Jitendra Bhatta

BESE 8th (161747)

SPM Assignment

Unit 1: Important questions

Q.Explain briefly Waterfall model. Also explain Conventional s/w management performance.

WATERFALLL MODEL :

It is the baseline process for most conventional software projects have used.

We can examine this model in two ways:

i. IN THEORY
ii. IN PRACTICE

IN THEORY:

- In 1970, Winston Royce presented a paper called "Managing the Development of Large Scale Software Systems" at IEEE WESCON.
- Where he made three primary points:

1. There are two essential steps common to the development of computer programs:

-analysis

-coding

Analysis ⟹ coding

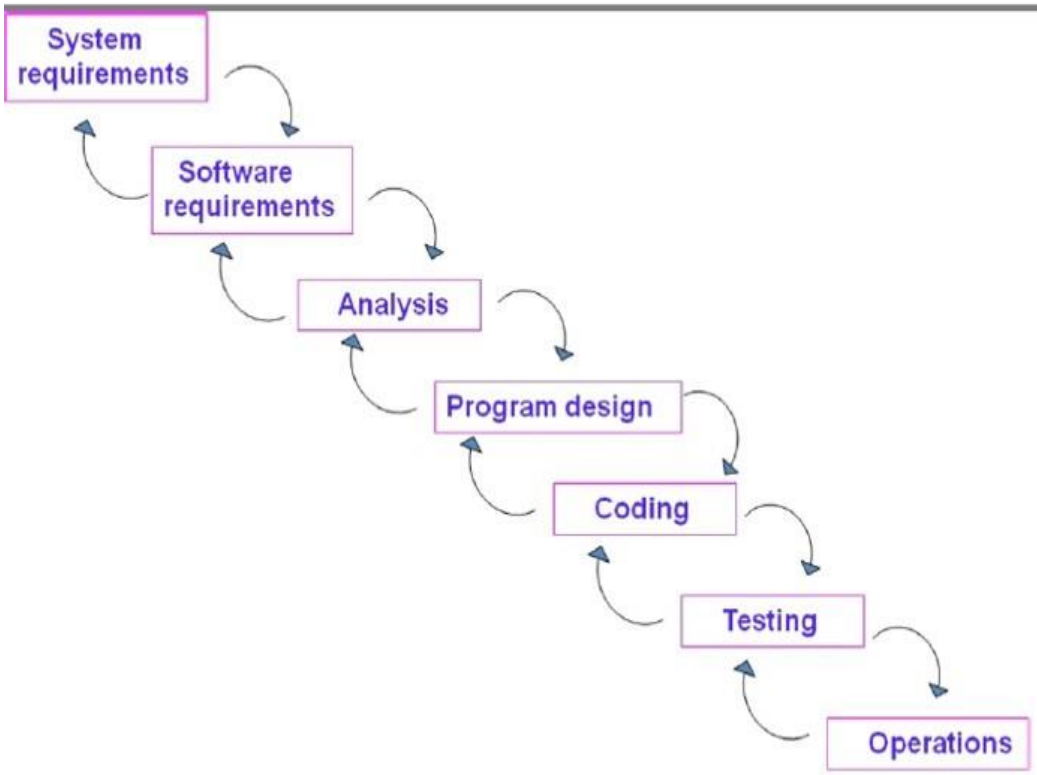
- Analysis and coding both involves creative work that directly contributes to the usefulness of the end product .

Waterfall Model Part I: The two basic steps to build a program

2. In order to mange and control all of the intellectual freedom associated with software development one should follow the following steps:

- System requirements definition
- Software requirements definition
- Program design and testing

- These steps addition to the analysis and coding steps



Waterfall Model Part 2: The large – scale system approach

3. Since the testing phase is at the end of the development cycle in the waterfall model, it may be risky and invites failure.

- So we need to do either the requirements must be modified or a substantial design changes is warranted by breaking the software in to different pieces.

  - There are five improvements to the basic waterfall model that would eliminate most of the development risks are as follows:

a) **Complete program design before analysis and coding begin (program design comes first):-**

- By this technique, the program designer give surety that the software will not fail because of storage, timing, and data fluctuations.

- Begin the design process with program designer, not the analyst or programmers.

- Write an overview document that is understandable, informative, and current so that every worker on the project can gain an elemental understanding of the system.

b) Maintain current and complete documentation (Document the design):-

- It is necessary to provide a lot of documentation on most software programs.
- Due to this, helps to support later modifications by a separate test team, a separate maintenance team, and operations personnel who are not software literate.

c) Do the job twice, if possible (Do it twice):

- If a computer program is developed for the first time, arrange matters so that the version finally delivered to the customer for operational deployment is actually the second version insofar as critical design/operations are concerned.
  - "Do it N times" approach is the principle of modern-day iterative development.

d) Plan, control, and monitor testing:

- The biggest user of project resources is the test phase. This is the phase of greatest risk in terms of cost and schedule.
- In order to carryout proper testing the following things to be done:

i) Employ a team of test specialists who were not responsible for the original design.

ii) Employ visual inspections to spot the obvious errors like dropped minus signs, missing factors of two, jumps to wrong addresses.

iii) Test every logic phase.

iv) Employ the final checkout on the target computer.

e) Involve the customer:

- It is important to involve the customer in a formal way so that he has committed himself at earlier points before final delivery by conducting some reviews such as,

i) Preliminary software review during preliminary program design step.

ii) Critical software review during program design.

iii) Final software acceptance review following testing.

IN PRACTICE:

- Whatever the advices that are given by the software developers and the theory behind the waterfall model, some software projects still practice the conventional software management approach.
- Projects intended for trouble frequently exhibit the following symptoms:

  i) Protracted (delayed) integration
- In the conventional model, the entire system was designed on paper, then implemented all at once, then integrated.
- Only at the end of this process was it possible to perform system testing to verify that the fundamental architecture was sound.
- Here the testing activities consume 40% or more life-cycle resources.

ACTIVITY	COST
Management	5%
Requirements	5%
Design	10%
Code and unit testing	30%
Integration and test	40%
Deployment	5%
Environment	5%

Conventional Software Management Performance:

Conventional software management practices are sound in theory, but practice is still tied to archaic technol¬ogy and techniques. Conventional software economics provides a benchmark of performance for conventional software manage¬ment principles. The best thing about software is its flexibil¬ity: It can be programmed to do almost anything. The worst thing about software is also its flexibility: The "almost anything" characteristic has made it difficult to plan, monitors, and control software development

Three important analyses of the state of the software engineering industry are

1. Software development is still highly unpredictable. Only about 10% of software projects are delivered successfully within initial budget and sched¬ule estimates.
2. Management discipline is more of a discriminator in success or failure than are technology advances.
3. The level of software scrap and rework is indicative of an immature process.

   All three analyses reached the same general conclusion: The success rate for soft¬ware projects is very low. The three analyses provide a good

introduction to the magnitude of the software problem and the current norms for conventional software management performance.

Barry Boehm's "Industrial Software Metrics Top 10 List" is a good, objective characterization of the state of software development.

1. Finding and fixing a software problem after delivery costs 100 times more than finding and fixing the problem in early design phases
2. You can compress software development schedules 25% of nominal, but no more.
3. For every $1 you spend on development, you will spend $2 on maintenance
4. Software development and maintenance costs are primarily a function of the number of source lines of code
5. Variations among people account for the biggest differences in software productivity
6. The overall ratio of software to hardware costs is still growing. In 1955 it was 15:85; in 1985, 85:15.
7. Only about 15% of software development effort is devoted to programming.
8. Software systems and products typically cost 3 times as much per SLOC as individual software programs. Software-system products (i.e., system of sys¬tems) cost 9 times as much.
9. Walkthroughs catch 60% of the errors 80% of the contribution comes from 20% of the contributors.

Q)Define Software Economics. Also explain pragmatic s/w cost estimation?

- Software economics is the study of how scarce project resources are allocated for software projects.
- Software economics helps software managers allocate those resources in the most efficient manner.
- Software Economics includes many different disciplines which are shown below:

Pragmatic s/w cost estimation:

It is based on a well-defined **software cost** model with a credible basis. It is based on a database of relevant **project** experience that includes similar processes, similar technologies, similar environments, similar quality requirements, and similar people.

- Little available on estimating cost for projects using iterative development.
  - Difficult to hold all the controls constant
    - Application domain; project size; criticality; etc. Very 'artsy.'
    - Metrics (SLOC, function points, etc.) NOT consistently applied EVEN in the same application domain!
    - Definitions of SLOC and function points are not even consistent!
  - Much of this is due to the nature of development. there is no magic date when design is 'done;' or magic date when testing 'begins' …

Three Issues in Software Cost Estimation:

- . Which cost estimation model should be used?
- 2. Should software size be measured using SLOC or Function Points? (there are others too…)
- 3. What are the determinants of a good estimate? (How do we know our estimate is good??)

So very much is dependent upon estimates!!!!

Cost Estimation Models:

- Many available.
- Many organization-specific models too based on their own histories, experiences...
  - Oftentimes, these are super if 'other' parameters held constant, such as process, tools, etc. etc.
- COCOMO, developed by Barry Boehm, is the most popular cost estimation model.
- Two primary approaches:
  - Source lines of code (SLOC) and
  - Function Points (FP)

Source Lines of Code (SLOC):

- Many feel comfortable with 'notion' of LOC
- SLOC has great value – especially where applications are custom-built.
  - Easy to measure & instrument – have tools.

- Nice when we have a history of development with applications and their existing lines of code and associated costs.

◆ Today – with use of components, source-code generation tools, and objects have rendered SLOC somewhat ambiguous.

- We often don't know the SLOC – but do we care? How do we factor this in? →

◆ In general, using SLOC is a more useful and precise basis of measurement.

◆ See Appendix D – an extensive case study.

- Addresses how to count SLOC where we have reuse, different languages, etc.

◆ We will address LOC in much more detail later. In the meantime, I urge you to read the five pages in Appendix D entitled, Software Size Evolution from the Case Study. Starts on page 348.

◆ This exposition provides a hint at the complexity of using LOC for software sizing particularly with the new technologies using automatic code generation, components, development of new code, and more.

◆ Read these five pages.

Function Points:

◆ Use of Function Points - many proponents.

- International Function Point User's Group – 1984 – "is the dominant software measurement association in the industry."
- Check out their web site
- Tremendous amounts of information / references
- Attempts to create standards….

Major advantage: Measuring with function points is independent of the technology (programming language, …) used and is thus better for comparisons among projects.

◆ Function Points measure numbers of

- external user inputs,
- external outputs,
- internal data groups,
- external data interfaces,
- external inquiries, etc.

◆ Major disadvantage: Difficult to measure these things.

- Definitions are primitive and inconsistent
- Metrics difficult to assess especially since normally done earlier in the development effort using more abstractions.

A Good Project Estimate:

◆ Is conceived and supported by the project manager, architecture team, development team, and test team accountable for performing the work.

◆ Is accepted by all stakeholders as ambitious but doable

◆ Is based on a well-defined software cost model with a credible basis

◆ Is based on a database of relevant project experience that includes similar processes, similar technologies, similar environments, similar quality requirements, and similar people, and

◆ Is defined in enough detail so that its key risk areas are understood and the probability of success is objectively assessed.

Q3) Explain important trends in improving software economics?

4. Software Economics Improvement Trends

Cost Model Parameters	Trends
Size Abstraction and component based development technologies	Higher-order languages (C++, Ada 95), Object-oriented (analysis, design, programming), reuse, commercial components
Process Methods and techniques	Iterative development, process maturity levels, architecture first development, acquisition reform
Personnel People factors	Training and personnel skill development, teamwork, win-win conditions
Environment Automation technologies and tools	Integrated tools (visual modeling, compiler, editor, debugger, change management), open systems, hardware platform performance, automation of coding, documentation, testing, analysis
Quality Performance, quality, accuracy	Hardware platform performance, demonstration-based assessment, statistical quality control

Q4) Explain five staffing principal offered by Boehm. Also explain Peer Inspections?

Boehm five staffing principles are -

- The principle of top talent: Use better and fewer people
- The principle of job matching: Fit the tasks to the skills and motivation of the people available.
- The principle of career progression: An organization does best in the long run by helping its people to self-actualize.

- The principle of team balance: Select people who will complement and harmonize with one another
- The principle of phase-out: Keeping a misfit on the team doesn't benefit anyone.

Peer inspection:

• Improving software quality remains a key challenge.

• Software development formal peer inspection has emerged as an effective approach to address this challenge.

• Software peer inspection aims at detecting and removing software development defects efficiently and early while defects are less expensive to correct.

• Transitioning engineering information from one artifact set to another, thereby assessing the consistency, feasibility, understandability, and technology constraints inherent in the engineering artifacts.

• Major milestone demonstrations that force the artifacts to be assessed against tangible in the context of relevant use cases.

• Environment tools that ensure representation rigor, consistency, completeness, and change control.

• Life-cycle testing for detailed insight into critical trade-offs, acceptance criteria, and requirements compliance.

• Change management metrics for objective insight

• Keep the review team small

• Find problems during reviews, but don't try to solve them

• Limit review meetings to about two hours.

• Require advance preparation

Critical component deserves to be inspected by several people, preferably those who have a stake in its quality, performance, or feature set. An inspection focused on resolving an existing issue can be an

effective way to determine cause or arrive at a resolution once the cause is understood. Random human inspections tend to degenerate into comments on style and first-order semantic issues. They rarely result in the discovery of real performance bottlenecks, serious control issues (such as deadlocks, races, or resource contention), or architectural weakness (such as flaws in scalability, reliability, or interoperability).

• Quality assurance is everyone's responsibility and should be integral to almost all process activities instead of a separate discipline performed by quality assurance specialists.

• Evaluating and assessing the quality of the evolving engineering baselines should be the job of an engineering team that is independent of the architecture and development team. • Their life-cycle assessment of the evolving artifacts would typically include change management, trend analysis, and testing, as well as inspection.

Q5.Explain principles of conventional software engineering?

Principles of Conventional Software Engineering We know that there are many explanations and descriptions of engineering software "the old way". After many years of experience regarding software development, the software industry has learned and understood many lessons and formulated or created many principles. Some of these principles are given below :

1. Create or Make Quality – Quality of software must be measured or quantified and mechanisms put into place to motivate towards achieving the goal.

2. Large or high-quality software is possible – There are several techniques that have been given a practical explanation to raise and increase and quality includes involving customer, prototyping, simplifying design, conducting inspections, and hiring good and best people.

3. Give end-products to Customers early – It doesn't matter how hard we try to learn and know about needs of users during requirements

stage, most important and effective to determine and identify their real and realistic needs is to give product to users and let them play with it.

4. Determine problem or issue before writing and needs or requirements – When engineers usually face with what they believe is a problem, they rush towards offering a solution. But we should keep in mind that before we try to solve any problem, be sure to explore all alternatives and just don't get blinded by obvious solution.

5. Evaluate all Design Alternatives – When requirements or needs are agreed upon, we must examine and understand various architecture and algorithms. We usually don't want to use "architecture" generally due to its usage in requirements specification.

6. Use an appropriate and correct process model – Each and every project must select an appropriate process that makes most sense for that project generally on basis of corporate culture, willingness to take risks, application area, volatility of requirements, and extent to which all requirements and needs are understood in a good manner.

7. Use different languages for various phases – Our industry generally gives simple solutions to large complex problems. Due to his, many declare that best development method is only one that makes use of notation throughout life cycle.

8. Minimize or reduce intellectual distance – The structure of software must be very close enough to a real-world structure to minimize intellectual distance.

9. Before tools, put Techniques – An undisciplined software engineer with a tool becomes very dangerous and harmful.

10. Get it right just before we make it very faster – It is very easy and simple to make a program that's being working run very faster than it is to simply make a program work fast.

11. Inspect Code – Assessing or inspecting design with its details and code is most and better way of identifying errors other than testing.

12. Rather than Good technology, Good management is more important – Good management simply motivates other people also to do their work at best, but there are no universal "right" styles for management.

13. Key to success is "PEOPLE" – People with high and many skills, experience, talent, and training are key to success.

14. Follow with Care and in a proper manner – just because everyone is doing something doesn't mean that it is right for us. It may or may not be right for us. So, we must inspect or assess its applicability to our environment very carefully.

15. Take Responsibility – When a bridge collapse we only ask one question, "What did engineers do wrong?" Even when software fails, we rarely ask this question. This is due to that in any engineering discipline, best and important methods can be used to produce or develop an awful design, and best and largely antiquated methods to develop an elegant design.

16. Understanding priorities of customers – It is simply possible that customer would tolerate 90 percent of performance that is delivered late only if they could have 10 percent of it on time.

17. More they see, more they need – More the functionality or performance we provide to user, more functionality or performance they will want. Their expectation increases by time to time.

18. Plan to throw one away – The most essential and critical factor is whether a product is entirely new or not. Brand-new applications, architectures, interfaces, or algorithms generally work rarely for first time.

19. Design for update or change – We should update or accommodate change architecture, components, and specification techniques that we generally use.

20. Without documentation, design is not a design – We have some engineers often saying that "I have finished and completed design and all things left is documentation".

21. Be realistic while using tools – Tools that software used make their customers and users more efficient.

22. Encapsulate – Encapsulate simply means to hide information and it is easy, simply. It is a concept that simply results in software that is simpler and easier to test and very easy to maintain.

23. Avoid tricks – Some of programmers love to create and develop programs with some tricks constructs that perform a function in correct way but in a non-understandable manner. Rather just prove to world how smart we are by avoiding trick code.

24. Don't test-own software – Software Developers should not test their own software. They should not be primary testers of their own software.

25. Coupling and Cohesion – Using coupling and cohesion are best way to measure software's inherent maintainability and adaptability.

25. Except for excellence – Our employees will work in a far better way if we have high expectations for them.

Q6.Explain briefly principal of modern software management ?

Modern Principles Of Software Management There are some modern principles for the development of software. By following these modern principles we can develop an efficacious software meeting all the needs of customer. To develop a proper software one should follow the following 10 Principles of software development.

These are explained as following below:

1. Architecture first approach: In this approach over the main aim is to build a strong architecture for our software. All the ambiguities and flaws are being identified during the very trivial phase. Also, we can take

all the decisions regarding the design of the software which will enhance the productivity of our software.

2. Iterative life cycle process: An iterative life cycle process we repeated the process again and again to eliminate the risk factors. An iterative life cycle we mainly have four steps requirement gathering, design, implementation, and testing. All these steps are repeated again and again until we mitigate the risk factor. Iterative life cycle process is important to alleviate risk at an early stage by repeating the above-mentioned steps again and again.

3. Component Based Approach: In component-based approach is a widely used and successful approach in which we reuse the previously defined functions for the software development. We reuse the part of code in the form of components. Component-based UI Development Optimizes the Requirements & Design Process and thus is one of the important modern software principle.

4. Change Management system: Change Management is the process responsible for managing all changes. The main aim of change management is to improve the quality of software by performing necessary changes. All changes implemented are then tested and certified.

5. Round Trip Engineering: In round trip engineering code generation and reverse engineering take place at the same time in a dynamic environment. Both components are integrated so that developers can easily work on both of them. In round trip engineering, the main automatic update of artifacts.

6. Model Based Evolution:

Model-based evolution is an important principle of software development. A model-based approach supports the evolution of graphics and textual notions.

7. Objective Quality Control: The objective of quality control is to improve the quality of our software. It involves Quality management

plan, Quality metrics, Quality checklist, Quality baseline, and Quality Improvement measures.

8. Evolving levels of details: Plan intermediate releases in groups of usage scenarios with evolving levels of details. We must plan an incremental realize in which we have an evolving level of use case, architecture, and details.

9. Establish a configurable process: Establish a configurable process that is economically scalable. One single process is not suitable for all the development so we must use a configurable process which can deal with various applications.

10. Demonstration Based approach: In this approach, we mainly focus on demonstration. It helps in the increase of productivity and quality of our software by representing a clear description about problem domain, approaches used and the solution.