

## 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 open market. Anyone can use it.
- open / closed sources
- free / proprietary

④ System: collection of related components ~~to~~ 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:

It's logical or conceptual entity which cannot be perceived by 5 senses.

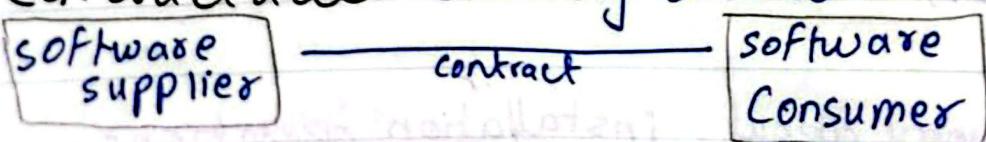
2) Size of software: (scale of software itself makes it complicated)

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)



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  
e.g. flex talks to admission system.

7) Change:

↳ change in requirements

↳ even team working on software may change

↳ people also leave the project midway (e.g. 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, New feature: Editive

adapting to new feature : adaptive.

9) Retirement: s/w no longer needed.

→ Quality →

conformous to requirements

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

Lec 3

conformance .

Software Quality degree of ~~conformous~~ 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      ] (within budget & time
  - ↳ schedule      ] 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 manager 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 bcz humans are stupid. 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)  $\equiv$  bugs  $\equiv$  defect
  - ↳ bcz of error state becomes faulty state.
  - \*  $\hookrightarrow$  some errors cause faults. not all. \*
  - ↳ You can't have a fault ~~a~~ without an error .

### 3 - Failure: (event)

↳ not all faults become failures. Some do.

Example: lets say s/w have 4 modules & B is faulty

A	B
C	D

- But lets say end users never use B so it doesn't become failure.

e.g. bugs exist 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.

e.g. the s/w corrupts database but if no one looks at it, it is not incident.

e.g. Flex may not calculate cgpa properly but you never noticed it cuz you never compared it with your calculated cgpa. Inshot, andha bharosa karna on flex & never verifying cgpa - So this is not an incident unless someone notice pt.

\* a severe failure is an ACCIDENT but the term severe is ambiguous/vague .

\* The developers are more concerned about errors & 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

SQA Quality Assurance is an umbrella activity that consists of a set of planned, disciplined and systematic actions to provide adequate confidence during development and maintenance to conform to all implicit & explicit, technical & non-technical requirements.

## SQA 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.

McCall's 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, audio playing.

↳ it displays outputs accurately.

↳ all outputs needs to be complete

↳ up to date honi chahiye outputs

↳ availability: defined in quantity. eg. Reaction time <sup>is 2secs.</sup>

↳ adhere to standards. eg. programming & design.

## 2) Reliability (deals with failures)

- ↳ system that fails a lot is not reliable.
- ↳ Down time & fail times ~~comes~~ 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, it's 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 <sup>to use</sup> of feature

## 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 in flexibility deals with CHANGE.

- ↳ Following coding conventions help with Maint & 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 run by BIOS installed in motherboard to check hardware working fine before OS.

## Lecture 5

Product transition deals with you in s/w.

### 8) Portability

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

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

### 9) Reusability

↳ 'With' : Developing your s/w with reusa<sup>b</sup>

↳ 'For' : " " " for reusability.

↳ With is related to past. developing s/w

so doing using ~~past~~ <sup>part s/w components</sup>.

↳ 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 others in your s/w talks to.



↳ 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.

- ~~QF~~ also called Classical Quality factor  
McCall's other name.

QF Model	Year	# of factors
McCall	1977	11
Evans & Marcinaik	1987	12
Deutsch & Willis	1988	15

\* 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 bcz 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.
- ↳ program should be written / documented using these formal languages.

### 2) Safety:

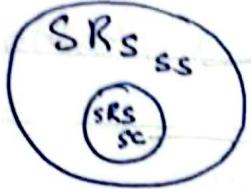
- ↳ should have safety hazards.
- ↳ if temp rises above  $1000^{\circ}\text{C}$ , the system should have alarm or should shut down.
- ↳ Threats vs hazards.  
↳ cybersecurity
- ↳ safety req. prevents hazards. Security prevent threat

### 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

→ consumer

Supplier SRS is superset of consumer SRS

Qualities that one concern for suppliers:

- Reusability (reuse req. or components). Appear in SRS of supplier.
- Verifiability (part of SRS<sub>SS</sub>)
- Portability ("") (can be part of consumer side too)

## Lec #6

### SQA System Components

#### ① Pre-project components (specific)

- Goal is to improve quality of proprietary steps even before the project starts.
- Contact review: SS & SC sign a contract.
  - ↳ make sure commitments 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

## ① Life cycle component (specific)

- split into 2 stages. Dev & maintenance stage.



- 1) - Reviews: focus on documentations made throughout 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 : authoritative & 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).

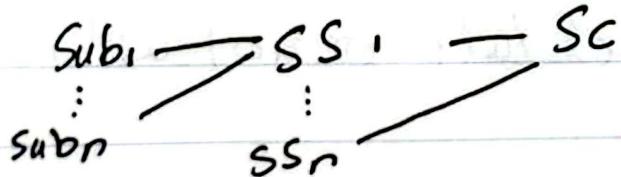
- Maintenance:

↳ maintenance types

- Expert opinions

↳ when you ~~don't have~~ can't setup a review team  
you hire experts.

↳ when there is disagreements b/w seniors.



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.

## ① 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 configuration management.
  - ↳ manage change at diff points.
- Staff training
  - Train & retrain your staff.
- Corrective and preventative actions

## ② 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



- ① Standards-related components (General)
- standards & certification
  - organization related ↑
  - Quality management standards (CMMI)
    - ↳ focus on what
  - Project process standl.
    - ↳ focus on how.
  - lots of global customers give projects to CMMI certified s/w houses
  - ISO 9001
  - IEEE standard: focus on validation

- ② Human components (generic)
- management have diff tiers of management
    - ↳ senior manag: r budget-making policies
    - ↳ middle & lower
  - SQA units make sure 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.
  - Organisational 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 acquaintance: how well team members know each other  
less acquaintance, higher components.

---

## Lec # 7

### Integration of SQA activities

## Software Processes

- It is an approach to SW development & maintenance
- Framework where SW dev & main. activities takes place
- Also called life cycle / roadmap

Processes are classified into

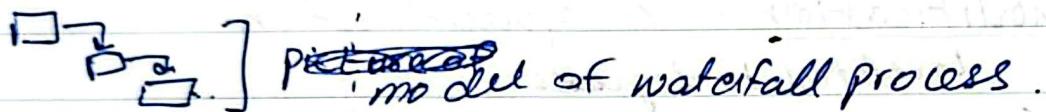
- 1) Traditional
  - 2) Agile. → change friendly
- also called conventional, heavy weight ... etc. ↳ no need to memorize.

## 1) Traditional

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

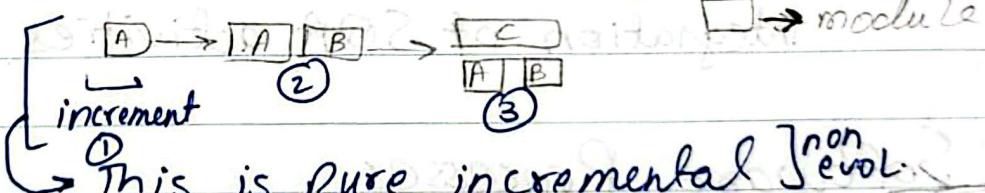
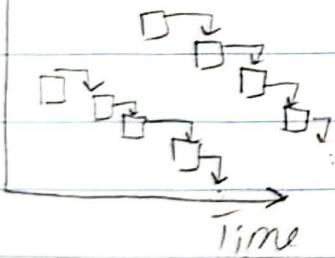
### Difference b/w process & models:

abstraction of a model.

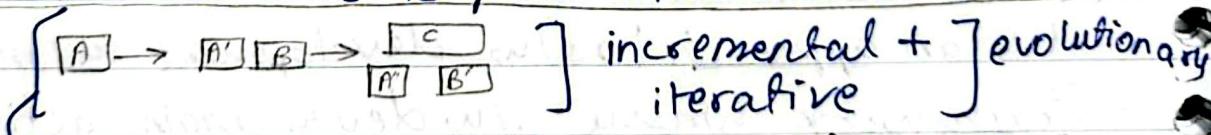


### ↳ Incremental (non evol.)

- S/w delivered earlier.
- Still following waterfall.
- Customer can start using it earlier.



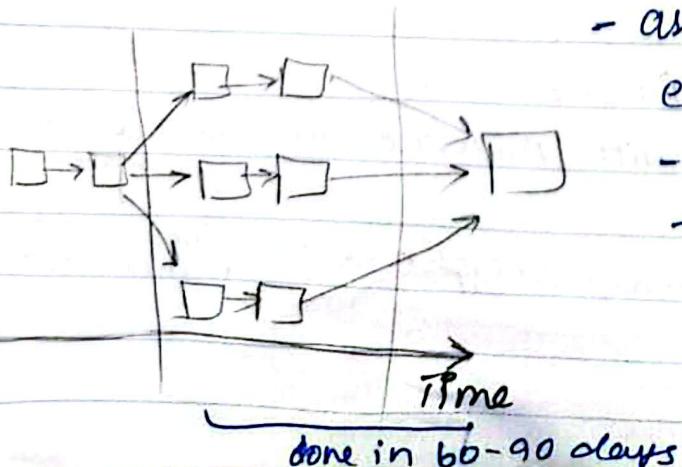
This is pure incremental [non evol.]



↳ change is happening here

### ↳ RAD. (non evol.)

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



## Q) 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  
IKIWI SF : I know it when I see it.
  - ↳ show me prototype & i'll know what I need.
- For small medium sized projects .
- nightmare for slw managers.
- Cause management related problems .
- Plans change continuously
- Customer highly involved. Probability of satisfying the customer is ~~too~~ high.
- The design is not much solid .
- Throwaway vs build upon prototypes.
  - not meant to be delivered. ↳ difficult to change later.



### ↳ Spiral

Risk analysis



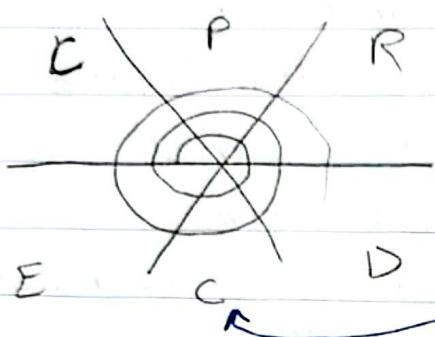
Engineering

Evaluation

- 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 customers.

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

Advanced spiral process



↳ called win win ~~process~~ process  
C = communication  
P = planning.  
D = design.  
C : construction

## 2) Agile Manifesto

- 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  $\rightarrow$  should adapt to it.

\* Agile says people are king & process should adapt.

$$RD = (POD + PD) \times 1. \frac{FE}{100}$$

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

$$TRC = RD \times CDR$$

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

\* The PD for next phase is

$$\text{Total-RD}$$

$$= 60$$

## Lec 10 Contract Review

↓  
Pre project Component

SS — SC

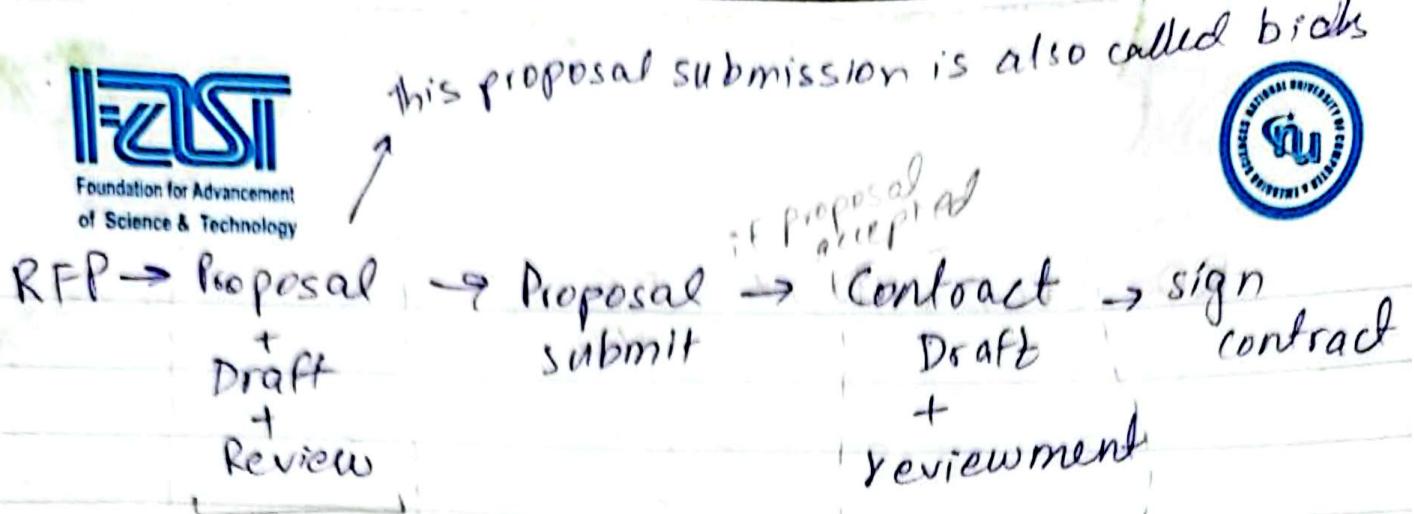
↳ focus on the

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

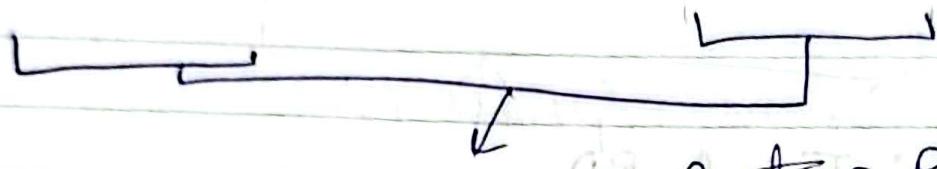
- Internal project: eg. IT dept make project of marketing dept in the same organization

RFP = request for proposal.

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



at this point SC gets multiple proposals from diff SS.



Part of contract ~~or~~ 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

↳ testers, servers, cost, schedule  
programmers

- Capacity

↳ SS & SC ki capacity is examined. Checking if SS is capable of making a sln & SC are able to pay.

- Participants

↳ identify subcontractors. How profits shared. If there's a penalty 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 Review <sup>draft</sup> ---

Contract Review: <sup>draft</sup>

- 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 runs SC

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

SS - SC  
software dept      hardware dept ] example.

- usually formal contract doesn't get signed but this can cause problems.
- Suppose req. were not classified, low customer satisfaction  
Customer requests changes - schedule slips, cost overruns, extra effort, project late.  
So idea is, make formal contract for internal projects too.