Software engineering

Computer S/W: is the product that S/W professionals build and then support over the long term. It encompasses programs, (data) content that is presented as the computer programs execute, and documents in both hardcopy and virtual forms that encompass all forms of electronic media.

* S/W is both a product and a vehicle that delivers a product.

S/W Characteristics:
- S/W is developed or engineered; (not manufactured)
- S/W does not "wear out" but it does deteriorate (better S/W design reduce deterioration)
- most S/W continues to be custom built, although the industry is moving toward component-based construction.

S/W Categories: ① System S/W ② Application S/W ③ Engineering/scientific S/W ④ embedded S/W ⑤ product-line S/W ⑥ web-applications ⑦ Artificial Intelligence S/W

unified theory for S/W evolution:
- the law of continuing change
- the law of increasing complexity
- " " " self-regulation
- " " " conservation of organizational stability
- " " " " " familiarity
- " " " continuing growth
- " " " declining quality
- the feedback system law

S/W process:
- a process defining who is doing what, when and how to reach a certain goal.
- a roadmap that helps you create a timely, high-quality result to build a product or system.
- at a detailed level, the process that you adopt depends on the S/W that you are building.
- the work products are programs, documents, and data that are produced as consequence of the activities and tasks defined by the process.

S/W engineering (a layered technology) = encompasses a process, methods, and tools with a quality focus.

* S/W process as process framework (encompasses a set of umbrella activities). These activities are framework activities (consists of actions, i.e., a collection of related tasks).

Framework activities := communication, planning, modeling, construction, deployment

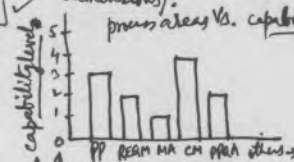
* different projects demand different task sets (the actual work to be done to accomplish the objectives of a software engineering action). The S/W team chooses the task set based on problem and project characteristics.

* umbrella activities (S/W project tracking and control, risk management, S/W quality assurance, formal technical reviews, measurement, S/W configuration management, reusability management, work product preparation and production) occur throughout the S/W process and focus primarily on project management, tracking, and control.

* **Capability Maturity Model Integration (CMMI)**: = a comprehensive process meta-model, represented in two ways: ① as a continuous model ② as a staged model (defines the same process areas against five maturity levels: performed, managed, defined, quantitatively managed, optimized))

① As a continuous model (describes a process in two dimensions).

process areas Vs. capability levels



PP = project planning
REQM = Requirement management
MA = measurement and analysis
CM = Configuration management
PPQA = Process and product QA
. . . .

Incomplete: o level
Performed: level 1
Managed: level 2
Defined: level 3
Quantitatively managed: level 4
Optimized: level 5

level 0: Incomplete (the process area is either not performed or does not achieve all goals and objectives defined by CMMI for level 1 capability)

level 1: Performed (All of the specific goals of process area have been satisfied. Work tasks required to produce defined work products are being conducted.)

level 2: Managed (All level 1 criteria + all work associated with process are conforms to an organizationally defined policy; all people doing the work access to adequate resources, stakeholders involved, all work tasks and work products are "monitored, controlled, and reviewed, evaluated.")

level 3: Defined (all level 2 criteria + process tailored, work product contributed, measures, assets, process improvement info)

level 4: Quantitatively managed (all level 3 + quantitative assessment, quantitative objectives for quality and process performance)

level 5: Optimized (all level 4 criteria + process are adapted, optimized for changing needs and improvement)

* S/w process patterns (a collection of process patterns provided with templates – a consistent method for describing an important characteristic of S/w process).

an example Process Pattern: an approach;

Pattern name. Prototyping.

Intent. (objective of the pattern is to build a model (a prototype) that can be assessed iteratively by stakeholders in an effort to identify or solidify S/w requirements)

Type. Phase pattern

Initial context. prior conditions before initiation of this pattern; ... ...

Problem. ...

Solution. ...

Resulting context. ...

Related patterns. ...

known uses/examples. Prototyping is recommended when requirements are uncertain.

* Process Assessment: approaches = Standard CMMI Assessment Method for Process Improvement (SCAMPI), CMM-Based Appraisal for Internal Process Improvement (CBA IPI), SPICE (ISO/IEC 15504), ISO 9001:2000 for S/w (plan-do-check-act) cycle

* Personal Software Process (PSP) – emphasizes personal measurement of both the work product that is produced and the resultant quality of the work product.

– PSP makes the practitioner responsible for project planning (e.g., estimating and scheduling) and empowers the practitioner to control the quality
– PSP process model defines five framework activities: planning, high-level design, high-level design review, development, postmortem
– PSP emphasizes the need to record and analyze the types of errors you make, so you can develop strategies to eliminate them.

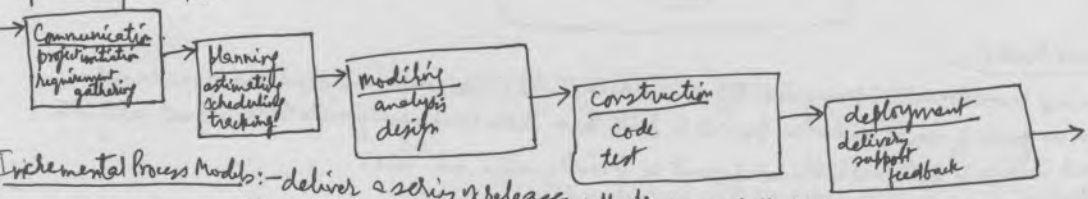
* Team Software Process (TSP) –

– the goal of TSP is to build a "self-directed" project team that organizes itself to produce high-quality software.
– to form a self-directed team, you must collaborate well internally and communicate well externally.
– TSP defines the framework activities: launch, high-level design, implementation, integration and test, and postmortem.
– TSP scripts (specific process activities, e.g., launch, design, ...) define elements of the team process and activities that occur within the process.

* Process and product are dual in nature. (The duality of process and product is one important element in keeping creative people engaged). If the process is weak, the end-product will undoubtedly suffer.

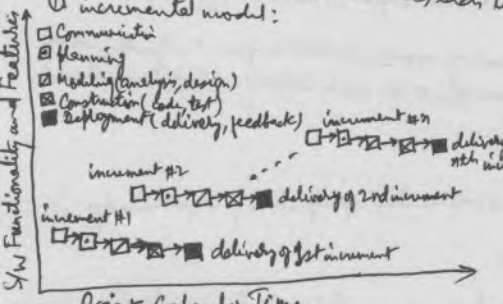
* Prescriptive process models define a distinct set of activities, actions, tasks, milestones, and work products that are required to engineer high quality software. These process models are not perfect but the process guides a software team through a set of framework activities that are organized into a process flow that may be linear, incremental, or evolutionary.
* a prescriptive process model populates a process framework with explicit task sets for S/W engineering actions.
* generic process framework encompasses the framework activities: communication, planning, modeling, construction, deployment.
* even though a process is prescriptive, do not assume that it is static. Prescriptive models should be adapted to the people, to the problem, to the people.

The waterfall model (classic life cycle) = a systematic, sequential approach to S/W development that begins with customer specification of requirements and progresses through planning, modeling, construction, deployment, culminating in ongoing support of the completed S/W.



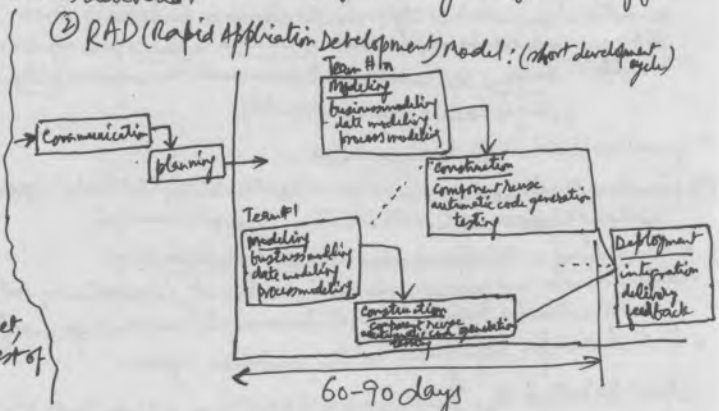
Incremental Process Models: - deliver a series of releases, called increments that provide progressively more functionality for the customer as each increment is delivered.

① incremental model:



* if your customer demands delivery by a date that is impossible to meet, suggest delivering one or more increments by that date and the rest of the S/W (additional increments) later.

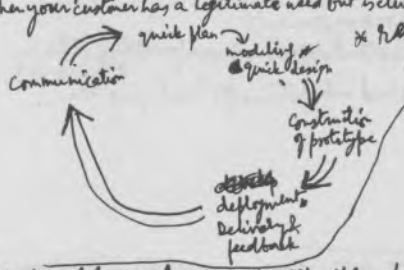
② RAD (Rapid Application Development) Model: (short development cycle)



Evolutionary Process models: - produce an increasingly more complete version of the S/W with each iteration.

① prototyping model:

* when your customer has a legitimate need but is clueless about the details, develop a prototype as a first step.
* resist pressure to extend a rough prototype into a production product. Quality almost always suffers as a result.



② spiral model - can be adapted to apply throughout the entire life cycle of an application, from concept development to maintenance.

* if your management demands fixed-budget development (generally a bad idea), the spiral can be a problem: as each circuit is completed, project cost is revisited and revised.

(3) The Concurrent Development model: (Concurrent engineering model) * is often more appropriate for system engineering projects where (4)
(different engineering teams are involved.)

* can be represented schematically as a series of framework activities, S/W engg actions and tasks, and their associated states.

represents the state (externally observable mode of behavior) of a S/W engg activity or task

modeling activity (one element of the concurrent process model)

Specialized Process Models :-

① Component-Based Development: (Commercial Off-the-shelf (COTS)) S/W components developed by vendors who offer them as products, can be used when S/W is to be built. These components provide targeted functionality with well-defined interfaces that enable the component to be integrated into the S/W.

* evolutionary, iterative model, composes application from prepackaged S/W components.
* modeling and construction activities begin with the identification of candidate components. These components can be designed as either conventional S/W modules or Object-oriented classes or packages of classes.

steps are: component-based products are researched, evaluated for the app. domain in question | considered about component integration issues, designed S/W architecture to accommodate the component | integrated components into the architecture | conducted comprehensive testing to ensure proper functionality
(for the model)

* special emphasis on S/W reuse

② Formal methods model :- encompasses a set of activities that leads to formal mathematical specifications of computer S/W by applying rigorous, mathematical notation, mathematical analysis.

* a variation on this approach, called cleanroom S/W engineering.
* discover and correct more easily ambiguity, incompleteness, and inconsistency, errors but quite time-consuming, expensive, need extensive training, difficult communication mechanism for unsophisticated customers.
* best choice for safety-critical S/W, real-time systems

Aspect-Oriented S/W Development (AOSD) - defines "aspects" that express customer concerns that cut across multiple system functions, features, and information.

* some concerns are high-level properties of a system (e.g., security, fault tolerance), other concerns affect functions (e.g., application of business rules), while others are systemic (e.g., task synchronization or memory management)
* also referred as Aspect-Oriented Programming (AOP), provides a process and methodological approach for defining, specifying, designing and constructing aspects - "mechanisms beyond subroutines and inheritance for localizing the expression of a crosscutting concern"
* evolutionary, parallel nature (concurrent development).

Process management tools :- ① GDPA, a research process definition tool suite, Bremen university in Germany
② SpeeDev, SpeeDev Corp.
③ Step gate Process, - .

The Unified Process: – a framework for object-oriented S/W engg. using UML.

* S/W process should be use-case driven, architecture-centric, iterative and incremental

Phases of the Unified Process: – * unified process phases are similar in intent to the generic framework activities

= inception phase of UP encompasses both customer communication and planning activities.
identify business requirements, propose rough architecture, develop plan for iterative, incremental nature of the ensuring project are basic goals.
(business requirements are described by use-cases), the rough architecture is later refined and expanded into a set of models (representing different views), planning identifies resources, assesses major risks, defines a schedule, and establishes a basis for the phases (to be applied as S/W increment is developed).

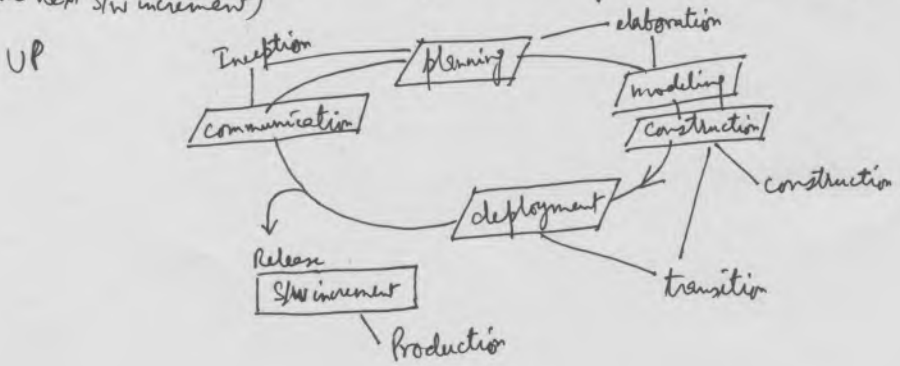
= elaboration phase encompasses the customer communication and modeling activities of generic process model.
* refines and expand the preliminary use-cases, expand the architectural representation to include five different views of the S/W * (use-case model, analysis model, design model, implementation model, deployment model). In some cases, elaboration creates an "executable architectural baseline" that represents a "first cut" executable system. The plan is carefully reviewed at the culmination of the elaboration phase to ensure that scope, risks and delivery dates remain reasonable.

= construction phase of UP is identical to the construction activity defined for the generic S/W process.
develop or acquire S/W components (making each use-case operational for end-users) by completing analysis and design models, implement all necessary and required features and functions of the S/W increment (i.e., the release) in source code, design and execute for each component unit tests, conduct integration activities (component assembly and integration testing); Use-cases are used to derive a suite of acceptance tests that are executed prior to the initiation of next UP phase.

= transition phase of UP encompasses the latter stages of the generic construction activity and the first part of the generic deployment activity. S/W is given to end-users for beta testing, and user feedback reports both defects and necessary changes. S/W team creates the necessary support information (e.g., user manuals, trouble-shooting guide, and installation procedures) that is required for the release. Finally, S/W increment becomes a usable S/W release.

= Production phase of the UP coincides with the deployment activity of the generic process.
monitor on-going use of the S/W, support provided for operating environment (infrastructure), Submit and evaluate defect reports and requests for changes,

* UP phases do not occur in a sequence but rather with staggered concurrency (at the same time the construction, transition, and production phases are being conducted, work may have already begun on the next S/W increment)



Major work products produced for each UP phase

inception phase

Vision document
initial use-case model
Initial project glossary
Initial business case
Initial risk assessment
Project plan
phases and iterations
Business model
if necessary
One or more prototypes

Elaboration phase

use case model
Supplementary
requirements
including non-functional
Analysis model
S/w architecture
description
Executable architectural
prototype
Preliminary design model
Revised risk list
Project plan including
iteration plan
adapted work flows
milestones
technical work products
Preliminary user manual

Construction phase

Design model
S/w components
Integrated S/w
increment
Test plan and procedure
Test cases
Support documentation
user manuals
installation manuals
description of current
increment

Transition phase

Delivered S/w
increment
Beta test reports
General user feedback

An Agile view of Process ⇒ (Agile method / light method / lean method) –

* Agile S/w engg combines a philosophy and a set of development guidelines. The philosophy encourages customer satisfaction and early incremental delivery of S/w; small, highly motivated project teams; informal methods; minimal S/w engg work products; and overall development simplicity. The development guidelines stress delivery over analysis and design (although these activities are not discouraged), and active and continuous communication between developers and customers.
* S/w engineers and other projects stakeholders (managers, customers, end-users) work together on an agile team – a team that is self-organized and in control of its own destiny. An agile team fosters communication and collaboration among all who serve on it.
* Agile development might best be termed "S/w engg lite". The basic framework activities – customer communication, planning, modeling, construction, delivery, evaluation – remain. But they morph into a minimal task set that pushes the project team towards construction and delivery (some would argue that this is done at the expense of problem analysis and solution design).
* the only really important work product is an operational "S/w increment" that is delivered to the customer on the appropriate commitment date.

Principles to achieve agility:

1. our highest priority is to satisfy the customer through early and continuous delivery of valuable S/w.
2. welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within development team is face-to-face conversation.
7. working S/w is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. continuous attention to technical excellence and good design enhances agility.
10. simplicity – the art of maximizing the amount of work not done – is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular interval, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

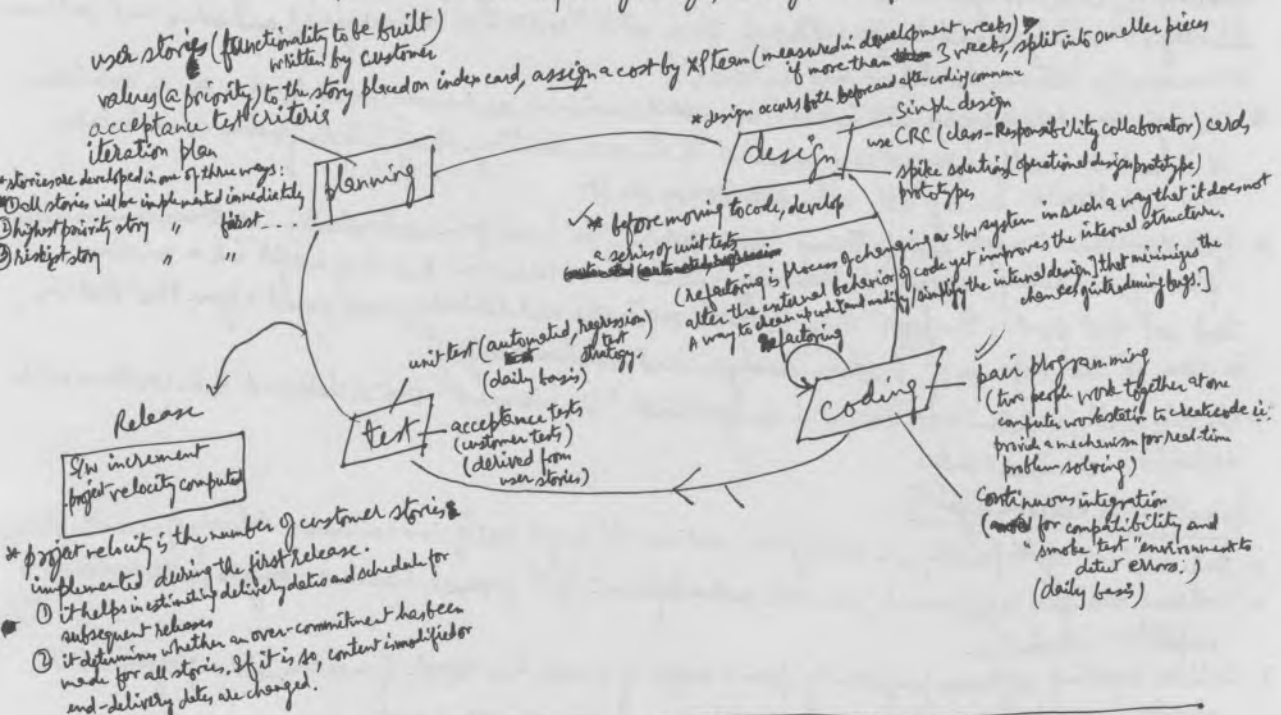
Agile Process :=
* an incremental development strategy should be instituted. S/w increments (executable prototypes or a portion of an operational system) must be delivered in short time periods)
* define a S/w engg approach that is agile.
* the process molds to the needs of the people and team.

key traits of agile team :– Competence, common focus, collaboration, decision-making ability, Fuzzy problem-solving ability, mutual trust and respect, self-organization

* a self-organizing team is in control of the work it performs. The team makes its own commitments and defines plans to achieve them.

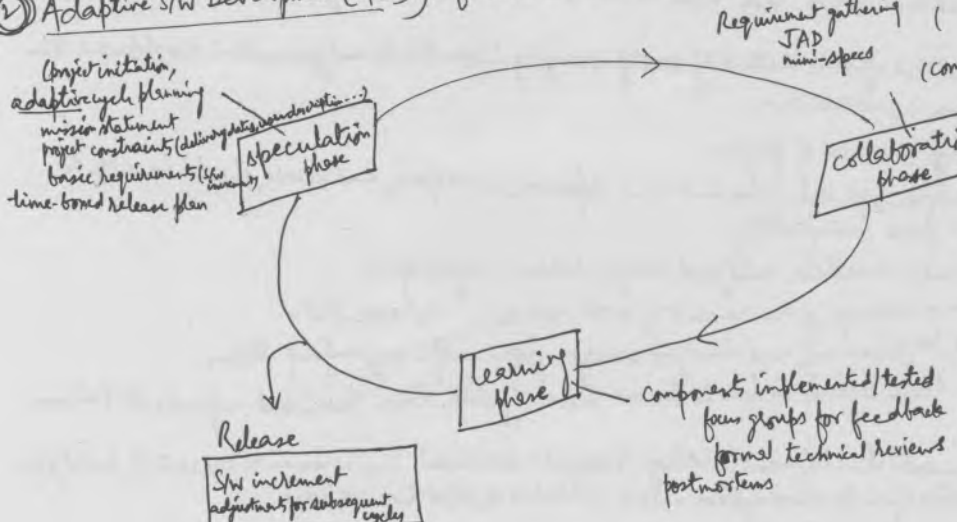
Agile Process Models :-

① Extreme Programming (XP) – uses an object-oriented approach as its preferred development paradigm.
– encompasses a set of rules and practices that occur within the context of four framework activities: planning, design, coding, testing



* project velocity is the number of customer stories implemented during the first release.
① it helps in estimating delivery dates and schedule for subsequent releases
② it determines whether an over-commitment has been made for all stories. If it is so, content is modified or end-delivery dates are changed.

② Adaptive SW Development (ASD) = focus on human collaboration and team self-organization.



(3) Dynamic Systems Development Method (DSDM) = (provides a framework for building and maintaining systems which meet tight time constraints through the use of incremental prototyping in a controlled project environment)

— follows modified Pareto principle (80% of an application can be delivered in 20% of the time it would take to deliver the complete (100%) application)

i.e. only enough work is required for each increment to facilitate movement to the next increment. The remaining detail can be completed later when more business requirements are known or changes have been requested and accommodated.

— DSDM life cycle (3 iterative cycles, preceded by 2 additional life cycle activities)

feasibility study – (business requirements and constraints)

business study – (functional and information requirements to provide business value, app. architecture and maintainability requirements)

Functional model iteration (produces a set of incremental prototypes, gather additional requirements by feedback)

Design and build iteration (revisits prototypes ...)

Implementation – (places latest S/w increment (an "operationalized" prototype) into operational environment)

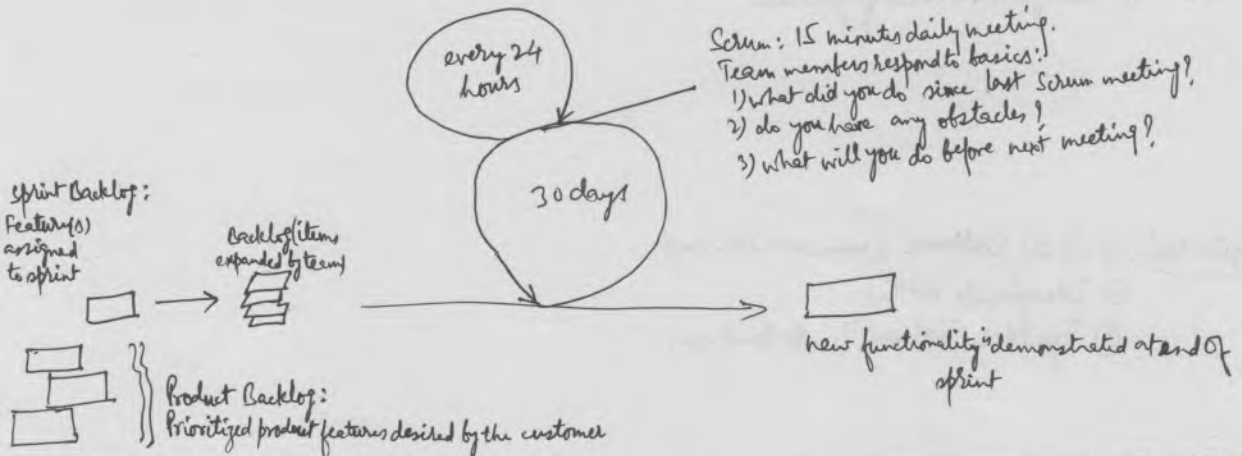
— DSDM can be combined with XP and ASP concepts

(4) Scrum: (principles are consistent with the agile manifesto):

- small working teams (maximum communication & sharing of tacit, informal knowledge, min. overhead)
- adaptable process to both technical & business changes
- frequent S/w increments by using process
- development work and people are partitioned "into clear, low coupling partitions, or packets."
- constant testing and documentation
- provides the "ability to declare a product 'done' whenever required.

* Within each framework activity (requirement, analysis, design, evolution, delivery), work tasks occur within a process pattern called a sprint. The work conducted within a sprint (the number of sprints required for each framework activity will vary depending on product complexity and size) is adapted to the problem at hand and is defined and often modified in real-time by the Scrum team.

* Scrum emphasizes the use of a set of "S/w process patterns" that have proven effective for projects with tight timelines, changing requirements, and business criticality.
  ↓
  (project priorities, compartmentalized work units, communication, and frequent customer feedback)



(5) Crystal / Crystal family of agile methods: (a set of agile processes that have been proven effective for different types of projects. The intent is to allow agile teams to select the member of the crystal family that is most appropriate for their project and environment.)

(6) Feature Driven Development (FDD) — an adaptive, agile process, can be applied to moderately sized and larger S/w projects.

— a feature is a client-valued function (small blocks, hierarchical) that can be implemented in two weeks or less.

— template for defining a feature:

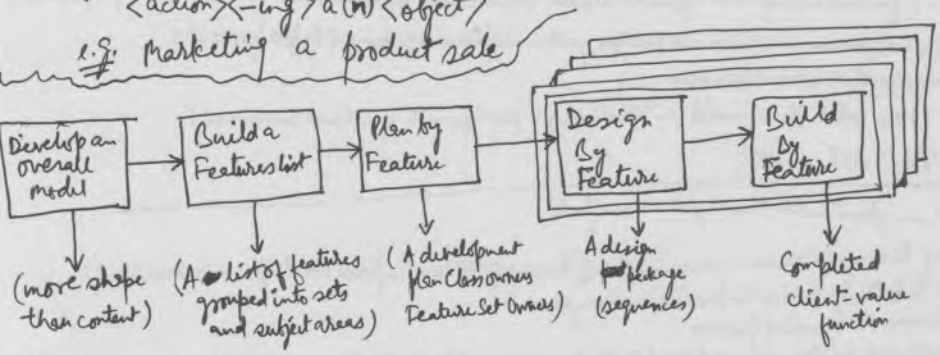
<action> the <result> <by | for | of | to> a(n) <object> → a person, place, or thing (including roles, moments in time or intervals of time, or catalog-entry-like descriptions)

e.g. Add the product to a shopping cart.

— A feature set groups related features into business-related categories and is defined as:

<action> <-ing> a(n) <object>

e.g. marketing a product sale



* greater emphasis on project management guidelines and techniques

five collaborating framework activities (processes)

with 6 milestones:
- design walkthrough
- design
- design inspection
- code
- code inspection
- promote to build

(7) Agile Modeling (AM) = a collection of values, principles, practices for modeling S/w that can be applied on a S/w development project in an effective and light-weight manner.

* courage and humility are two essential properties.

modeling principles :-

(i) model with purpose (goal).
(ii) use multiple models.
(iii) Travel light (Build only those models that provide long-term value — no more, no less)
(iv) content is more important than representation (useful content (information) is essential)
(v) know the models and tools you use to create them
(vi) Adapt locally (to the needs of agile team)

Agile tool : ① Actif Extreme (www.microtool.com)
② Ideogramic UML (
③ Together Tool set (borland.com)

unit testing frame work for .Net → NUnit

* S/W engg practices : Core principles -

(1) The reason it all exists (to provide value to its users)
(2) KISS (Keep it Simple, Stupid!)
(3) Maintain the vision (clear vision)
(4) What you produce, others will consume (always specify, design, implement knowing someone else will have to understand what you are doing)
(5) Be open to the future (a system with long lifetime) (generalize) (also, S/W must be built to be maintainable)
(6) Plan ahead for reuse (
(7) Think (placing clear, complete thought before action almost always produce better results)

* Communication Practices :- (before communicating be sure you understand the point of view of other party, know a bit about his/her needs)

(1) Listen
(2) Prepare before you communicate
(3) someone should facilitate the activity
(4) Face-to-Face communication is best
(5) take notes and document decisions
(6) strive for collaboration
(7) stay focused, modularize (leaving one topic only after it has been resolved) your discussion
(8) If something is unclear, draw a picture
(9) (a) once you agree to something, move on (b) if you can't agree to something, move on (c) if a feature or function is unclear and cannot be clarified at the moment, move on.
(10) Negotiation is not a contest or a game. It works best when both parties win (negotiation will demand compromise from all parties).

task set :

(1) identify primary customer and other stakeholders
(2) meet with primary customer to address "context free questions" that define
   - Business needs and business values
   - end-users' characteristics/needs
   - Required user-visible outputs
   - Business constraints
(3) develop one-page written statement of project scope that is subject to revision
(4) Review statement of scope with stakeholders and amend as required.
(5) Collaborate with customer/end-users to define:
   - Customer visible usage scenarios using standard format
     * resulting outputs and inputs
     * important S/W features, functions and behaviors
     * Customer-defined business risks
(6) Develop a brief written description (e.g., a set of lists) of scenarios, outputs/inputs, features/functions and risks
(7) iterate with customer to refine scenarios, outputs/inputs, features/functions, and risks.
(8) Assign customer-defined priorities to each user scenario, feature, function, and behavior.
(9) Review all information gathered during the communication activity with the customer and other stakeholders and amend as required.
(10) prepare for planning activity

Planning practices:

1. understand the scope of the project
2. involve the customer in the planning activity
3. Recognize that planning is iterative
4. Estimate based on what you know
5. Consider risk as you define the plan
6. Be realistic
7. Adjust granularity (level of detail) as you define the plan
8. define how you intend to ensure quality
9. Describe how you intend to ~~communicate~~ accommodate change
10. track the plan frequently and make adjustments as required.

W^{5}HH principle:

why is the system being developed?
what will be done?
when will it be accomplished?
who is responsible for a function?
where are they organizationally located?
How will the job be done technically and managerially?
How much of each resource is needed?

Task set for planning:

1. recalculate project scope
2. Assess risks
3. Develop and/or refine user scenarios
4. extract functions and features from the scenarios
5. define technical functions and features that enable S/W infrastructure.
6. Group functions and features (scenarios) by customer priority
7. create a coarse granularity (level of detail) project plan.
   - define number of projected S/W increments
   - establish an overall project schedule
   - establish projected delivery date for each increment
8. Create a fine granularity (level of detail) plan for the current iteration.
   - define work tasks for each function feature
   - estimate effort for each work task
   - Assign responsibility for each work task
   - define work products to be produced
   - identify quality assurance methods to be used
   - describe methods for managing change
9. Track progress regularly
   - note problem areas (e.g., schedule slippage)
   - make adjustments as required

Modeling practice: [Analysis model = customer requirements, and
Design model = concrete specification for the construction of software]

1. information domain of the problem must be represented and understood.
2. the functions that the S/W performs must be defined,
3. the behavior of S/W (as a consequence of external events) must be represented.
4. the models that depict information, function, and behavior must be partitioned in a manner that uncovers details in a layered (or hierarchical) fashion
5. The analysis task should move from essential information towards implementation detail

Analysis modeling task sets:

1. Review business requirements, end-users' characteristics/needs, user-visible outputs, business constraints, and other technical requirements that were determined during the customer communication and planning activities.
2. Expand and refine user scenarios.
   - define all actors
   - Represent how actors interact with the S/W
   - extract functions and features from the user scenarios.
   - Review the user scenarios for completeness and accuracy
3. Model the information domain.
   - Represent all major information objects
   - define attributes for each information object
   - Represent the relationships between information objects
4. model the functional domain.
   - Show how functions modify data objects
   - Refine functions to provide elaborative details.
   - Write a processing narrative that describes each function and subfunction
   - Review the functional model
5. Model the behavioral domain.
   - identify external events that cause behavioral changes within the system
   - identify states that represent each externally observable mode of behavior
   - depict how an event causes the system to move from one state to another.
   - review the behavioral models.
6. Analyze and model the user interface.
   - Conduct task analysis.
   - Create screen image prototypes
7. Review all models for completeness, consistency and omissions.

---

design modeling:

1. design should be traceable to the analysis model
2. Always consider the architecture of the system to be built.
3. design of data is as important as design of processing functions.
4. Interfaces (both internal and external) must be designed with care
5. user interface design should be tuned to the needs of the end-user.
6. Component-level design should be functionally independent.
7. Components should be loosely coupled to one another and to the external environment.
8. Design representations (models) should be easily understandable
9. The design should be developed iteratively. With each iteration, the designer should strive for greater simplicity.

Agile modeling info:

principles:
1. The primary goal of the software team is to build S/W, not create models.
2. Travel light – don't create more models than you need
3. Strive to produce the simplest model that will describe the problem or the S/W
4. Build models in a way that makes them amenable to change
5. Be able to state an explicit purpose for each model that is created
6. Adapt the models you develop to the system at hand
7. Try to build useful models, but forget about building perfect models
8. Don't become dogmatic about the syntax of the model. If it communicates content successfully, representation is secondary.
9. If your instincts tell you, a model is not right even though it seems okay on paper, you probably have reason to be concerned.
10. Get feedback as soon as you can.

Design task set:-

1. using analysis model, select an architectural style (pattern) that is appropriate for S/W
2. Partition the analysis model into design subsystems and allocate these subsystems within the architecture
   - Be certain that each subsystem is functionally cohesive.
   - design subsystem interfaces.
   - Allocate analysis classes or functions to each subsystem.
   - Using the information domain model, design appropriate data structures
3. design the user interface.
   - Review results of task analysis.
   - Specify action sequence based on user scenarios.
   - Create behavioral model of the interface.
   - Define interface objects, control mechanisms.
   - Review the interface design and revise as required
4. Conduct component-level design.
   - Specify all algorithms at a relatively low level of abstraction.
   - Refine the interface of each component.
   - Define component level data structures.
   - Review the component level design
5. Develop a deployment model

Construction practice := (coding and testing principles)

Construction task set:

1. Build architectural infrastructure.
   - Review the architectural design.
   - Code and test the components that enable architectural infrastructure.
   - Acquire reusable architectural patterns.
   - Test the infrastructure to ensure interface integrity.
2. Build a S/W component.
   - Review the component-level design.
   - Create a set of unit tests for the component
   - Code component data structures and interface.
   - code internal algorithms and related processing functions.
   - Review code as it is written.
   - Look for correctness.
   - Ensure that coding standards have been maintained.
   - Ensure that the code is self-documenting.
3. Unit test the component.
   - Conduct all unit tests.
   - Correct all errors uncovered. Reapply unit tests.
4. integrate completed component into the architectural infrastructure.

Testing principles:

1. all tests should be traceable to customer requirements. (functional tests = focus on requirements)
2. tests should be planned long before testing begins.
3. The Pareto principle applies to S/W testing. (80% of all errors uncovered during testing will likely be traceable to 20% of all program components.)
4. testing should begin "in the small" and progress toward testing "in the large." (components → S/W architecture)
5. exhaustive testing is not possible.

Testing task set:

1. design unit tests for each S/W component.
   - Review each unit test to ensure proper coverage.
   - Conduct the unit test.
   - Correct errors uncovered.
   - Reapply unit tests.
2. develop an integration strategy.
   - Establish order of and strategy to be used for integration.
   - Define "builds" and the tests required to exercise them.
   - Conduct smoke testing on a daily basis.
   - Conduct regression tests as required.
3. Develop validation strategy.
   - Establish validation criteria.
   - Define tests required to validate S/W.
4. Conduct integration and validation tests.
   - Correct errors uncovered.
   - Reapply tests as required.
5. Conduct high-order tests.
   - Perform recovery testing
   - Perform security testing
   - Perform stress testing
   - Perform performance testing
6. Coordinate acceptance tests with customer

Deployment practices:=

1. Customer expectations for the S/W must be managed. (Be sure that your customer knows what to expect before a S/W increment is delivered.)
2. A complete delivery package should be assembled and tested.
3. A support regime must be established before the S/W is delivered.
4. Appropriate instructional materials must be provided to end-users.
5. Buggy S/W should be fixed first, delivered later.

Deployment task set:

1. create delivery media.
   - Assemble and test all executable files.
   - Assemble and test all data files.
   - Create and test all user documentation.
   - Implement electronic (e.g., pdf) versions.
   - Implement a troubleshooting guide.
   - Test delivery media with a small group of representative users.
2. Establish human support person or group.
   - Create documentation and/or computer support tools.
   - Establish contact mechanisms (e.g., website, phone, e-mail).
   - Establish problem-logging mechanisms.
   - Establish problem-reporting mechanisms.
   - Establish problem/error " DB.
3. Establish user feedback mechanisms.
   - define feedback process.
   - " " forms (paper and electronic).
   - Establish " DB.
   - Define " assessment process.
4. Disseminate delivery media to all users.
5. Conduct on-going support functions.
   - Provide installation and start-up assistance.
   - " continuing troubleshooting assistance.
6. Collect user feedback.
   - Log feedback.
   - Assess feedback.
   - Communicate with users on feedback.

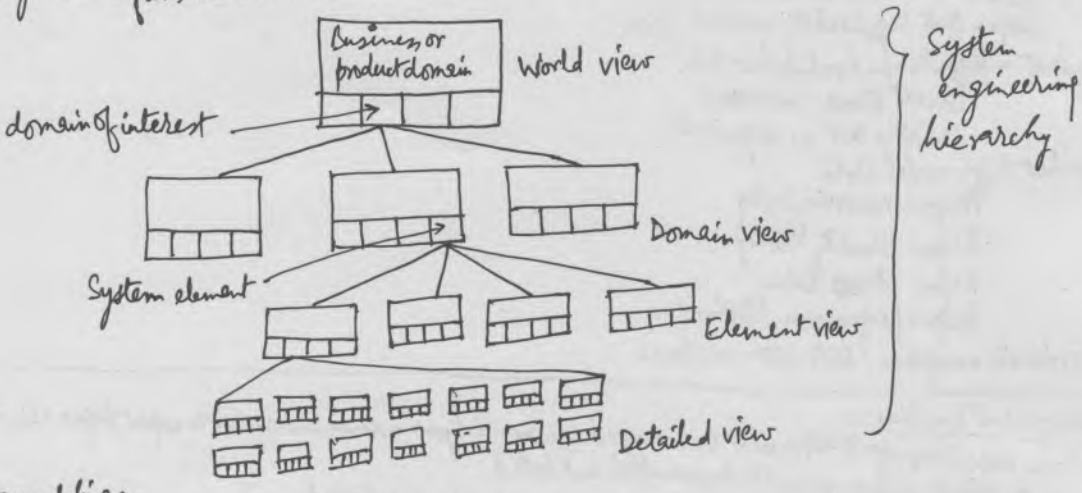
System Engineering: steps:

Objectives and more detailed operational requirements are ~~determined~~ identified by eliciting information from the customer; requirements are analyzed to assess their clarity, completeness, and consistency; a specification, often incorporating a system model, is created and then validated by both practitioners and customers. Finally, system requirements are managed to ensure that changes are properly controlled.

work product of System engineering :-

An effective representation of the system must be produced as a consequence of system engineering. This can be a prototype, a specification or even a symbolic model, but it must communicate the operational, functional, and behavioral characteristics of the system to be built and provided insight into the system architecture.

* Complex systems are actually a hierarchy of macro elements that are themselves systems.
* good system engineering begins with a clear understanding of context — the world view — and then progressively narrows focus until technical detail is understood.



System modeling :=

- define the processes that serve the needs of the view under consideration.
- Represent the behavior of the processes and the assumptions on which the behavior is based.
- Explicitly define both exogenous (link one constituent of a given view with other constituents at the same level or other levels) and endogenous (links individuals components of a constituent at a particular view) inputs to the model.
- Represent all linkages (including output) that will enable the engineer to better understand the view.

* A system engineer considers the following factors when determining alternative solutions: assumptions, simplifications, limitations, constraints, and customer preferences.

system simulation :-

* If simulation capability is unavailable for a reactive system, project risks increases. Consider using an incremental process model that will enable you to deliver a working product in the first iteration and then use other iterations to tune performance.

Business Process Engineering - to define architectures that will enable a business to use information effectively. - data architecture, Application architecture, Technology infrastructure are analyzed and designed.

* you may never get involved in ISP or BAA. However, if it is clear that these activities have not been done, inform stakeholders that project risk is very high.

Product engineering :- to translate the customer's desire for a set of defined capabilities into a working product. - architecture [S/w, hardware, data (and database), people] components and infrastructure [technology to tie components, information (documents, CD-Rom, video, ...) to support] must be derived.

✓ Requirements Engineering :-

- helps S/w engineers to better understand the problem they will work to solve. It encompasses the set of tasks that lead to an understanding of what the business impact of the S/w will be, what the customer wants, and how end-users will interact with the software.

Requirement engineering steps :-

- Requirements engineering begins with inception - a task that defines the scope and nature of the problem to be solved. It moves onward to elicitation - a task that helps the customer to define what is required, and then elaboration - where basic requirements are refined and modified. As the customer defines the problem, negotiation occurs - what are the priorities, what is essential, when is it required? Finally, the problem is specified in some manner and then reviewed or validated to ensure that your understanding of the problem and the customer's understanding of the problem coincide.

Work product of requirements engineering : user scenarios, functions and features lists, analysis models, or a specification

* Requirements engineering work products are reviewed with the customer and end-users to ensure that what you have learned is what they really meant.

** things may change throughout the project. (requirement change)

* Requirements engg. establishes a solid base for design and construction. Without it, the resulting S/w has a high probability of not meeting customer's needs.

* expect to do a bit of design during requirements work and a bit of requirements work during design.

Requirement engg. tasks :-

(1) inception
(2) elicitation (problem of scope, problem of understanding, problem of volatility)
(3) elaboration (elaboration is good thing, but you have to know when to stop. The key is to describe the problem in a way that establishes a firm base for design. If you work far beyond that point, you are doing design.)
(4) negotiation (there should be no winner and no loser in an effective negotiation. Both sides win because a deal that both can live with is solidified)
(5) specification (the formality and format of a specification varies with the size and the complexity of the S/w to be built.)
(6) validation (A key concern during requirement validation is consistency. Use analysis model to ensure that requirements have been consistently stated.)

~~increasing flexibility in ordering of work products due to tight timeline~~ poor

⑦ Requirement management (features traceability table, source traceability table, dependency traceability table, subsystem traceability table, interface traceability table)

*When a system is large and complex, determining the connections between requirements can be a daunting task. Use traceability tables to make the job a bit easier.)

* Requirement engg. software tools: (EasyRM, OnYourMarkPro, Rational RequisitePro, RTM, . .)

Initiating The Requirements engg. Process :-

① identify the stakeholders
② Recognizing multiple viewpoints
③ working toward collaboration (may be used a voting scheme based on priority points during conflict)
④ asking the first questions

Eliciting Requirements :-

① collaborative requirements gathering
* (if a system or product will serve many users, be absolutely certain that requirements are elicited from a representative cross-section of users.)
* (Avoid the impulse to shoot down a customer's idea as "too costly" or "impractical". Keep an open mind.)
② Quality Function Deployment (QFD = defines requirements in a way that maximizes customer satisfaction.)
- normal requirements
- expected requirements
- exciting requirements

③ user scenarios
④ elicitation work products

Developing USE-CASES :- (defined from an actor's point of view. An actor is a role that people (users) or devices play as they interact with the software.)

* each use-case can be assessed by stakeholders, and the relative priority for each can be assigned.

Software tools :- Clear Requirement workbench, Objects by Design

Building the Analysis Model :-

① elements of the Analysis Model
② -Scenario-based elements -(
* it is good idea to get stakeholders involved.
- Class-based elements
- Behavioral elements (a state is an externally observable mode of behavior.)
- flow-oriented element

③ Analysis patterns

Negotiating Requirements :- (① win win strategy, ② map out strategy and decide what you would like to achieve and what others, ③ make both of them happen,
④ listen actively
⑤ Focus on other party's interests
⑥ be positive and open mind (focus on problem)
⑦ be creative
⑧ be ready to commit. once an agreement has been reached, commit to it and move on.

Validating Requirements :-

Building the Analysis Model :=

- Analysis modeling uses a combination of text and diagrammatic forms to depict requirements for data, function, behaviors in a way that is easy to understand, and straightforward to review for correctness, completeness, consistency.

steps:- once preliminary models (like scenario-based, flow-oriented, class-based, or behavioral model) are created, they are refined and analyzed to assess their clarity, completeness, consistency. The final model is then validated by all stakeholders.

work product: diagrammatic forms (representations)

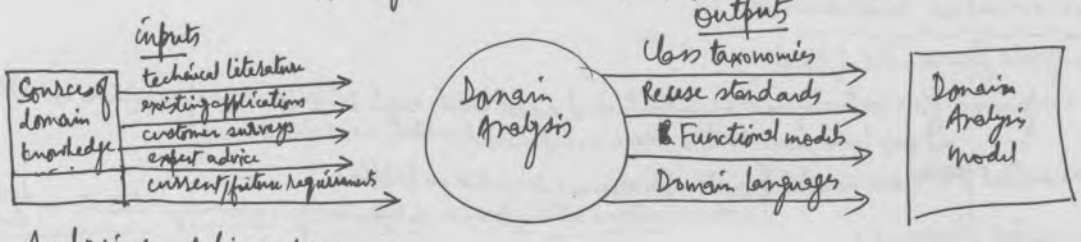
Requirements Analysis:-

* the analysis model and requirement specification provide a means for assessing quality once the SW is built.

① Analysis Rules of Thumb:- ✓

- model should focus on requirements that are visible within the problem or business domain. The level of abstraction should be relatively high (less granularity / less complexity).
- each element of the analysis model should add to an overall understanding of SW requirements and provide insight into the information domain, function, behavior of system.
- delay consideration of infrastructure and other non-functional models until design.
- minimize coupling throughout the system. (.... (refactoring)
- Be certain that the analysis model provides values to all stakeholders.
- keep the model as simple as it can be.

② domain Analysis - (identification, analysis, specification of common requirements from a specific application domain, typically for reuse on multiple projects within that application domain.)



Analysis Modeling Approaches:-

- structured analysis = data objects are modeled in a way that defines their attributes and relationships. Processes that manipulate data objects are modeled in a manner that shows how they transform data as data objects flow through the system.
- object-oriented analysis = focuses on the definition of classes and the manner in which they collaborate with one another to effect customer requirements.

Elements of analysis model:-

- Scenario-based elements = use-cases, use-case diagram, activity diagram, swimlane diagram
- Flow-oriented elements = data flow diagrams, control-flow diagram, processing narratives.
- Class-based elements = class diagram, analysis package, CRC model, collaboration diagrams.
- Behavioral elements = state diagram, sequence diagram

Data Modeling Concepts –

① data objects = (representation of any composite information that is processed by software)
② data attributes = (attributes name a data object, describe its characteristics, and, in some cases, make reference to another object)
③ Relationship = (indicate the manner in which data objects are connected to one another)
④ Cadinality and modality = (

Software Tools :- AllFusion ERWin, ER/Studio, Oracle Designer, MetaScope, ModelSphere, Visible Analyst

Object-oriented Analysis = (define all classes, relationships, behaviors associated with them, relevant to the problem to be solved.)

① basic user requirements must be communicated between the customer and software engineer.
② Classes must be identified (i.e., attributes and methods are defined)
③ Class hierarchy is defined.
④ object-to-object relationships (object connections) should be represented.
⑤ object behavior must be modeled.
⑥ Task 1 through 5 are reapplied iteratively until the model is complete.

Scenario-based Modeling :=

① writing use-cases
② developing activity diagrams (represents actions and decisions)
③ Swimlane diagram (represents flow of actions and decisions)

Flow-oriented modeling =

① creating data flow model (DFD) :-
* information flow continuity must be maintained as each DFD level is refined. (input and output at one level must be the same as input and output at a refined level)
* grammatical parses are useful. { verbs = processes = represented as bubbles, nouns = external entities (boxes) or data/control objects (arrows) or data stores (double lines) }

② creating control flow model –
③ control specification (CSPEC) – behavior of the system (at the level from which it has been referenced) in two different ways.
④ process specification (PSPEC) – "mini specification" for each transform at the lowest refined level of a DFD.

Class-based modeling :–
① identifying analysis classes (selection characteristics are retained info., needed services, multiple attributes, essential requirements, common operation, common attributes)
② specifying attributes (are the set of data objects that fully define the class within the context of the problem)
③ defining operations (focus on problem-oriented behavior)
④ Class-Responsibility-Collaborator (CRC) modeling – (create index cards)

Class : name Description :	
Responsibility :	Collaborator :
. .	. .

* allocating responsibilities to classes by (stating generally as possible, related info. and behavior residing within same class, (refactoring)
information about one thing should be localized with a single class.)
* if a class cannot fulfill all of its obligations itself, then a collaboration is needed.

(5) association and dependencies – (association defines a relationship between classes. Multiplicity defines how many of one class are related to how many of another class)

(6) Analysis package (a package is used to assemble a collection of related classes)

Creating a Behavioral Model =

(1) evaluate all use-cases to fully understand the sequence of interaction within the system.

(2) identify events that drive the interaction sequence and understand how these events related to specific classes.

* use-cases are parsed to define events. To accomplish this, the use-case is examined for points of information exchange.

(3) create sequence for each use-case

(4) Build a state diagram for the system

(5) Review the behavioral model to verify accuracy and consistency.

* Design Engineering = a representation or model of S/W, provides detail about S/W data structures, architecture, interfaces, components

steps :-
- the architecture of the system or product must be represented. (architectural design)
- the interfaces that connect the S/W to end-users, to other systems and devices, and to its own constituent components are modeled. (interface design)
- the S/W components, that are used to construct the system are designed. (Data/class design)

* design engineering should always begin with a consideration of data – the foundation for all other elements of the design.

* Assessing design Quality by Formal Technical Review

good design :-
- must implement all of the explicit requirements contained in the analysis model, and must accommodate all of the implicit requirements desired by the customer.
- the design must be a readable, understandable guide for those who generate code and for those who test and subsequently support the S/W.
- design should be provide a complete picture of the S/W, addressing the data, functional, and behavioral domains from an implementation perspective.

Quality guidelines :- exhibit recognizable architectural styles or patterns, components of good design characteristics, facilitation for implementation and testing
- modular (logically partitioned into elements or subsystems),
- distinct representation of data, architecture, interfaces and components.
- data structures, appropriate for classes implementation etc.
- components, exhibit independent functional characteristics.
- interfaces, reducing complexity of connections between components and external environment
- using a repeatable method, driven by info. obtained during S/W requirement analysis
- common, understandable notation

Quality attributes – FURPS (Functionality, usability, reliability, performance, supportability)

generic task set for design-

① examine the info. domain model and design appropriate data structures for data objects and their attributes.
② using the analysis model, select an architectural style (patterns) that is appropriate for S/w.
③ Partition the analysis model into design subsystems and allocate these subsystems within the architecture. Be certain that each subsystem is functionally cohesive.
Design subsystem interfaces.
Allocate analysis classes or functions to each subsystem.
④ Create a set of design classes or components.
Translate each analysis class description into a design class.
Check each design class against against design criteria; consider inheritance issues.
Define methods and messages associated with each design class.
Evaluate and select design patterns for a design class or a subsystem.
Review design classes and revise as required.
⑤ Design any interface required with external systems or devices.
⑥ design the user interface.
Review results of task analysis.
Specify action sequence based on user scenarios.
Create behavioral model of the interface.
Define interface objects, control mechanisms.
Review the interface design and revise as required.
⑦ Conduct component-level design.
Specify all algorithms at a relatively low level of abstraction.
Refine the interface of each component.
Define component-level data structures.
Review each component and correct all errors uncovered.
⑧ develop a deployment model.

Design concepts-

① Abstraction-
* derive both procedural and data abstractions that serve the problem at hand. If they can be serve an entire domain of problems, that's even better.
② Architecture - (design architecture explicitly.)
③ Patterns-
④ Modularity - (simple small modules).
* under modularity or over modularity should be avoided because it increases cost or effort.
⑤ information hiding - (to hide the details of data structures and procedural processing behind a module interface)
⑥ Functional Independence - (cohesion is qualitative indication of the degree to which a module focuses on just one thing.)
(coupling is " " " " " " " " " " " " " " is connected to other modules)
Refactoring

⑦ Refinement – (perform stepwise refinement to move to full detail)

⑧ Refactoring – (reorganization technique to simplify design (code) of a component without changing its function or behavior.)

⑨ design classes – (complete and sufficient, primitiveness, high cohesion, low coupling)

Design Model – (major elements → data, architecture, components, and interface)

① data design element (at the architectural (application) level, data design focuses on files or database, at the component level, data design considers data structures that are required to implement local data objects.)

② Architectural Design element – (architectural model is derived from the application domain, the analysis model, and available styles and patterns.)

③ interface design ~~elements~~ elements (
- user interface
- interfaces to system external to the application
- interfaces to components within the application

④ component-level design element

⑤ Deployment-level design elements (deployment diagrams begin in descriptor form, where the deployment environment is described in general terms. Later, instance form is used, and elements of the configuration are explicitly described.)

Pattern-based S/w design –

① describing a design pattern
* design forces are those characteristics of the problem and attributes of the solutions that constrain the way in which the design can be developed.

② using patterns in design
- Architectural patterns
- design patterns
- idioms (coding patterns)

③ frameworks (code skeleton that can be fleshed out with specific classes or functionality that have been designed to address the problem at hand.)

Creating an architectural Design – (represents the structures of data and program components that are required to build a computer-based system.)

steps: begins with data design, then proceeds to derivation of one or more representations of the architectural structure of the system. Alternative architectural styles or patterns are analyzed to derive the structure that is best suited to customer requirements and quality attributes. Once an alternative has been selected, the architecture is elaborated using an architectural design method.

S/w Architecture –

* S/w architecture must model the structure of a system and the manner in which data and procedural components collaborate with one another.

Data design –

① data design at architectural level
② " " " component "–(

Architectural Styles and Patterns –

① – Taxonomy of Architectural styles
- data-centered architecture
- data-flow architecture
- call and return architecture
- object-oriented architecture
- layered architecture

② Architectural Patterns –
issues { – concurrency
– Persistence
– Distribution

③ Organization and refinement –
- Control
- data

Architectural Design –

① Representing the system in Context (how the S/w interacts with entities external to its boundary)
② defining Archetypes (abstract building blocks of an architectural design)
③ Refining Architecture into components (
from three sources { – application domain
– infrastructure domain
– interface domain
④ describing instantiations of the system

Assessing Alternative Architectural designs :–

① Architecture Trade-Off Analysis method (collect scenarios, elicit requirements, constraints, and environment description, describe architectural style/patterns that have been chosen, evaluate quality attributes by considering each attribute in isolation, identify sensitivity of quality attributes to various architectural attributes, critique candidate architectures)
② Architectural Complexity
③ Architectural Description languages (ADL)

Mapping data flow into a S/w architecture :=(transform flow, transaction flow, transform mapping, transaction mapping, refining architectural design)

Modeling component-level design – defining the data structures, algorithms, interface characteristics, and communication mechanisms allocated to each S/W component.

steps :- design representations of data, architecture, and interfaces
- class definition or processing narrative for each component is translated into a detailed design that makes use of diagrammatic or text-based forms that specify internal data structures, local interface details, and processing logic.
- Design notation encompasses UML diagrams and supplementary representations.
- procedural design is specified using a set of structured programming constructs.

* from an OO viewpoint, a component is a set of collaborating classes.

designing class-based components

① basic design principles
- Open-closed principle (OCP) – (A module (component) should be open for extension but closed for modification)
- Liskov Substitution Principle (LSP) – (subclasses should be substitutable for their base class)
- Dependency Inversion Principle (DIP) – (Depend on abstractions. Don't depend on concretions)
  * if you dispense with design and hack out code, just remember that code is the ultimate "concretion". you are violating DIP.
- interface Segregation Principle (ISP) – (many client-specific interfaces are better than one general purpose interface)
- Release Reuse Equivalency Principle (REP) – (the granule of reuse is the granule of release)
  * designing components for reuse requires more than good technical design. It also requires effective configuration control mechanisms.
- Common Closure Principle (CCP) – (classes that change together belong together)
- Common Reuse Principle (CRP) – (classes that are not reused together should not be grouped together)

② component-level design guidelines –
- components
- interfaces
- dependencies and inheritance

③ Cohesion (keep cohesion as high as is possible)
- functional
- layer
- communicational
- sequential
- procedural
- Temporal
- Utility

④ Coupling (

– content coupling	– inclusion or import coupling
– Common coupling	– external coupling
– Control coupling	
– stamp coupling	
– data coupling	
– Routine call coupling	
– type use "	

(8) Conducting component-level design -

① identify all design classes that correspond to the problem domain
② " " " " " " " " infrastructure domain.
③ elaborate " " " that are not acquired as reusable components
- specify message details when classes or components collaborate.
- identify appropriate interfaces for each component.
- elaborate attributes and define data types and data structures required to implement them.
- describe processing flow within each operation in detail.

④ describe persistent data sources (databases and files) and identify the classes required to manage them.
⑤ develop and elaborate behavioral representations for a class or component stepwise.
⑥ elaborate ~~development~~ deployment diagrams to provide additional implementation detail.
⑦ factor every component-level design representation and always consider alternatives.

Performing user Interface Design := ~~as~~ (creates an effective communication medium between human and computer.)

steps :- identification of users, tasks, and environmental requirements
- once user task have been identified, user scenarios are created and analyzed to define a set of interface objects and actions.
- creation of screen layouts that depict graphical design and placement of icons, definition of descriptive screen text, specification and titling for windows, and specification of major and minor menu items.
- Tools are used to prototype and ultimately implement the design model, and
- ~~the~~ the result is evaluated for quality.

Golden Rules -

① place the user in control.
② Reduce the user's memory load
③ make the interface consistent

* even a novice user wants short-cuts; even knowledgeable users, frequent users sometimes need guidance. Give them what they need.
*

Testing Strategies =

steps :- begins "in the small" and progresses "to the large".

- ~~each~~ component / small group of related components → high order tests for integrated component / system

* three quality and error detection throughout the S/W process.
* an independent test group does not have the "conflict of interest" that builders of S/W might experience.
(ITG)

S/W testing strategy for conventional S/W architectures :-

code, unit test, integration test, design, validation test, requirement, system test, system engg.

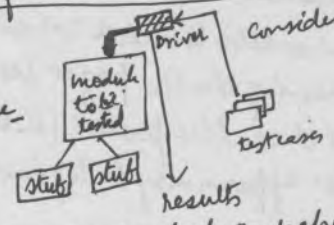
* OO testing begins "in the small". Mostly, the smallest element tested is a class / package of collaborating classes.

strategic issues -

- specify product requirements in a quantifiable manner long before testing commences.
- state testing objectives explicitly.
- understand the users of software and ~~develop~~ develop a profile for each user category
- develop a testing plan that emphasizes "rapid cycle testing"
- build "robust" S/W that is designed to test itself.
- use effective formal technical reviews as a filter prior to testing
- conduct formal technical reviews to assess the test strategy and test cases themselves.
- develop a continuous improvement approach for the testing process.

test strategies for Conventional S/W -

① unit test -



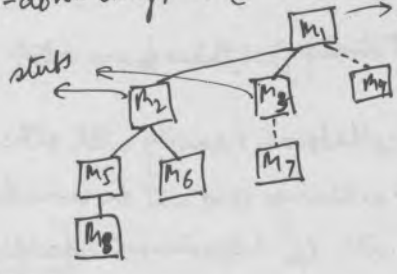
procedure -

consideration { interface, local data structures, boundary conditions, independent paths, Error handling paths ✓

~~unit test~~
* if there is no ~~enough~~ resources to do comprehensive unit testing, select critical modules and those with high cyclomatic complexity, and unit test only those.

② integration testing -

- top-down integration (incremental approach)



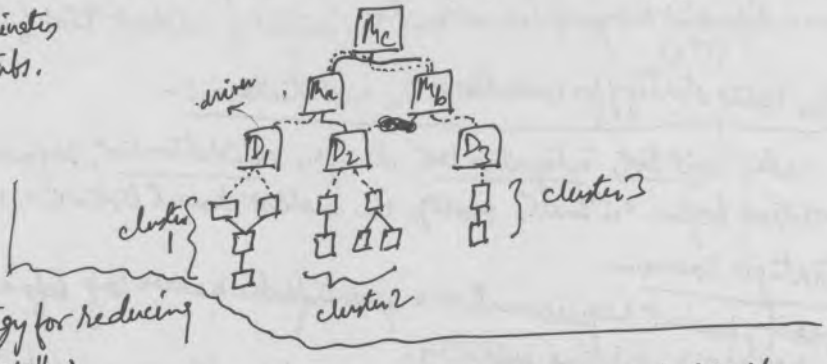
two manners :-
{ (i) depth - first integration
(ii) breadth - first integration

- * main control module is used as a test driver, and stubs are substituted for all components directly ~~subordinate~~ subordinate to the main control module.
- later, subordinate stubs are replaced one at a time with actual components.
- test are conducted as each component is integrated.
- regression test and a set of test is conducted every integration. after

*

- Bottom-up integration -
  - low-level components are combined into clusters (sometimes called builds) that perform a specific S/W subfunction.
  - a driver (a control program for testing) is written to coordinate test case input and output.
  - cluster is tested.
  - drivers are removed and clusters are combined moving upward in the program structure.

* bottom up integration eliminates the need for complex stubs.



Regression test - (a strategy for reducing "side effects".)

* run regression tests every time a major change is made to the S/W (including the integration of new components)

Smoke test - (rolling integration strategy)

* S/W is rebuilt (with new components added) and smoke tested every day.
  - S/W components are integrated into a "build" that includes all data files, libraries, reusable modules, and engineered components.
  - a series of tests is designed. The intent should be to uncover "show stopper" errors that have the highest likelihood of throwing the S/W project behind schedule.
  - the build is integrated with other builds and the entire product (in its current form) is smoke tested daily. The integration approach may be top down or bottom up.

- strategic options
  - sandwich testing (top down tests for upper levels of program structure + bottom-up tests for subordinate levels)
  - critical modules (integration test for highly controlled and complex/error prone modules)

* thread-based testing (OO software integration test strategy) = (threads are sets of classes that respond to an input or event.)
* use-based tests focus on classes that do not collaborate heavily with other classes.

Validation test - (focus on the requirement level - on things that will be immediately apparent to the end-user) in accordance with S/W requirement specification

- validation test criteria
- configuration review
- Alpha and Beta testing

System testing :- (Recovery testing, security test, stress test, performance test)

tools :- OTF (Object Testing Framework), QADirector, Testworks

Debugging process

Debugging strategies – (brute force, backtracking, cause elimination)

Tools:- JprobeThreadAnalyzer, C++test, CodeMedic, BugCollector, GNATS

Testing Tactics = provide systematic guidance for designing tests that (1) exercise the internal logic and interfaces of every software component and (2) exercise the input and output domains of the program to uncover errors in program function, behavior, and performance.

steps: For conventional applications, SW is tested from two different perspectives:
(1) internal program logic is exercised using white box test case design techniques.
~~For~~ object software requirements are exercised using black box test case design techniques.

For object-oriented application, testing begins prior to the existence of source code, but once the code has been generated, a series of tests are designed to exercise operations with a class and examine whether errors exist as one class ~~collaborates~~ collaborates with others. As classes are integrated to form a subsystem, use-based testing, along with fault-based approaches, is applied to fully ~~exercise~~ exercise collaborating classes. Finally, use-cases assist in the design of tests to uncover errors at the software validation level.

Testing fundamentals–

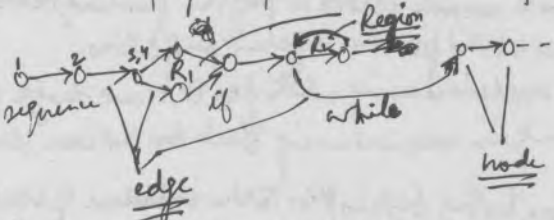
- testability
- operability
- observability
- controllability
- decomposability
- simplicity
- stability
- understandability

Test characteristics – a good test is/has
- high probability of finding an error.
- not redundant
- best of breed
- neither too simple nor too complex

Black-box and
White box / glass-box testing – can be designed only after component-level design (or source code) exists. The logical details of the ~~test~~ program must be available.

basis path testing = (a white-box testing technique, enables test case designer to derive a logical complexity measure of a procedural design and use this measure as a guide for defining a basis set of execution paths.)

(1) flow graph / program graph notation —
- should be drawn only when the logical structure of a component is complex.
- allows you to trace program paths more readily.



(2) independent program paths — (any path that introduces at least one new set of processing statements or new condition)

* cyclomatic complexity is a useful metric for predicting those modules that are likely to be error prone. use it for test planning as well as test case design. (complex it can be computed. (three ways)

measuring complexity of program

— the number of regions corresponds to the cyclomatic complexity

— cyclomatic complexity, V(G), for a flow graph, G, is defined as

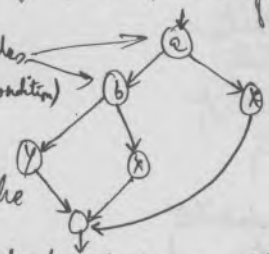
$$V(G) = E - N + 2$$

E → edge, N → node

— cyclomatic complexity, V(G), for a flow graph, G, is also defined as

$$V(G) = P + 1$$

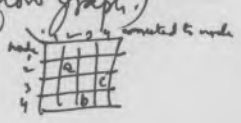
P → no. of predicate nodes (containing condition)



* cyclomatic complexity provides the upper bound on the number of test cases that will be required to guarantee that every statement in the program has been executed at least one time.

(3) Deriving test cases —
- using the design or code as a foundation, draw a corresponding flow graph.
- determine the cyclomatic complexity of the resultant flow graph.
- determine a basis set of linearly independent paths
- prepare test cases that will force execution of each path in the basis set

(4) Graph matrix — (square matrix whose size is equal to number of nodes on the flow graph.)
each row and column correspond to an identified node



Control structure testing –

① condition test (logical conditions)
② Data flow test (test paths of a program)
③ Loop test (simple, nested, concatenated, unstructured)

Black box testing = (behavioral testing) – focuses on functional requirements of S/W.)

① graph-based testing methods – (a graph represents the relationships between data objects and program objects, enabling us to derive test cases that search for errors associated with these relationships)
   - transaction flow modeling
   - finite state "
   - data flow "
   - timing "

② Equivalence partitioning – (divides the input domain of a program into classes of data from which test cases can be derived.)
   * input classes are known relatively early in S/W process ⇒ Begin thinking about equivalence partitioning as the design is created.

③ Boundary value analysis – (extends equivalence partitioning by focusing on data at the "edges"/boundary of an equivalence class.)

④ orthogonal array testing – (enables you to design test cases that provide maximum test coverage with a reasonable number of test cases)

Tools: McCabe Test, Panorama, TestWorks, T-Vec Test Generation System

Object-oriented Testing Methods –

- Fault-based testing – (to hypothesize a set of plausible faults and then derive tests to prove each hypothesis.)
   * even though a base class has been thoroughly tested, you will still have to test all classes derived from it.
- Scenario-based testing – (uncover errors that occur when any actor interacts with S/W)
   * use cases better
- testing surface structure (analogous to black box test) and deep structure (similar to white box test)

* test derived from behavior models (e.g. state diagram, ...)
* documentation testing :-

* testing for real-time system (task testing, behavioral test, inter-task test, system test)

* Product metrics = help to gain insight into the design and construction of S/w.
  - derive S/w measures and metrics
  - data required are collected
  - ~~is~~ computed and appropriate metrics are analysed
  - result interpretation and for quality and modification

Software quality factors:
(correctness, reliability, efficiency, integrity, usability, maintainability, flexibility, testability, portability, reusability, interoperability)

ISO 9126 quality factors – (functionality, reliability, usability, efficiency, maintainability, portability)

* measurement principles (formulation, collection, analysis, interpretation, feedback)
* Goal-oriented S/w measurement (Goal/Question/Metric (GQM))

Attributes of effective S/w metrics – (simple and computable, empirically and intuitively persuasive, consistent and objective, consistent in the use of units and dimensions, programming language independent, an effective mechanism for high-quality feedback)

* Metrics for analysis model (functionality delivered, system size, specification quality)
* Metrics for the design model (Architectural, component-level, specialized OO design)
* " " Source code (Halstead, complexity, ~~length~~ length metrics)
* " " testing (statement and branch coverage, defect-related, testing effectiveness, In-process)

Metrics for the Analysis Model :=

① Function-based metrics (no of external inputs, no of external outputs, no. of external inquiries, no of internal logical files, no. of external interface files)

$$FP = \text{count total} \times [0.65 + 0.01 \times \Sigma(F_i)]$$

F_i = value adjustment factors (VAF) ($i = 1$ to 14)

② metrics for specification Quality (by measuring characteristics of specification, it is possible to gain ~~quality~~ quantitative insight into ~~specification~~ specificity and completeness.)

* specificity (lack of ambiguity) of requirements $= Q_1 = \frac{n_{ui}}{n_r}$ → no. of requirements for which all reviewers had identical interpretations
  n_r → no. of requirements in specification $= n_f + n_{nf}$ (functional / non-functional)
* Completeness of functional requirements $= Q_2 = \frac{n_u}{[n_i \times n_s] + n_r}$
  n_i → no. of inputs, n_s → no. of states, n_r → unfulfilled req.
* Q_3 = degree to which requirements have been validated $= \frac{n_c}{[n_c + n_{nv}]}$
  n_c → no. of req. validated as correct, n_{nv} → no. of req. not yet been validated

* Architectural design metrics –

– structural complexity of a module = $S(i) = f^2_{out}(i)$ → fan-out of module i

– data complexity in internal interface for a module i = $D(i) = \frac{V(i)}{[f_{out}(i)+1]}$ → no. of input and output variables

– system complexity = $C(i) = S(i) + D(i)$

* class-oriented metrics – The CK metrics Suite –
  - weighted methods per class (WMC)
  - depth of inheritance tree (DIT)
  - no. of children (NOC)
  - coupling between object class (CBO)
  - Response for a class (RFC)
  - lack of cohesion in methods (LCOM)

* class-oriented metrics – the MOOD metrics suite –
  - method inheritance factor (MIF)
  - coupling factor (CF)

* Metrics for OO testing – (lack of cohesion in methods, percent public and protected, public access to data members, no. of root class, Fan-in, no. of children and depth of the inheritance tree)

Tools :- Krakatau Metrics, Metrics4C, RationalRose, RSM, UnderStand, – –

Web Engineering

Attributes of web-based systems and applications :-

- network intensiveness
- concurrency
- unpredictable load
- performance
- Availability
- data driven
- content sensitive
- Continuous evolution
- immediacy
- Security
- Aesthetics

webApp engineering layers –

- process (often Agile, incremental)
- methods (communication methods, requirements analysis methods, design methods, testing methods)
- tools and technology

* web engg. best practices =

(1) take the time to understand business needs and product objectives, even if the details of the webApp are vague.
(2) describe how users will interact with the webApp using a scenario-based approach.
(3) develop a project plan, even if it is very brief.
(4) spend some time modeling what it is that you are going to build.
(5) review the models for consistency and quality.
(6) use tools and technology that enable you to construct the system with as many reusable components as possible.
(7) don't rely on early users to debug the webApp – design comprehensive tests and execute them before releasing the system.

Metrics for web engg. and webApp: –

- Metrics for web engg. effort –
  * Application authoring and design tasks (structuring effort, interlinking effort, interface planning, interface building, link-testing effort, media-testing effort, total effort)
  * Page authoring (text effort, page-link effort, page-structuring effort, total page effort)
  * Media Authoring (media effort, media-digitizing effort, total media effort)
  * Program Authoring (programming effort, reuse effort)
- metrics for assessing business value

Tools :– Clicktracks, Marketforce, webMetrics Tetfed, webTrends

* Project management for web :–

- webApp Planning – outsourcing (
- initiate the project (analysis tasks, rough design, rough project schedule, milestones, list of responsibilities, the degree of oversight and interaction by the contracting organization with the vendor should be identified)
- select candidate outsourcing vendors
- assess the validity of price quotes and the reliability of estimates.
- understand the degree of project management you can expect/perform
- Assess the development schedule
- manage scope (frozen increment until next webApp increment)

* Architectural design metrics –

– structural complexity of a module = $S(i) = f^2_{out}(i)$ → fan-out of module i

– data complexity in internal interface for a module i = $D(i) = \frac{V(i)}{[f_{out}(i)+1]}$ → no. of input and output variables

– system complexity = $C(i) = S(i) + D(i)$

* class-oriented metrics – The CK metrics Suite –
  - weighted methods per class (WMC)
  - depth of inheritance tree (DIT)
  - no. of children (NOC)
  - coupling between object class (CBO)
  - Response for a class (RFC)
  - lack of cohesion in methods (LCOM)

* class-oriented metrics – the MOOD metrics suite –
  - method inheritance factor (MIF)
  - coupling factor (CF)

* Metrics for OO testing – (lack of cohesion in methods, percent public and protected, public access to data members, no. of root class, Fan-in, no. of children and depth of the inheritance tree)

Tools :- Krakatau Metrics, Metric 4C, RationalRose, RSM, UnderStand, – –

Web Engineering

* Attributes of web-based systems and applications :-
  - network intensiveness
  - concurrency
  - unpredictable load
  - performance
  - Availability
  - data driven
  - content sensitive
  - continuous evolution
  - immediacy
  - Security
  - Aesthetics

* WebApp planning – in house web Engg. – management –

- understand scope, dimensions of change, and project constraints
- define an incremental project strategy
- perform risk analysis
- develop a quick estimate
- select a task set (process description)
- establish a schedule
- define project tracking mechanisms (establish milestones)
- establish a change management approach.

Tools:- Business Engine, Teamwork, OurProject, Proj-net, StartWright

project management =

① people ② product ③ process ④ project

People - stakeholders
- team leaders (motivation, organization, ideas or innovation, problem solving, managerial identity, achievement, influence and team building)
- software team (
- Agile teams (self organizing team, has autonomy to plan and make decisions)
- coordination & communication issues

Product -- SW scope (Context, info. objectives, function and performance)
- problem decomposition (partitioning / problem elaboration)

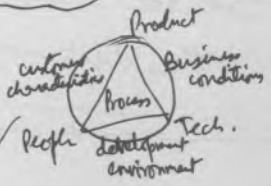
Process - melding product & process (scheduling tool like MS project pro)
- process decomposition

Project - (start on right foot, maintain momentum, track progress, make smart decisions, conduct a post mortem analysis).

W⁵HH Principle = why what when who where How Howmuch
system todo todo responsible people todo resource

Tools: - the Software program manager's network tool (www.spmn.com),
- check list (Ganttheadcom)
guides, templates, worksheets. (itoolkit.com)

Metrics for process and projects -



- process metrics & S/W process improvement
- project metrics (

SW measurement -
- size-oriented metrics
- function-oriented metrics
- reconciling LOC & FP metrics
- OO metrics (no. of scenario scripts, no. of key classes, no. of support classes, avg. no. of support classes per key class, no. of subsystems)
- use-case oriented metrics
- web engg. project metrics (no. of static web pages, no. of dynamic web pages, no. of internal page links, no. of persistent data objects, no. of ext. systems interfaced, no. of static content objects, no. of dynamic content objects, no. of custom executable functions)

Tools: - Metricenter, Function Point Workbench, Tychometrics (PSM Insight, SLIM tool set, SPR tool set,)

Metrics for S/W Quality -

① measuring quality - (correctness, maintainability, integrity & usability)

integrity = $\sum [1 - (\text{threat} \times (1 - \text{security}))]$

② defect Removal efficiency —

$$DRE = \frac{E}{E+D}$$

E → (no. of errors found before delivery of software)

D → (no. of defects found after delivery)

$$DRE_i = \frac{E_i}{(E_i + E_{i+1})}$$

E_i → (no. of errors found during S/w engg. activity i)

E_{i+1} → (" " " " " " " " i+1)

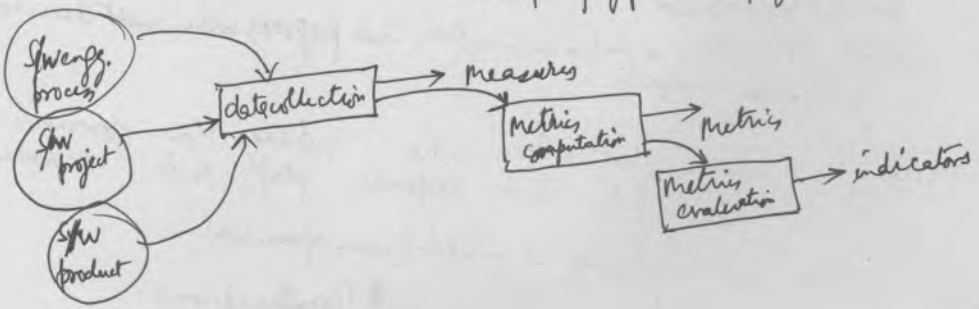
(to find errors before they are passed to next framework activity or S/w engg task)

Integrating metrics within the S/w process —

① Arguments for S/w metrics

② establishing a baseline

③ Metrics collection, computation, evaluation (baseline metrics data should be collected from a large representative sampling of past S/w projects)



Metrics for small organizations :

$$DRE = \frac{\text{Errors uncovered during work to make the change } (E_{change})}{E_{change} + D_{change}}$$

D_{change} → defects uncovered after change is released

ESTIMATION =

Task set for Project Planning —

1. establish project scope (communication with all stakeholders, use-cases) *
2. Determine feasibility and business needs
3. Analyze risks
4. define required resources
   a. determine human resources required (location, skills, number)
   b. define reusable S/w resources (COTS, components, full-experience components, part-experience components, new components)
   c. identify environmental resources (S/w tools, Hardware, Network)
5. estimate cost and effort (delay estimation until late in project, base estimates on similar projects completed, simple decomposition, use one or more empirical models)
   * Consider additional costs also like development environment and tool, travel, training, hardware, office supplies
   a. decompose the problem (simple decomposition like size in LOC or FP)
   b. develop two or more estimates using size, function points, process tasks, or use-cases (use one or more empirical models)
   c. Reconcile the estimates (to produce a single estimate of effort, duration, cost)
6. Develop a project schedule
   a. establish a meaningful task set
   b. define a task network
   c. use scheduling tools to develop a timeline chart
   d. define schedule tracking mechanisms

* be sure to establish a taxonomy of project types. This enables you to compute domain specific averages, making estimation more accurate.

* For FP estimates, decomposition focuses on information domain characteristics.

three-point or expected value, $S = \frac{S_{optimistic} + 4S_{most likely} + S_{pessimistic}}{6}$

Decomposition techniques –

① Software size in LOC or FP
② Problem based estimation
③ Process based " (into set of tasks)
④ use cases " " (
⑤ Reconciling estimates (to produce a single estimate of effort, duration, cost)

Empirical Estimation Models –

① effort in person-month, $E = A + B \times (e_v)^C$
↳ estimation variable (LOC or FP)
A, B, C constants

② COCOMO II – (application composition model, early design stage model, post-architecture stage model)
– based on object-points (computed using counts of no. of screens (at the user interface) & ③ reports ② components)
(Each is classified into simple/medium/difficult level complexity.) & complexity weight is assigned.
(– object point count = no. of object instances × complexity weight

when component-based development or general s/w reuse is to be applied, the percent of reuse (% reuse) is estimated and the object point count is adjusted:

new object points, NOP = (object points) × [(100 – % reuse)/100]
productivity rate, PROD = NOP/person-month
estimated efforts = NOP/PROD

③ Software equation:–
effort in person-month or person-years, $E = [LOC \times B^{0.333}/P]^3 \times (1/t^4)$
B → special skills factor
P → productivity parameter
t → duration in math or years

Estimation for OO projects –

– estimate using effort decomposition, FP, or...
– use case/OOA modeling, develop use-cases and determine a count. iteratively.
– From analysis model, determine no. of key classes (analysis classes)
– categorize type of interface for application, and develop a multiplier for support classes
– (no. of key classes × multipliers) = estimate for no. of support classes
– total no. of classes (key + support) × avg. no. of work-units per class
– Cross-check the class-based estimate by multiplying the average no. of work-units per use-case

Specialized Estimation Techniques –

⑧ estimation based for Agile development based on historical data, empirical model, "experience", LOC, FP, object points, ...
⑨ " " web app. projects " " inputs, outputs, tables, interfaces, queries

Tools:– Costar, Cost Xpert, Estimate professional, Knowledge plan, Price S, SEER/SEM, SLIM-Estimate

Make/Buy Decision :– ① creating decision tree (build, buy, reuse, contract)
② outsourcing

* Earned Value Analysis –

$\sum_k$ budgeted cost of work scheduled (BCWS) for all tasks k = BAC, budget at completion

Schedule performance index, SPI = $\frac{\text{BCWP (budgeted cost of work performed)}}{\text{BCWS}}$

Schedule variance, SV = BCWP − BCWS

Percent scheduled for completion = BCWS / BAC

Percent complete = BCWP / BAC

cost performance index, CPI = BCWP / ACWP → Actual cost of work performed

cost variance, CV = BCWP − ACWP

Risk management – (known & unknown risks), (predictable & unpredictable) risks

- project risks
- technical risks
- Business ''

Risk management principles –

- maintain a global perspective
- take a forward-looking view
- encourage open communication
- integrate
- emphasize a continuous process
- develop a shared product vision
- encourage teamwork

– Risk identification – (product size, business impact, customer characteristics, process definition, development environment, technology to be built, staff size and experience)

Risk components and drivers –

- performance risk
- cost risk
- support risk
- schedule risk

Risk Projection – (risk estimation) = (develop risk table based on probability and impact, assessing risk impact by RE = P × C (probability, cost of the project); risk exposure)

Risk Refinement = (into a set of more detailed risks, each somewhat easier to mitigate, monitor, manage)

Risk mitigation, monitoring and management =

- risk avoidance
- risk monitoring
- risk management and contingency planning

* if RE is less than cost of mitigation, do not try to mitigate that risk, but continue to monitor.

Risk mitigation, monitoring & management Plan (RMMM Plan) = (document all work performed as part of risk analysis)

* Risk information sheet (RIS) is also used for the same purpose (documentation of individual risk)

Tool: Riskman, RiskRadar, RiskTrak, Riskit, X:Planner

Quality management =

quality concepts - (quality, quality control, quality assurance, cost of quality, ...)

SQA (software quality assurance) - (prepares SQA plan, participate in development of software process description, review S/w engg. activities to verify compliance with defined S/w process, audit designated S/w work products to verify compliance with those defined as part of S/w process, ensures that deviations in S/w work and work products are documented and handled according to a documented procedure, records any non-compliance and reports to senior management)

S/W Reviews = (like filters in S/w process workflow. Use metrics to determine which reviews work and emphasize them.)

- formal technical review (FTR) / walkthrough / inspection
  (review meeting, reporting, record keeping)
  guidelines (review product not producer, set agenda and maintain it, limit debate and rebuttal, enunciate problem areas, take written note, limit no. of participants and insist upon advance preparation, develop checklist for each product to be reviewed, allocate resources and schedule time for FTRs, conduct meaningful training for all reviewers, review your early reviews)
- sample-driven reviews

Statistical S/W quality assurance - (information about S/w defects is collected and categorized, attempt is made to trace each defect to its underlying cause, using Pareto principle 80% defects from 20% causes, move to correct the problems that have caused defects)

- generic method
- six sigma for S/w engg. (- define requirements, deliverables, goals via communication
  - measure existing process and its output to determine current quality performance (collect defect metrics)
  - analyze defect metrics and determine the vital few causes

  for process improvement, two additional steps are:
  - improve process by eliminating root causes of defects
  - control process to ensure that future work does not reintroduce causes of defects.

** DMAIC = (define, measure, analyze, improve, control) method

S/W Reliability = ① mean-time-between-failure (MTBF) = MTTF + MTTR
(MTTF: mean time to failure; MTTR: mean time to repair)

Availability = [MTTF / (MTTF + MTTR)] × 100%

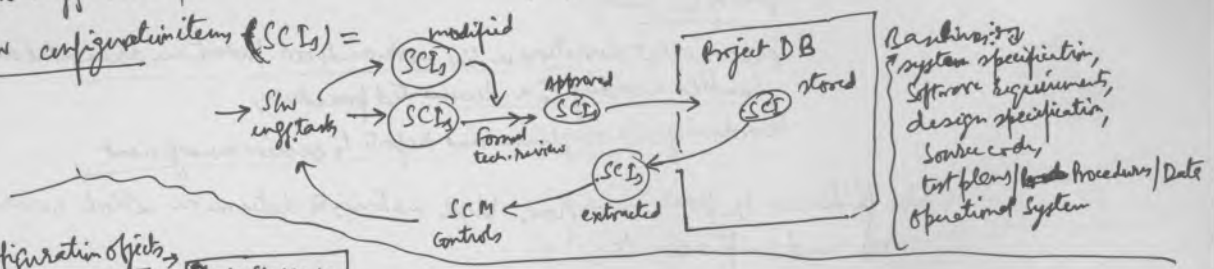
② software safety

ISO 9001:2000 Standard :- (establish elements of quality management system. Develop, implement, improve system. Define policy to emphasize importance of system, document quality system, Describe process, produce operational manual, develop methods for controlling & updating documents, ...)

Sw change management (Sw configuration management) =

* there must be a mechanism to ensure that simultaneous changes to the same component are properly tracked, merged, and executed.

- elements of configuration management system – (Component element, process element, construction element, human element)
- Baselines
- Sw engg. work product becomes a baseline only after it has been reviewed and approved.
- Sw configuration items (SCIs) =



Configuration objects →

SCM Repository – role:- (data integrity, information sharing, tool integration, data integration, methodology enforcement, document standardization)

– Contents of repository – (Business, project management, model, construction, V&V, documents)

– SCM features – (versioning, dependency tracking and change management, requirements tracing, configuration management, Audit trails)

– SCM process – (SCIs, identification, change control, version control, configuration auditing, reporting)

Assess and synchronization control by version and change control tools

"by "make" facility in (CVS) concurrent Version System

– configuration Audit – (Complement formal technical review)

– status reporting (keep "needs to know" list for every configuration object up-to-date and notified to everyone.)

Tools:- CCC/Harvest, ClearCase, Concurrent Version Control System (CVS), PVCS, SourceForge, SurroundSCM, Vesta

SCM process =

① to identify all items that collectively define S/w configuration
② to manage changes to one or more of these items
③ to facilitate the construction of different versions of an application
④ to ensure that software quality is maintained as the configuration evolves over time.