iteration.
- Both sides try to reach a consensus on which user stories to implement.
- XP advocates simple design.
- Iteration plan.

Design: CRC. class responsibility collaborator.
(attributes) (class referring to another class)

- Every class fulfills responsibilities
- Collaborators help that class fulfill responsibilities.
- Spike solution: for complicated designs. → quick experimental prototypes
- constant refactoring of design. without changing external behaviour, you improve your code.
- Constant refactoring is one of XP principles.

Coding:

- Test driven development (TDD). Before writing code you make test cases to test that code.
- One of common practices is pair programming. (process pattern)
(unit testing here)

Testing:

- Unit already done in coding.
- Acceptance testing also called user acceptance test. code. Once user satisfied you release s/w.
- No. of user stories implemented is called project velocity.
- You check whether code implements user stories or not.

- You can prepare test cases for acceptance test in planning too.
- Based on project velocity you can plan your next iteration.

XP principles: - Customer should be co-located with dev team. meaning he should be part of dev team.
- It advocates a 40 hour work week. NO OVERTIME (Yay). Eventual employee turn over.

* user stories are written in plain / simple angrezi.

2) Scrum.
- Diff b/w Scrum & XP is XP provides technical side/factors. Scrum provides managerial factors.
- Scrum has process pattern.
- It proposes 3 roles: scrum master, scrum owner (project manager; - req. engineer, - represent customer) and development team owner.

- Product backlog: has feature list. The high features are on top.
- In scrum every iteration starts with sprint. ↳ in sprint plan we decide which features from product backlog we will implement & put them in sprint backlog.
- Spring backlog is subset of Product backlog.
- who chooses features? Both sides.

- Plan gamification is in scrum too.
  ↳ Planning poker: Card based game. Every person ~~in~~ part of planning team has their own deck of cards & they ~~have~~ labelled with fibbonacci series. The numbers tells how much effort it will take to implement it. They select one card & reveal at same time.

* you can play PP virtually as well. Websites exist.
- Every sprint is time boxed. Time is fixed. Small iteractions are recommended.
* Size of timebox determines how many features will be implemented.

There are scrum meetings too. Standup small meetings.
  ↳ There are 3 questions in meeting. What did you do in past 24 hours. What u will do in next 24 hours & what problems are you facing.

- There are 2 meetings at the end of sprint.
1) Sprint review: with customer. provide demo. Product related. Get feedback.

2) Sprint retrospective: between team: how well you have done. Process related.

- Then based on feedback you plan next iteration.

- Technical guidance not present here.

* Scrum most followed by Pakistani s/w houses.

CBSE ↔ = component based s/w engineering.
- also called object oriented s/w development.
- object means components & modules.
- advocates reuse.
- starts with uses stories. Identify how many components will be there. You check reuse library to find if there are any already made components.

First step (after getting req.): Assessment: checking which component you will make & which you will reuse. Saves time, improve quality. Save effort

* You can use CBSE with other processes.
- Developing component from scratch is "for" reuse in future

Foundation for Advancement of Science & Technology

Lec #9

SQA

- verification
- Validation
- qualification

These are 3 different aspects of SQA.

* In programming phase: you verify the code with design (previous phase).
* ~~Test~~ Verify if output conforms to conditions imposed by inputs.

Validation:

↳ Always from customer's pov. Validate code if it fulfills customer's req. Validation is done respective of requirements.

- focus on customer's interest.

Qualification: does the code conform to the good coding practices. operational aspect * me problem comes due to maintenance. *

eg. You're writing a func to calculate area of triangle. You verify if the formula used is correct. Validation: if coded properly or if customer wanted area or not.

If this code isn't written ~~as in~~ following good programming practice eg variable names are A, B, C so in this case qualification fails.

Foundation for Advancement of Science & Technology

verification ↓ is product right?

validation ↓ are you making the right product?

- models helps in helping you understand things.

New model

It assumes s/w project follows waterfall model

This model treats all QA activities as filters

↳ filter: removes impurities.

↳ removes defects. Allow some defects to pass through and ~~sop~~ stops some.

↳ meaning some defects stays in project.

* Some defects are eg. introduced only in design phase that are pure design defects. Some comes from previous phase eg. req. gathering.

- It assumes filtering is 100% accurate. meaning no defects remain.

* All QA activites are integrated in phases. The dev. activities introduce defects & QA activities removes them.

$$RD = (POD + PD) \times \% \cdot \frac{FE}{100}$$

$$= \frac{120}{100} \times 50 = 60.$$

$$TRC = RD \times CDR$$

$$= 60 \times 16 = 960 \text{ cu}$$

* The PD for next phase is
Total - RD
= 60

Lec 10

Contract Review

↓

Pre project Component

SS — SC

↳ focus on the

* Companies have legal departments. Those who don't, contact ~~other law~~ outside lawyers. Netsol has its own legal department.

- Internal project: eg. IT dept make project of marketing dept in the same organization

RFP = request for proposal.

SC publishes RFP. Diff s/w houses companies sent their own proposals to SC. Proposals + drafts + Review

this proposal submission is also called bids

RFP → Proposal → Proposal → Contract → sign contract
(+ proposal accepted)

Proposal: + Draft + Review
Contract: Draft + reviewment

at this point SC gets multiple proposals from diff SS.

Part of contract ~~draft~~ Review

Goals & objective of why we do this:

- Requirements clarified:
  - ↳ Becz you have to estimate time and risk.
- Alternatives:
  - ↳ alternate approaches to doing something.
- Relationship:
  - ↳ eg. who will make srs & when will it complete
- Risks
  - ↳ very imp for large size projects. Risks identified.
- Estimate:
  - ↳ Both human & nonhuman resources
    - human: testers, programmers
    - nonhuman: ↳ servers, tool, schedule
- Capacity
  - ↳ SS & SC ki capacity is examined. checking if SS is capable of making a s/w & SC are able to pay.
- Participants
  - ↳ identify subcontractors. How profits shared. If there's a panelty who will pay.

- IP : intellectual property
  ↳ who will own the s/w at the end.

* These all determine how much you will charge customer. If customer says no reusability, then charge more.

- - - - - - Till now was Proposal draft review - - -

Contract draft review:

- Unclassified
  ↳ no ambiguous clauses, statements.
- Extra clauses
  ↳ nothing is extra. If SS & SC hasn't agreed on something it shouldn't be there.
- No missing clauses.

has many participants, size huge, complex, criticality

* For major projects, you end up doing all of these.
* read checklists from book.
* For small projects 1 single person is enough for Contract review. But for large, make a team.
* Sometimes outside consultant(s) are involved too.

Internal project.

- SS — SC = This relationship is informal

  so SS ~~~ SC

- Projects made for eg. open market. or diff dept of same company.

SS — SC] example.

↙ software dept

↓ hardware dept

- usually formal contract doesn't get signed but this can cause problems.

- Suppose req. were not clarified, low customer satisfactio
Customer requests changes - schedule slips, cost overruns, extra effort, project late.
So idea is, make formal contract too internal projects too.