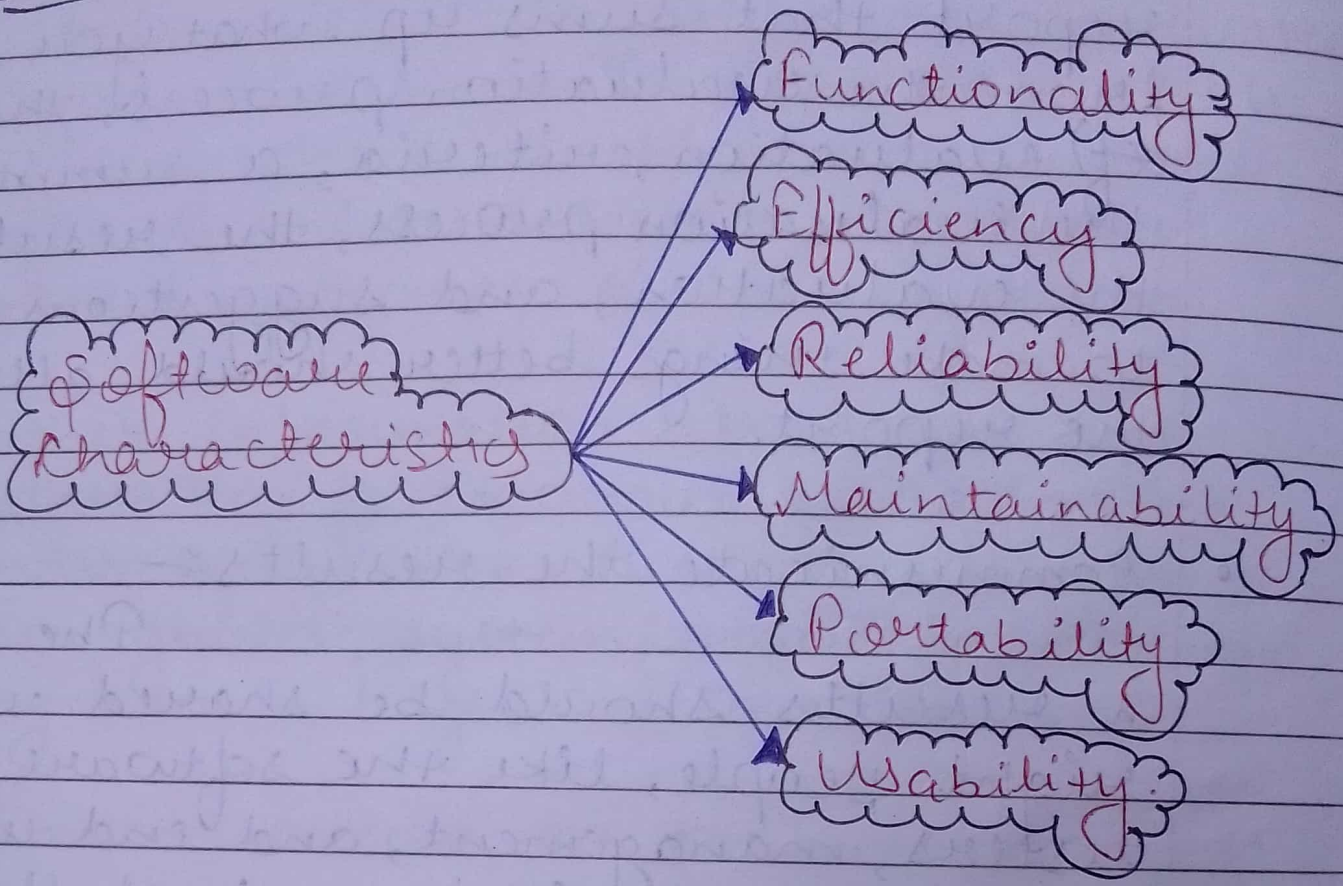
and expectations.

Components of Software Characteristics:-



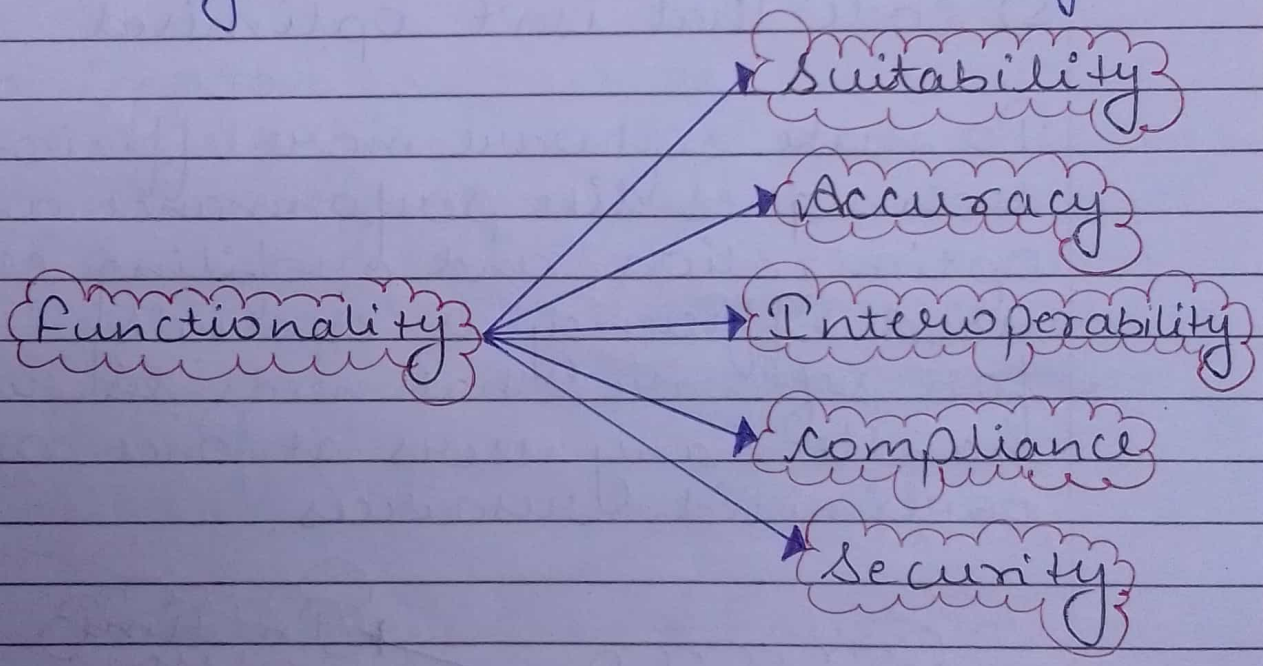
1) Functionality:-

It refers to how well software works for its intended purpose. It includes the features and capabilities that help users accomplish their tasks. Some key examples of functionality in software are:

- Data storage and retrieval
- Data processing and manipulation

- User interface and navigation
- Communication and networking
- Security and access control
- Reporting and visualization
- Automation and scripting.

More functionality means the software can do more things, but it can also become more complicated. It's essential to find a balance between having many features and ensuring the software is easy to use, maintain, and grow



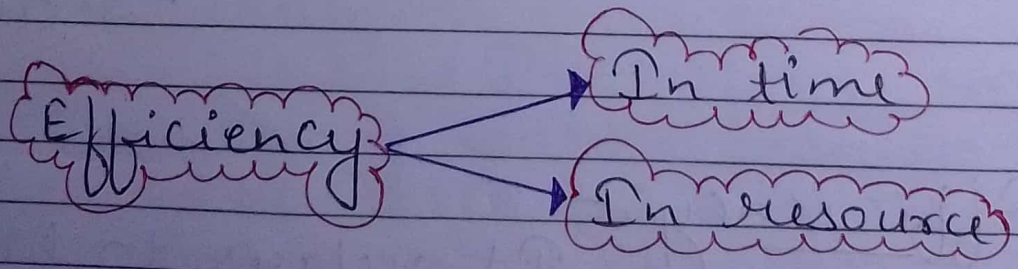
2) Efficiency:-

It refers to how well software uses system resources like memory, processing power, and network bandwidth. Efficient software runs quickly and uses fewer resources while inefficient software can be slow

and use too much.
Factors that can affect software efficiency include:

1) Poorly designed algorithms and data structures.
2) Inefficient use of memory and processing power
3) High network delays or heavy bandwidth use
4) Unnecessary calculations or processing power
5) Code that isn't optimised

To make software more efficient, techniques like performance analysis, optimization, and profiling can be used. Efficiency is especially important for software that needs to run fast, handle many users at once, or work on limited resources.

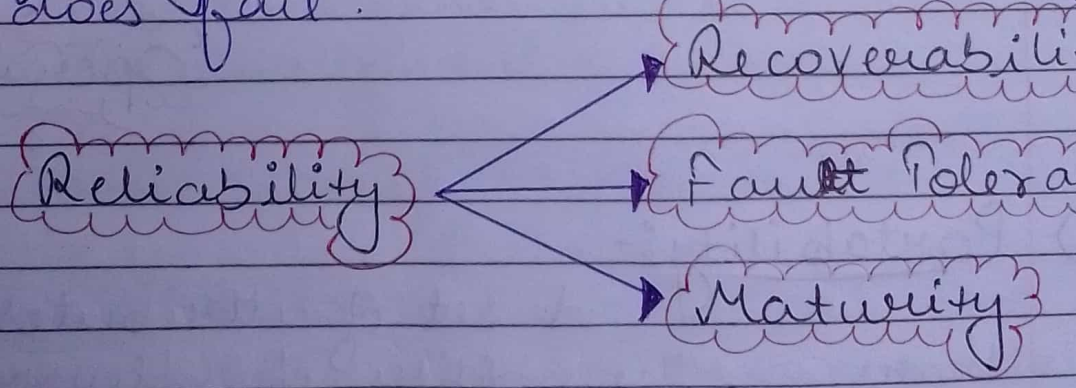


3) Reliability:-

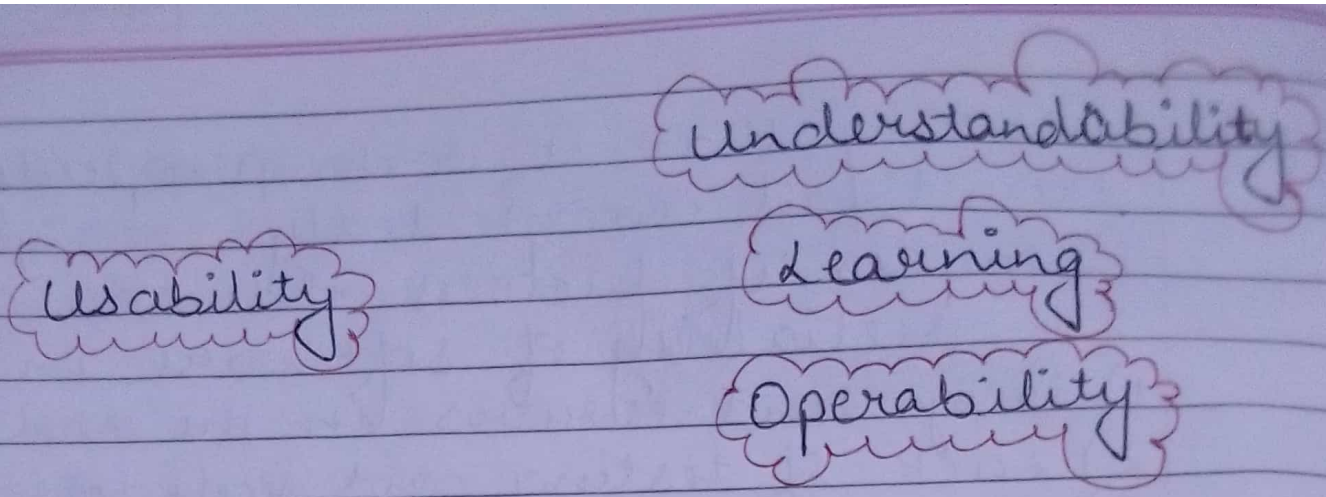
It refers to how well software can perform it's task correctly and consistently over time. It's an important part of software quality,

ensuring that it works properly and doesn't fail unexpectedly.
Examples of factors that can affect the reliability of software include:
1) Bugs and errors in the code
2) Lack of testing and validation
3) Poorly designed algorithms and data structures
4) Inadequate error handling and recovery
5) Incompatibilities with other softwa-re or hardware

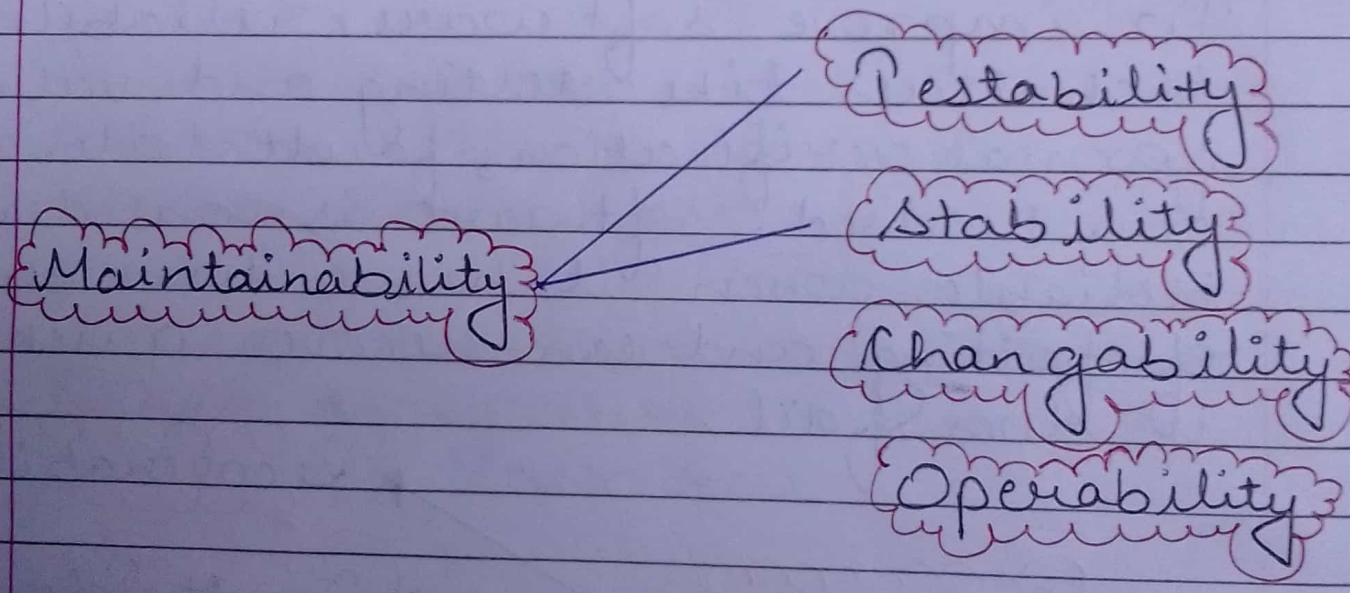
To improve software reliability, techniques like testing and validation, formal verification, fault tolerance can be used. Software is considered reliable when it has a low chance of failing and can recover quickly if it does fail.



4) Usability :-
It refers to extent to which can be used with ease, the amount of effort or time required to learn how to use the software



5) Maintainability:- It refers to the ease with which modifications can be made in a software system to extend its functionality, improve its performance, or correct errors



6) Portability:- A set of attributes that bears on the ability of software to be transferred from one environment to another, without minimum changes.

Adaptability

Portability

Installability

Replaceability

Characteristics of Software in Software Engineering :-

1) Software is created, not manufactured:

- While software development and hardware manufacturing have some things in common, they are quite different
- In both cases, good design is important for high quality, but in hardware, the manufacturing process can introduce new problems that don't happen in software development.

2) Software doesn't "wear out":

- Hardware eventually gets old and stops working due to environmental factors like heat or physical wear
- When hardware breaks, you replace parts There's no such thing as "spare parts" for software.

- If software fails, it's usually due to a mistake in the design or the way the code was written. Fixing software problems can be more complex than fixing hardware but the key point is that software doesn't wear out. However, it can become outdated or inefficient over time.

3) Software is still custom-built :-

- Software should be designed so that parts of it can be reused in other projects.

- Reusable software parts often include both the data and the logic needed to handle the data, allowing developers to build new programs faster.

- In hardware, reusing components is a normal part of the engineering process, and software development is moving toward this too, but it's still often custom-built

Characteristics of Software :-

(i) Not tangible :- You can't see or touch software; it's just a code running on a computer

(ii) It doesn't expire :- Unlike physical items, software doesn't age or

degrade over time.

(iii) Easy to copy:- You can make multiple copies of software and share them easily

(iv) It can be complex:- Software can have many parts and features that work together, making it hard to build.

(v) Not always easy to understand or change:- Especially with large systems, it can be tough to figure out how everything works or to make updates.

(vi) Changes with user Needs:- As user's needs change, software often needs to be modified to keep up.

(vii) It can have errors:- Software may contain bugs or issues, so testing and fixing are required to make sure it works correctly.

Software crises:-

The term software crises refers to a set of problems that arise during the ~~(early)~~ early development of software engineering

in the 1960s and 1970s, but the issues are still revelant today. ~~Let's~~ In software development, the software crises refers to situations where:
- Projects take longer than expected to finish
- Costs exceed initial budgets
- The software is unreliable, meaning it crashes often or has bugs
- It doesn't meet the needs of the users
- It's difficult to maintain or upgrade.

⇒ Causes of Software Crises:

The software crises occured because software development was growing more complex, but the methods to handle that complexity were not advanced enough. Some major reasons include:-

- Growing demand:- More companies needed software, but there weren't enough skilled programmers.
- Complexity Increase:- Software was becoming more intricate making it harder to manage and predict outcomes
- Lack of planning:- Software development lacked the structured planning and strategies we have today.

Date:
Page No.:

- Changing Requirements:- Customers or stakeholders often changed their requirements midway, making it hard to adapt.

⇒ Problems it Caused:

- Late Delivery:- Projects were often completed much later than promised.
- Unmanageable Code:- The code became messy and unmanageable, making it difficult for anyone else to understand or modify
- Buggy Software:- Software was full of errors, which led to malfunctions, crashes, and even system failures.
- Increased Costs:- Due to delays and constant rework, the cost of software development skyrocketed

⇒ How was it Addressed?:

- Software Engineering Principles:- Structured methods for planning, designing, testing, and maintaining software
- Project Management Practices:- Tools to track project progress, timelines, and resources

Date:
Page No.:

- Agile Development:- A flexible approach where software is built incrementally, allowing for adjustments based on feedback

- Improved Testing:- Automated testing and debugging tools to catch errors early.

⇒ Some Examples:-

1) Y2K Problem:- It was a computer bug that occured because many older computers and software stored years using only last two digits (i.e., '86' for 1986). It was a problem 'cause that would've meant when the century changes it might read the dates wrong (i.e., it would read '2001' as '01' which could be similar to '1901'). Therefore calculation errors and system crashes would've occured if the error continued.

2) Patriot Missile Problem:- In "Star Wars" program of "USA, "The Patriot Missile" was a defence system used to detect and destroy incoming enemy missile. The problem happenned because of a tiny software bug in the system's clock. Over time, this tiny error caused the system to miscalculate the position of incoming

Date:

Page No.:

missiles hence causing 28 deaths and injuring over 100 soldiers of USA Army.

2) Waterfall Model :-

It is a traditional way of developing software that works like a step-by-step process. You move through each stage one step at a time and never go back.

- **Principle:-** It follows a linear sequence, where you move from one stage to the next in a straight line, like water flowing down a waterfall. The stages are:-

(i) **Requirements:-** Gather all the information about what the software should

do.

(ii) Design:- Plan out how the software will work

(iii) Implementation:- Start coding and building the software.

(iv) Testing:- check for bugs and errors

(v) Deployment:- Release the software to the users.

(vi) Maintenance:- Fix issues and keep the software running smoothly.

Each stage must be completed and signed off before moving on to the next one. There's no going back to a previous stage unless you start all over again.

• Advantages:- (i) Easy to Understand:- The step-by-step approach is straightforward and simple to follow.

(ii) Clear Documentation:- Each stage produces clear documents that help everyone know what's going on.

(iii) Structured Approach:- Everything is planned in advance, reducing

confusing and making tracking progress easier.

(iv) Works for Smaller Projects :- It's good for smaller projects with very clear and unchanging requirements.

• Disadvantages :- (i) Rigid and Inflexible :- Once you move to the next stage, it's very hard to go back and make changes

(ii) Poor handling of Changing Requirements If customer needs change midway, it's difficult and expensive to update the software.

(iii) Late Testing :- Testing happens only at the end so (~~it's difficult and expensive to update the software~~) if there's a major problem, it's discovered too late.

(iv) Not ideal for Complex Projects :- For big or unclear projects, it's hard to foresee all the requirements upfront, making this model unsuitable.

3) Increment Process Model :-

It is a way of developing software piece by piece Instead of building the whole software

in one go, you build it in small parts, called increments. Each part adds more features and functionality until the full software is complete

(i) Iterative Enhancement Model:-

It is based on building and improving the software repeatedly in cycles

(i) Start with a Basic Version:- Create a simple, functional version of the software.

(ii) Get Feedback:- Show it to users and get their thoughts.

(iii) Add Features:- Based on feedback, add more features or fix issues.

(iv) Repeat:- continue this process until the software is fully built.

• Advantages:- (i) Early Delivery:- Users get to see a working product early, even if it's basic.

(ii) Easier to Manage:- Since changes are made in small steps, each iteration is easier to manage and track.

(iii) Accommodates changes:- Easy to incor-porate changes based on user feedback after each iteration.

(iv) Improved Quality:- Regular testing after each cycle improves overall quality.

→ Disadvantages:- (i) Requires Good Planning:- If the foundation isn't strong, each iteration can become problematic.

(ii) Scope Creep:- Constant Additions can cause the project to grow beyond its original scope.

(iii) Time-consuming:- If there are too many iterations, it can take longer to complete the project.

(ii) RAD (Rapid Application Development) Model:-

It emphasizes speed and quick delivery. The idea is to build software rapidly by using prototyping, small teams, and fast iterations.

(i) Requirements Planning:- Quickly gather basic requirements and understand what's needed

(ii) Prototype Building:- Create a quick

version of the software (a prototype) in days or weeks, not months.

(iii) User Feedback:- Show it to users, get their feedback, and make improvements

(iv) Refinement and Finalization:- Keep refining the prototype until it meets user expectations, and then finalize it as the complete product.

- Advantages:-(i) Very Fast Development:- Suitable for projects with tight deadlin-es.

(ii) User Involvement:- Users get to see the product early and provide feedback.

(iii) Reduced Rework:- Early user feedback means changes are made before a lot of time is spent on coding

(iv) Flexibility:- Requirements can be adjusted quickly based on user responses.

- Disadvantages:-(i) Not suitable for Large Projects:- It works well for small to medium-sized projects but struggles with large, complex systems.

(ii) Needs Highly Skilled Teams:- RAD requires teams that can quickly adapt and build prototypes fast.

(iii) Can be expensive:- If changes keep happening, the cost can rise

(iv) Not ideal for long-term Projects:- If the project stretches out, the focus on quick delivery can lead to quality issues.

4) Evolutionary Process Model :-

It is a software development approach that focuses on gradually refining and improving a system through multiple iterations

(i) Prototype Model:-

The idea is to build something quickly (a prototype) which demonstrates the basic features, allowing users to interact with it and provide input on what works and what doesn't

(i) Requirement Gathering:- Identify basic requirements for the software

(ii) Prototype Development:- Build a

simple version of the software that includes the key features.

(iii) User Feedback:- Present the prototype to users and gather their feedback

(iv) Refine and Repeat:- Use the feedback to improve the prototype, adding features and making changes until the final product is ready.

- Advantages:-(i) User Involvement:- Users can see and interact with a working model early on, leading to better feedback

(ii) Clear Requirement:- Helps clarify what users really want, as they can see a tangible example

(iii) Reduces Risk:- Identifying issues early reduces the risk of significant changes later.

(iv) Improved Quality:- Continuous refinement based on user feedback improves overall product quality.

- Disadvantages:-(i) Not a complete product:- The prototype may not include all features leading to misunderstandings about the final product.

(ii) User Expectations :- Users may assume the prototype is closer to the final product than it really is, leading to disappointment.

(iii) Time-consuming :- Multiple iterations may take time, especially if feedback leads to major changes.

(iv) Scope Creep :- The process can lead to adding too many features, making it hard to manage the project.

(ii) Spiral Model :-

It combines elements of both the Prototype Model and the traditional Waterfall Model. It emph-asizes a risk driven approach to development through repeated cycles.

(i) Planning :- Define objectives and requirements for that cycle.

(ii) Risk Analysis :- Identify potential risks and develop strategies to manage them

(iii) Engineering :- Develop and implement the product for that cycle

(iv) Evaluation :- Review the product

and gather user feedback before starting the next cycle

- Advantages:- (i) Risk Management:- Continuous evaluation of risks allows for better decision-making

(ii) Flexibility:- You can adjust features and requirements in each cycle based on user feedback.

(iii) User Feedback:- Regular feedback helps ensure the product meets user needs.

(iv) Early Detection of Problems:- Potential issues can be identified and resolved early in the development process

- Disadvantages:- (i) Complexity:- The model can be complicated to manage due to its iterative nature and multiple phases.

(ii) High Cost:- Because of the emphasis on risk assessment and prototyping, it can be more expensive than other models

(iii) Time-consuming:- The iterative cycles may prolong the overall development time.

(iv) Requires Expertise:- It requires skilled project managers to navigate through the

various cycles effectively

5) Agile :-

It is a flexible and collabor-ative approach to software development that focuses on delivering small, working pieces of software quickly and iteratively

(i) Customer Collaboration :- Working closely with users to understand their needs and get feedback regularly

(ii) Iterative Development :- Building software in small increments or iterations, usually lasting a few weeks.

(iii) Adaptive Planning :- Being flexible to changes in requirements, even late in the development process.

(iv) Continuous Improvement :- Regularly reflecting on the process and making adjustments to improve efficiency and quality.

• Advantages :- (i) Flexibility to change :- Agile allows for adjustments at any stage of development, making it easier to respond to changing user needs.

(ii) Faster Delivery :- With small increments

users can see and use parts of the software earlier, rather than waiting for the final product.

(iii) Better Quality:- Regular testing and feedback help catches issues early, leading to a more reliable product.

(iv) Increased Collaboration:- Team members work closely together and communicate often, which can lead to better teamwork and ideas

(v) Customer Satisfaction:- Continuous involvement of users leads to a product that better meets their expectations

• Disadvantages:- (i) Less Predictability:- The flexibility of Agile can make it hard to predict when the project will be finished and how much it will cost.

(ii) Requires Strong Team Dynamics:- Success relies heavily on good communication and collaboration, if the team isn't cohesive, it can lead to problems.

(iii) Can be Disorganised:- Without clear documentation and structure, Agile projects can sometimes feel chaotic and unfocused.

(iv) Not ideal for All Projects:- Agile may not be suitable for very large or complex projects with rigid requirements.

(v) Time-Intensive:- Frequent meetings and collaboration can be time-consuming, potentially slowing down progress if not managed well.

Software Quality:-

It refers to how well and reliably a software product works.

⇒ Key Factors of Software Quality:-

(i) Portability:- The ability of software to work on different systems and devices. For eg:- Imagine a notepad app that runs smoothly on a windows PC, a MacBook, and an Android tablet. This means it is ~~pro~~ portable because it doesn't separate versions for each device.
Benefit:- Users can use the software on multiple (~~software~~) platforms without needing different versions, and developers don't have to create separate apps for each device.

(ii) Reusability:- The ability to reuse parts of the software for creating new applications. For eg:- If you created a simple calculator

~~your~~ app and later used the same code to develop a more advanced scientific calculator you're reusing the basic features like addition and subtraction instead of writing new code.
Benefit:- Saves time and effort, and reduces the chances of errors in the reused parts.

(iii) Usability:- How easy and intuitive it is for different users (both beginners and experts) to use the software. For eg:- Think of a photo editing app that's simple for a beginner to use for basic edits, but also offers advanced features for professional photographers.
Benefit:- Beginners can quickly do ~~(quickly)~~ simple tasks, while advanced users can explore deeper functions - all within the same app.

(iv) Correctness:- The software's ability to meet all the requirements mentioned in its specification document. For eg:- If a library system is supposed to allow students to search for books and notify them about overdue books, and it does exactly that, then the software is correct
Benefit:- Correct software fulfills its intended purpose as described in its design document

v) Maintainability:- How easily the software can be fixed, updated, or improved. For eg:- If an online shopping site can quickly fix

checkout page (~~efforts~~) errors, add new payment methods, and redesign its homepage without affecting the entire website, it has good maintainability.
Benefit:- It becomes easier to correct errors add new features, or modify the software over time.

(vi) Reliability:- The ability of the software to function without failure and handle errors properly. For eg:- A banking app should be reliable so that it doesn't crash, it should recover and ensure no incorrect transactions happen.
Benefit:- Users can trust it to perform important operations without problems, and if something goes wrong, it won't impact users negatively.

By ensuring these qualities, software developers can create products that are effective, user-friendly, and long-lasting.

⇒ Software Quality Assurance (SQA):-

It is a process to make sure that the software meets the required standards and is of good quality. It involves checking and testing the software throughout its development to identify and fix any issues early on.

Key Elements of SQA:-

(i) Standards:-
- Ensure the software follows guidelines set by organizations like ISO and IEEE

(ii) Reviews and Audits:-
- Regular checks by engineers and quality teams to find errors and ensure standards are being followed.

(iii) Testing:-
- Planned tests to identify bugs and errors before the software is released.

(iv) Error Analysis:-
- Collect and study data on errors to understand how they occurred and prevent them in the future

(v) Change Management:-
- Track and control any changes made to the software to avoid new problems.

(vi) Education:-
- Train the team to follow good software practices and improve their skills.

(vii) Security and Safety:-
- Make sure the software is safe to use and secure against threats

(viii) Risk Management:-
Identify potential risks early and plan solutions in advance.

Focus Areas of SQA:-

- Portability:- Software should work on different devices without needing a lot of changes.
- Usability:- It should be easy for users to navigate and use.
- Reusability:- Software components should be reusable in future projects.
- Correctness:- It should function as expected without errors.
- Maintainability:- Easy to update or fix over time.
- Error Control:- Detect and handele errors smoothly.

SQA Activities:-

(i) SQA Management Plan:- Make a plan for how quality assurance will be done

(ii) Set Checkpoint:- Define points to evaluate

the project's quality.

(iii) Measure Change Impact:- Track the effect of changes to avoid new issues.

(iv) Multi-testing strategy:- Use different testing methods to catch all possible errors.

(v) Maintain Good Relations:- Ensure the SQA team has a positive working relationship with other teams

(vi) Keep Records and Reports:- Document test results, defects, and changes for future refrence.

Benefits of SQA:-

(i) Produces high-quality software.

(ii) Saves time and cost in the long run

(iii) Reduces maintenance needs

(iv) Increases customer trust and market share

(v) Improves software reliability.

Disadvantages of SQA:-

(i) Costly:- Requires more resources and skilled

personnel

(ii) Time-consuming :- Testing can delay project deadlines.

(iii) Complexity :- Managing quality for large projects can be challenging.

(iv) Resistance to Change :- Some team members may find SQA process unnecessary.

(v) Not Foolproof :- Even with SQA, software might still have hidden defects.

Software Process (~~Compaba~~) Capability Maturity Model (CMM) :-

The CMM is a framework that helps software compdnies improve their development process step-by-step, resulting in a better quality software and more efficient workflows

Levels of CMM :-

(i) Level 1 : Initial (Chaotic)

- No formal process exists
- Developers do things their own way leading to ~~up~~ unpredictable results
- Success depends on individual rather than a clear process.

Date:
Page No.:

(ii) Level 2: Repeatable (Basic Management)
- Basic project management processes are established
- Successful practices are repeated in similar projects
- Teams start tracking tasks, timelines, and roles, ensuring more control over project outcomes.

(iii) Level 3: Defined (Standardized)
- Processes are well-documented and used by everyone in the organization
- Standard methods for coding, testing, and delivery are followed.
- Reduces errors and ensures consistency across projects.

(iv) Level 4: Managed (Measured and Controlled):
- Data is collected to measure the performance of each project.
- The company can predict project timelines and software quality based on collected metrics
- Decisions are based on data, not guesswork

(v) Level 5: Optimizing (Continuous Improvement)
- The company continuously refines and optimizes its processes
- Focuses on using data to innovate and make things even better

Date:

Page No.:

- Always looks for new ways to reduce costs, improve quality, and stay ahead of the competition.