6) Software Project Management

6.1 The management spectrum-4P's

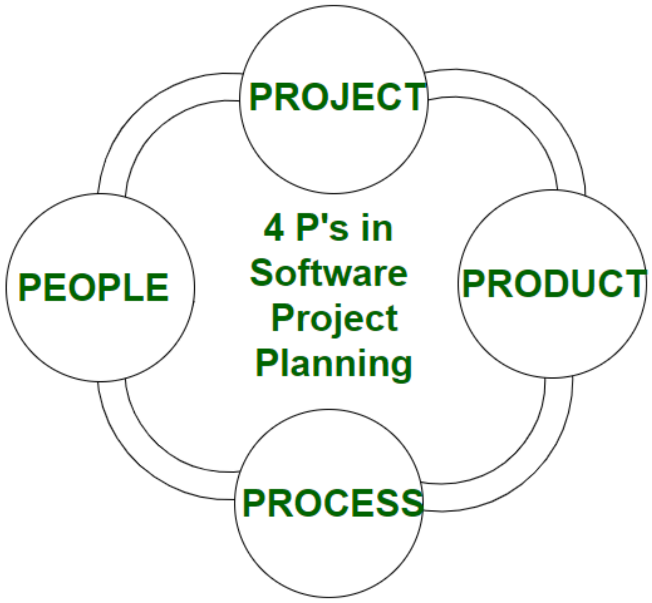


Figure: The management spectrum-4P's

People:

☐ **The Stakeholders**

The software process (and every software project) is populated by stakeholders who can be categorized into one of five constituencies:

1. Senior managers who define the business issues that often have a significant influence on the project.
2. Project (technical) managers who must plan, motivate, organize, and control the practitioners who do software work.
3. Practitioners who deliver the technical skills that are necessary to engineer a product or application.
4. Customers who specify the requirements for the software to be engineered and other stakeholders who have a peripheral interest in the outcome.
5. End users who interact with the software once it is released for production use.

☐ **Team Leader:**

In an excellent book of technical leadership, Jerry Weinberg [Wei86] suggests an MOI model of leadership:

- Motivation. The ability to encourage (by "push or pull") technical people to produce to their best ability.
- Organization. The ability to mold existing processes (or invent new ones) that will enable the initial concept to be translated into a final product.
- Ideas or innovation. The ability to encourage people to create and feel creative even when they must work within bounds established for a particular software product or application.

Another view [Edg95] of the characteristics that define an effective project manager emphasizes four key traits:

- Problem solving. An effective software project manager can diagnose the technical and organizational issues that are most relevant, systematically structure a solution or properly motivate other practitioners to develop the solution, apply lessons learned from past projects to new situations, and remain flexible enough to change direction if initial attempts at problem solution are fruitless.

- Managerial identity. A good project manager must take charge of the project. She must have the confidence to assume control when necessary and the assurance to allow good technical people to follow their instincts.
- Achievement. A competent manager must reward initiative and accomplishment to optimize the productivity of a project team. She must demonstrate through her own actions that controlled risk taking will not be punished.
- Influence and team building. An effective project manager must be able to "read" people; she must be able to understand verbal and nonverbal signals and react to the needs of the people sending these signals. The manager must remain under control in high-stress situations.

☐ **The Software Team**

Constantine [Con93] suggests four "organizational paradigms" for software engineering teams:

1. A closed paradigm structures a team along a traditional hierarchy of authority. Such teams can work well when producing software that is quite similar to past efforts, but they will be less likely to be innovative when working within the closed paradigm.
2. A random paradigm structures a team loosely and depends on individual initiative of the team members. When innovation or technological breakthrough is required, teams following the random paradigm will excel. But such teams may struggle when "orderly performance" is required.
3. An open paradigm attempts to structure a team in a manner that achieves some of the controls associated with the closed paradigm but also much of the innovation that occurs when using the random paradigm. Work is performed collaboratively, with heavy communication and consensus-based decision making the trademarks of open paradigm teams. Open paradigm team structures are well suited to the solution of complex problems but may not perform as efficiently as other teams.
4. A synchronous paradigm relies on the natural compartmentalization of a problem and organizes team members to work on pieces of the problem with little active communication among themselves.

☐ **Agile Teams**

Cockburn and Highsmith note this when they write: "If the people on the project are good enough, they can use almost any process and accomplish their assignment. If they are not good enough, no process will repair their inadequacy—"people trump process" is one way to say this. However, lack of user and executive support can kill a project—"politics trump people." Inadequate support can keep even good people from accomplishing the job."

<u>The Product:</u>

☐ **Software Scope** The first software project management activity is the determination of software scope. Scope is defined by answering the following questions:

- Context. How does the software to be built fit into a larger system, product, or business context, and what constraints are imposed as a result of the context?
- Information objectives. What customer-visible data objects are produced as output from the software? What data objects are required for input?
- Function and performance. What function does the software perform to transform input data into output? Are any special performance characteristics to be addressed?

☐ **Problem Decomposition**

Problem decomposition, sometimes called partitioning or problem elaboration, is an activity that sits at the core of software requirements analysis. During the scoping activity no attempt is made to fully decompose the problem. Rather, decomposition is applied in two major areas:

(1) the functionality and content (information) that must be delivered and

(2) the process that will be used to deliver it.

The Process

Melding the Product and the Process

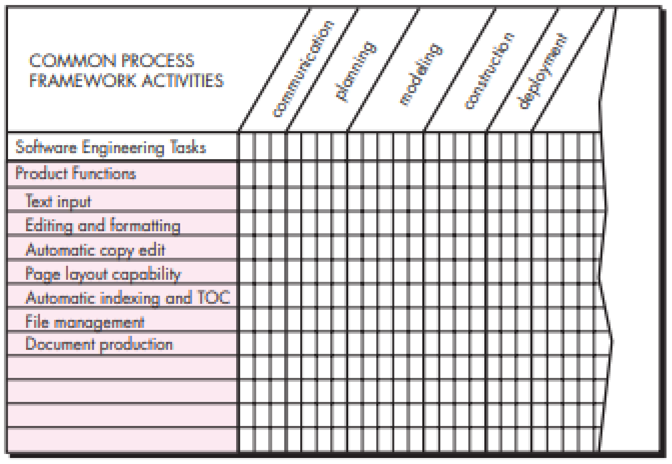


Figure: Melding the problem and the process

☐ **Process Decomposition**

- A software team should have a significant degree of flexibility in choosing the software process model that is best for the project and the software engineering tasks that populate the process model once it is chosen. A relatively small project that is similar to past efforts might be best accomplished using the linear sequential approach. If the deadline is so tight that full functionality cannot reasonably be delivered, an incremental strategy might be best
- Simple project might require the following work tasks for the communication activity:
  1. Develop list of clarification issues.
  2. Meet with stakeholders to address clarification issues.
  3. Jointly develop a statement of scope.
  4. Review the statement of scope with all concerned.
  5. Modify the statement of scope as required.
- a project might require the following work tasks for the communication:
  1. Review the customer request.
  2. Plan and schedule a formal, facilitated meeting with all stakeholders.
  3. Conduct research to specify the proposed solution and existing approaches.
  4. Prepare a "working document" and an agenda for the formal meeting.
  5. Conduct the meeting.
  6. Jointly develop mini-specs that reflect data, functional, and behavioral features of the software. Alternatively, develop use cases that describe the software from the user's point of view.
  7. Review each mini-spec or use case for correctness, consistency, and lack of ambiguity.
  8. Assemble the mini-specs into a scoping document.
  9. Review the scoping document or collection of use cases with all concerned.
  10. Modify the scoping document or use cases as required.

The Project

- In order to manage a successful software project, you have to understand what can go wrong so that problems can be avoided.
- In an excellent paper on software projects, John Reel [Ree99] defines 10 signs that indicate that an information systems project is in jeopardy:
  1. Software people don't understand their customer's needs.
  2. The product scope is poorly defined.
  3. Changes are managed poorly.
  4. The chosen technology changes.

5. Business needs change [or are ill defined].
6. Deadlines are unrealistic.
7. Users are resistant.
8. Sponsorship is lost [or was never properly obtained].
9. The project team lacks people with appropriate skills.
10. Managers [and practitioners] avoid best practices and lessons learned.

6.2 Metrics for Size Estimation: Line of Code (LoC), Function Points (FP).

Define Size Estimation. Enlist its Technique

1. Estimation of the size of software is an essential part of Software Project Management. It helps the project manager to further predict the effort and time which will be needed to build the project. Various measures are used in project size estimation.
2. Some of these are:
   - Lines of Code
   - Number of entities in ER diagram
   - Total number of processes in detailed data flow diagram
   - Function points

Lines of Code (LOC)

1. As the name suggest, LOC count the total number of lines of source code in a project. The units of LOC are:
   - KLOC- Thousand lines of code
   - NLOC- Non comment lines of code
   - KDSI- Thousands of delivered source instruction
2. The size is estimated by comparing it with the existing systems of same kind. The experts use it to predict the required size of various components of software and then add them to get the total size.

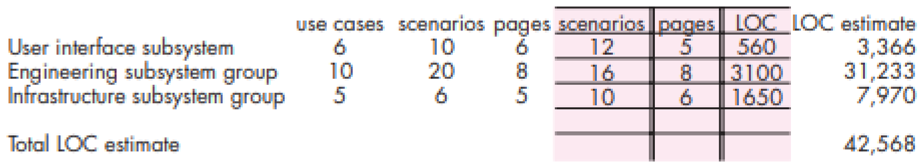
Example :

```
void selSort(int x[], int n) {
  //Below function sorts an array in ascending order
   int i, j, min, temp;
   for (i = 0; i < n - 1; i++) {
      min = i;
      for (j = i + 1; j < n; j++)
      if (x[j] < x[min])
      min = j;
      temp = x[i];
      x[i] = x[min];
      x[min] = temp;
   }
}
```

1. So, now If LOC is simply a count of the number of lines then the above function shown contains 13 lines of code (LOC).

2. But when comments and blank lines are ignored, the function shown above contains 12 lines of code (LOC).

#Use-case estimation



	use cases	scenarios	pages	scenarios	pages	LOC	LOC estimate
User interface subsystem	6	10	6	12	5	560	3,366
Engineering subsystem group	10	20	8	16	8	3100	31,233
Infrastructure subsystem group	5	6	5	10	6	1650	7,970
Total LOC estimate							42,568

Advantages:

1. Universally accepted and is used in many models like COCOMO.
2. Estimation is closer to developer's perspective.
3. Simple to use.

Disadvantages:

1. Difficult to measure LOC in the early stages of a new product.
2. Source instructions vary with coding languages, design methods and with programmer's ability.
3. No industry standard for measuring LOC.
4. LOC cannot be used for normalizing if platforms and languages are different.
5. The only way to predict LOC for a new app to be developed is through analogy based on similar software applications.
6. Programmers may be rewarded for writing more LOC based on a misconception of higher management by thinking that more the LOC, means more the productivity of the programmer.

Function Point (FP)

1. A Function Point (FP) is a unit of measurement to express the amount of business functionality an information system (as a product) provides to a user.
2. FPs measure software size. They are widely accepted as an industry standard for functional sizing.

#Uses:

1. The function point (FP) metric can be used effectively as a means for measuring the functionality delivered by a system.
2. Using historical data, the FP metric can then be used to (1) estimate the cost or effort required to design, code, and test the software; (2) predict the number of errors that will be encountered during testing; and (3) forecast the number of components and/or the number of projected source lines in the implemented system.
3. Function points are derived using an empirical relationship based on countable (direct) measures of software's information domain and qualitative assessments of software complexity.

#Information domain values are defined in the following manner:

1. **Number of external inputs (EIs):** Each external input originates from a user or is transmitted from another application and provides distinct application-oriented data or control information. Inputs are often used to update internal logical files (ILFs). Inputs should be distinguished from inquiries, which are counted separately.
2. **Number of external outputs (EOs):** Each external output is derived data within the application that provides information to the user. In this context external output refers to reports, screens, error messages,

etc. Individual data items within a report are not counted separately. Number of external inquiries (EQs). An external inquiry is defined as an online input that results in the generation of some immediate software response in the form of an online output (often retrieved from an ILF).

3. **Number of internal logical files (ILFs):** Each internal logical file is a logical grouping of data that resides within the application's boundary and is maintained via external inputs.
4. **Number of external interface files (EIFs):** Each external interface file is a logical grouping of data that resides external to the application but provides information that may be of use to the application.

#For sizing software based on FP, several recognized standards and/or public specifications have come into existence. As of 2013, these are −

ISO Standards

• **COSMIC −** ISO/IEC 19761:2011 Software engineering. A functional size measurement method.

• **FiSMA −** ISO/IEC 29881:2008 - Software and systems engineering - FiSMA 1.1 functional size measurement method.

• **IFPUG − I**SO/IEC 20926:2009 Software and systems engineering - Software measurement - IFPUG functional size measurement method.
• **Mark-II −** ISO/IEC 20968:2002 Software engineering - Ml II Function Point Analysis - Counting Practices Manual.

• **NESMA −** ISO/IEC 24570:2005 Software engineering - NESMA function size measurement method version 2.1 -

#Definitions and counting guidelines for the application of Function Point Analysis.

1. Function Point Analysis (FPA) technique quantifies the functions contained within software in terms that are meaningful to the software users.
2. FPs consider the number of functions being developed based on the requirements specification.
3. Function Points (FP) Counting is governed by a standard set of rules, processes and guidelines as defined by the International Function Point Users Group (IFPUG).
4. These are published in Counting Practices Manual (CPM).

#Elementary Process (EP)

Elementary Process is the smallest unit of functional user requirement that −

1. Is meaningful to the user.
2. Constitutes a complete transaction.
3. Is self-contained and leaves the business of the application being counted in a consistent state.

- ☐ **Functions:-**

  There are two types of functions −

  • Data Functions

  • Transaction Functions

- ☐ **Data Functions:-**

  There are two types of data functions −

  • Internal Logical Files

  • External Interface Files

#Data Functions are made up of internal and external resources that affect the system.

- ☐ **Internal Logical Files**

Internal Logical File (ILF) is a user identifiable group of logically related data or control information that resides entirely within the application boundary. The primary intent of an ILF is to hold data maintained through one or more elementary processes of the application being counted. An ILF has the inherent meaning that it is internally maintained, it has some logical structure and it is stored in a file.

☐ **External Interface Files**

External Interface File (EIF) is a user identifiable group of logically related data or control information that is used by the application for reference purposes only. The data resides entirely outside the application boundary and is maintained in an ILF by another application. An EIF has the inherent meaning that it is externally maintained, an interface has to be developed to get the data from the file.

6.3 COCOMO (Constructive Cost Model)

Constructive Cost Model (COCOMO)

- COCOMO is one of the most widely used software estimation models in the world.
- This model is developed in 1981 by Barry Boehm to give estimation of number of man-months it will take to develop a software product.
- COCOMO predicts the efforts and schedule of software product based on size of software.
- COCOMO has three different models that reflect complexity
  1. **Basic Model**
  2. **Intermediate Model**
  3. **Detailed Model**
- **Similarly, there are three classes of software projects.**
  1. Organic mode In this mode, relatively simple, small software projects with a small team are handled. Such team should have good application experience to less rigid requirements.
  2. Semi-detached projects In this class intermediate project in which team with mixed experience level are handled. Such project may have mix of rigid and less than rigid requirements.
  3. Embedded projects In this class, project with tight hardware, software and operational constraints are handled.

#Each Model in detail

1. Basic Model

The basic COCOMO model estimate the software development effort using only Lines of code
Various equations in this model are

$$E = a_b KLOC_b^b$$
$$D = c_b E_b^d$$

Where,

- **E is the effort applied in person-months,**
- **D is the development time in chronological months and**
- **KLOC is the estimated number of delivered lines of code for the project**

2. Intermediate Model
This is extension of COCOMO model.
This estimation model makes use of set of “Cost Driver Attributes” to compute the cost of software**.**

- I. Product attributes
  - a. required software reliability
  - b. size of application data base
  - c. complexity of the product
- II. Hardware attributes
  - a. run-time performance constraints
  - b. memory constraints
  - c. volatility of the virtual machine environment
  - d. required turnaround time
- III. Personnel attributes
  - a. analyst capability
  - b. software engineer capability
  - c. applications experience
  - d. virtual machine experience
  - e. programming language experience
- IV. Project attributes
  - a. use of software tools
  - b. application of software engineering methods
  - c. required development schedule

Each of the 15 attributes is rated on a 6 point scale that ranges from "very low" to "extra high" (in importance or value).
The intermediate COCOMO model takes the form

$$E = a_i KLOC_i^b \times EAF$$

Where,

- **E is the effort applied in person-months and**
- **KLOC is the estimated number of delivered lines of code for the project**

3. Detailed COCOMO Model
The detailed model uses the same equation for estimation as the intermediate Model.
But detailed model can estimate the effort (E), duration (D), and person (P) of each of development phases, subsystem and models.

6.4 Risk Management: Risk Identification, Risk Assessment, RMMM Strategy.

Software Risk:

1. Although there has been considerable debate about the proper definition for software risk, there is general agreement that risk always involves two characteristics: uncertainty—the risk may or may not happen; that is, there are no 100 percent. quote: “If you don't actively attack the risks, they will actively attack you.” Tom

Gilb probable risks1 —and loss—if the risk becomes a reality, unwanted consequences or losses will occur [Hig95].
2. When risks are analyzed, it is important to quantify the level of uncertainty and the degree of loss associated with each risk. To accomplish this, different categories of risks are considered

Risk Types:

1. **Project risks** threaten the project plan. That is, if project risks become real, it is likely that the project schedule will slip and that costs will increase. Project risks identify potential budgetary, schedule, personnel (staffing and organization), resource, stakeholder, and requirements problems and their impact on a software project.
2. **Technical risks** threaten the quality and timeliness of the software to be produced. If a technical risk becomes a reality, implementation may become difficult or impossible. Technical risks identify potential design, implementation, interface, verification, and maintenance problems. In addition, specification ambiguity, technical uncertainty, technical obsolescence, and "leading-edge" technology are also risk factors. Technical risks occur because the problem is harder to solve than you thought it would be.
3. **Business risks** threaten the viability of the software to be built and often jeopardize the project or the product. Candidates for the top five business risks are (1) building an excellent product or system that no one really wants (market risk), (2) building a product that no longer fits into the overall business strategy for the company (strategic risk), (3) building a product that the sales force doesn't understand how to sell (sales risk), (4) losing the support of senior management due to a change in focus or a change in people (management risk), and (5) losing budgetary or personnel commitment (budget risks). It is extremely important to note that simple risk categorization won't always work. Some risks are simply unpredictable in advance.
4. Another general categorization of risks has been proposed by Charette [Cha89]. **Known risks** are those that can be uncovered after careful evaluation of the project plan, the business and technical environment in which the project is being developed, and other reliable information sources (e.g., unrealistic delivery date, lack of documented requirements or software scope, poor development environment).
5. **Predictable risks** are extrapolated from past project experience (e.g., staff turnover, poor communication with the customer, dilution of staff effort as ongoing maintenance requests are serviced).
6. **Unpredictable risks** are the joker in the deck. They can and do occur, but they are extremely difficult to identify in advance

Risk Management:

Seven Principles of Risk Management

The Software Engineering Institute (SEI) (www.sei.cmu.edu) identifies seven principles that "provide a framework to accomplish effective risk management." They are:

1. **Maintain a global perspective—**view software risks within the context of a system in which it is a component and the business problem that it is intended to solve.
2. **Take a forward-looking view—**think about the risks that may arise in the future (e.g., due to changes in the software); establish contingency plans so that future events are manageable.
3. **Encourage open communication—**if someone states a potential risk, don't discount it. If a risk is proposed in an informal manner, consider it. Encourage all stakeholders and users to suggest risks at any time.
4. **Integrate—**a consideration of risk must be integrated into the software process.
5. **Emphasize a continuous process—**the team must be vigilant throughout the software process, modifying identified risks as more information is known and adding new ones as better insight is achieved.
6. **Develop a shared product vision—**if all stakeholders share the same vision of the software, it is likely that better risk identification and assessment will occur.
7. **Encourage teamwork—**the talents, skills, and knowledge of all stakeholders should be pooled when risk management activities are conducted.

Risk Identification:

- Risk identification is a systematic attempt to specify threats to the project plan (estimates, schedule, resource loading, etc.).
- By identifying known and predictable risks, the project manager takes a first step toward avoiding them when possible and controlling them when necessary.
- **There are two distinct types of risks for each of the categories that have been presented**
    1. **Generic risks** and product-specific risks. Generic risks are a potential threat to every software project.
    2. **Product-specific risks** can be identified only by those with a clear understanding of the technology, the people, and the environment that is specific to the software that is to be built. To identify product-specific risks, the project plan and the software statement of scope are examined, and an answer to the following question is developed: "What special characteristics of this product may threaten our project plan?"

One method for identifying risks is to create a risk item checklist. The checklist can be used for risk identification and focuses on some subset of known and predictable risks in the following generic subcategories:

1. **Product size**—risks associated with the overall size of the software to be built or modified.
2. **Business impact**—risks associated with constraints imposed by management or the marketplace.
3. **Stakeholder characteristics**—risks associated with the sophistication of the stakeholders and the developer's ability to communicate with stakeholders in a timely manner.
4. **Process definition**—risks associated with the degree to which the software process has been defined and is followed by the development organization.
5. **Development environment**—risks associated with the availability and quality of the tools to be used to build the product.
6. **Technology to be built**—risks associated with the complexity of the system to be built and the "newness" of the technology that is packaged by the system.
7. **Staff size and experience**—risks associated with the overall technical and project experience of the software engineers who will do the work.

Risk Assessment:

1. **Risk Identification -** produces lists of the project-specific risk items likely to compromise a project's success.
2. **Risk Analysis** - assesses the loss probability and loss magnitude for each identified item, and it assesses compound risks in risk-item interactions.
3. **Risk Prioritization -** produces a ranked ordering of the risk items identified and analyzed.

#Risk Control:-

1. **Risk Management Planning** - helps prepare to address each risk item, including the coordination of the individual risk-item plans with each other and with the overall project plan.
2. **Risk Resolution** - produces a situation in which risk items are eliminated or otherwise resolved.
3. **Risk Monitoring**- involves tracking the project's progress toward resolving its risk items and taking corrective action where appropriate.

The following questions have been derived from risk data obtained by surveying experienced software project managers in different parts of the world [Kei98]. The questions are ordered by their relative importance to the success of a project.

1. Have top software and customer managers formally committed to support the project?
2. Are end users enthusiastically committed to the project and the system/ product to be built?
3. Are requirements fully understood by the software engineering team and its customers?
4. Have customers been involved fully in the definition of requirements?
5. Do end users have realistic expectations?

6. Is the project scope stable?
7. Does the software engineering team have the right mix of skills?
8. Are project requirements stable?
9. Does the project team have experience with the technology to be implemented?
10. Is the number of people on the project team adequate to do the job?
11. Do all customer/user constituencies agree on the importance of the project and on the requirements for the system/product to be built?

RMMM Strategy

1. A risk management strategy can be defined as a software project plan or the risk management steps. It can be organized into a separate Risk Mitigation, Monitoring and Management Plan. The RMMM plan documents all work performed as part of risk analysis and is used by the project manager as part of the overall project plan.
2. Teams do not develop a formal RMMM document. Rather, each risk is documented individually using a risk information sheet . In most cases, the RIS is maintained using a database system, so that creation and information entry, priority ordering, searches, and other analysis may be accomplished easily.
3. Once RMMM has been documented and the project has begun, risk mitigation and monitoring steps commence. As we have already discussed, risk mitigation is a problem avoidance activity.

#Risk monitoring is a project tracking activity with three primary objectives:

(1) to assess whether predicted risks occur.
(2) to ensure that risk aversion steps defined for the risk are being properly applied; and
(3) to collect information that can be used for future risk analysis.

Effective strategy must consider three issues:

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

Proactive approach to risk – avoidance strategy.

1. Develop risk mitigation plan.

2. Develop a strategy to mitigate this risk for reducing turnover.

3. Meet with current staff to determine causes for turnover.

4. Mitigate those causes that are under our control before the project starts.

- Organize project teams so that information about each development activity is widely dispersed.
- Define documentation standards and establish mechanisms to be sure that documents are developed in a timely manner.Project manager monitors for likelihood of risk,Project manager should monitor the effectiveness of risk mitigation steps.Risk management and contingency planning assumes that mitigation efforts have failed and that the risk has become a reality.RMMM steps incur additional project cost.

THE RMMM PLAN

Risk Mitigation, Monitoring and Management Plan (RMMM) – documents all work performed as part of risk analysis and is used by the project manager as part of the overall project plan.RIS is maintained using a database system, so that creation and information entry, priority ordering, searches, and other analysis may be accomplished easily. Risk monitoring is a project tracking activity

Three primary objectives:

- assess whether predicted risks do, in fact, occur
- ensure that risk aversion steps defined for the risk are being properly applied
- collect information that can be used for future risk analysis.

6.5 DevOps testing methods.

Here are some DevOps testing methods [1] [2]:

1. **Automation Testing:** This method is used to reduce human intervention, verify the product's overall functionality and detect bugs.
2. **Chain Test:** This test confirms that all the apps on the chain are working well together.
3. **Component Test:** This test is applied to big applications that are built on different components.
4. **DevOps Unit Testing:** This method inspects individual components of the application to confirm the code they're built on works as expected.
5. **Functional Acceptance Test:** This test verifies if the happy path of the app is working according to the functional requirements.
6. **Integration Test:** This test works similar to the Component Test but can work on multiple components simultaneously.
7. **Performance Stress Test:** This test checks if the system can handle the request on time by a set number of users, background workload, and transactions.
8. **Production Acceptance Test:** This test verifies if the app can work fine in the target environment.
9. **System Test:** This test verifies if the app can fulfill the requirements.
10. **Unit Test:** This test examines a single small object.
11. **User Acceptance Test:** This test figures out if the user can use the app and whether it's user-friendly, effortless, and usable.

Notes: Kindly add minimum four points to each testing method based on marks