**What is Requirements Gathering?**

Requirements gathering is a crucial phase in the software development life cycle (SDLC) and project management. It involves collecting, documenting, and managing the requirements that define the features and functionalities of a system or application. The success of a project often depends on the accuracy and completeness of the gathered requirements in software.

**Main Requirements Gathering Subprocesses:**

Requirements gathering is a critical phase in the software development lifecycle, and it involves several subprocesses to ensure a comprehensive understanding of the project’s needs. The main subprocesses include:

**Stakeholder Identification:**

* **Objective:** Identify all stakeholders who will be affected by the system, directly or indirectly.
* **Process**: Conduct interviews, surveys, or workshops to determine the key individuals or groups involved.

**Stakeholder Analysis:**

* **Objective:** Understand the needs, expectations, and influence of each stakeholder.
* **Process:** Analyze stakeholder inputs to prioritize requirements and manage conflicting interests.

**Problem Definition:**

* **Objective:** Clearly define the problems or opportunities that the software system aims to address.
* **Process:** Engage stakeholders in discussions to uncover and articulate the core problems or opportunities.

**Requirements Extraction:**

* **Objective:** Gather detailed requirements by interacting with stakeholders.
* **Process:** Employ techniques such as interviews, surveys, observations, or brainstorming sessions to extract requirements.

**Requirements Documentation:**

* **Objective:** Document gathered requirements in a structured format.
* **Process:** Create requirements documents, use cases, user stories, or prototypes to capture and communicate requirements effectively.

**Validation and Verification:**

* **Objective:** Ensure that gathered requirements are accurate, complete, and consistent.
* **Process:** Conduct reviews, walkthroughs, or use validation tools to verify that the requirements meet the defined criteria.

**Processes of Requirements Gathering in Software Development:**

There are 6 steps crucial for requirement gathering processes



*Processes of Requirements Gathering in Software Development*

**Step 1- Assigning roles:**

* The first step is to identify and engage with all relevant stakeholders. Stakeholders can include end-users, clients, project managers, subject matter experts, and anyone else who has a vested interest in the software project. Understanding their perspectives is essential for capturing diverse requirements.

**Step 2- Define Project Scope:**

* Clearly define the scope of the project by outlining its objectives, boundaries, and limitations. This step helps in establishing a common understanding of what the software is expected to achieve and what functionalities it should include.

**Step 3- Conduct Stakeholder Interviews:**

* Schedule interviews with key stakeholders to gather information about their needs, preferences, and expectations. Through open-ended questions and discussions, aim to uncover both explicit and implicit requirements. These interviews provide valuable insights that contribute to a more holistic understanding of the project.

**Step 4- Document Requirements:**

* Systematically document the gathered requirements. This documentation can take various forms, such as user stories, use cases, or formal specifications. Clearly articulate functional requirements (what the system should do) and non-functional requirements (qualities the system should have, such as performance or security).

**Step 5- Verify and Validate Requirements:**

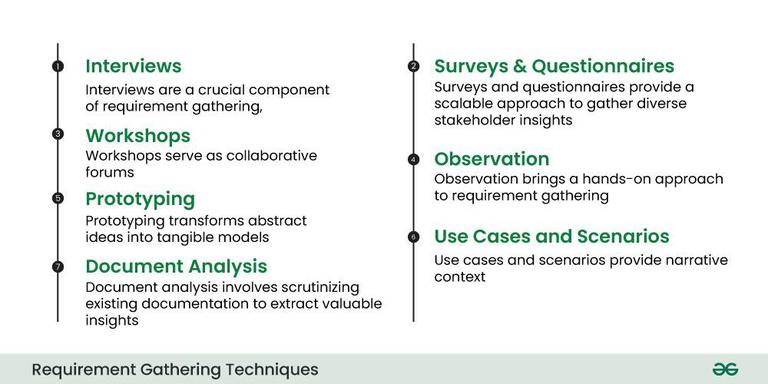
* Once the requirements are documented, it’s crucial to verify and validate them. Verification ensures that the requirements align with the stakeholders’ intentions, while validation ensures that the documented requirements will meet the project’s goals. This step often involves feedback loops and discussions with stakeholders to refine and clarify requirements.

**Step 6- Prioritize Requirements:**

* Prioritize the requirements based on their importance to the project goals and constraints. This step helps in creating a roadmap for development, guiding the team on which features to prioritize. Prioritization is essential, especially when resources and time are limited.

**Requirement Gathering Techniques:**

Effective requirement gathering is essential for the success of a software development project. Various techniques are employed to collect, analyze, and document requirements.



*Requirements Gathering Techniques*

Here are some commonly used requirement gathering techniques:

1. **Interviews:**
   * Conducting one-on-one or group interviews with stakeholders, including end-users, clients, and subject matter experts. This allows for direct interaction to gather detailed information about their needs, expectations, and concerns.
2. **Surveys and Questionnaires:**
   * Distributing surveys and questionnaires to a broad audience to collect information on a larger scale. This technique is useful for gathering feedback from a diverse set of stakeholders and can be particularly effective in large projects.
3. **Workshops:**
   * Organizing facilitated group sessions or workshops where stakeholders come together to discuss and define requirements. Workshops encourage collaboration, idea generation, and the resolution of conflicting viewpoints in a structured environment.
4. **Observation:**
   * Directly observing end-users in their work environment to understand their workflows, pain points, and preferences. Observational techniques help in uncovering implicit requirements that users might not explicitly state.
5. **Prototyping:**
   * Creating mockups or prototypes of the software to provide stakeholders with a tangible representation of the proposed system. Prototyping allows for early visualization and feedback, helping to refine requirements based on stakeholders’ reactions.
6. **Use Cases and Scenarios:**
   * Developing use cases and scenarios to describe how the system will be used in different situations. This technique helps in understanding the interactions between users and the system, making it easier to identify and document functional requirements.
7. **Document Analysis:**
   * Reviewing existing documentation, such as business process manuals, reports, and forms, to extract relevant information. This technique provides insights into the current processes and helps identify areas for improvement.

**Why Requirement Gathering is important?**

Requirement gathering holds immense importance in software development for several critical reasons:

1. **Clarity of Project Objectives:**
   * Requirement gathering sets the stage by defining and clarifying the objectives of the software project. It ensures that all stakeholders, including clients, users, and development teams, have a shared understanding of what needs to be achieved.
2. **Customer Satisfaction:**
   * Understanding and meeting customer needs is paramount for customer satisfaction. Requirement gathering allows developers to comprehend the expectations of end-users and clients, leading to the creation of a product that aligns with their desires and requirements.
3. **Scope Definition:**
   * Clearly defined requirements help in establishing the scope of the project. This delineation is crucial for managing expectations, avoiding scope creep (uncontrolled changes to project scope), and ensuring that the project stays on track.
4. **Reduced Misunderstandings:**
   * Ambiguities and misunderstandings are common sources of project failures. Requirement gathering facilitates clear communication between stakeholders, reducing the risk of misinterpretations and ensuring that everyone involved is on the same page.
5. **Risk Mitigation:**
   * Identifying and addressing potential issues at the requirements stage helps mitigate risks early in the development process. This proactive approach minimizes the chances of costly errors, rework, and delays later in the project life cycle.

**Benefits of Requirements Gathering:**

The benefits of effective requirements gathering in software development include:

* **Cost Reduction:** One of the primary benefits of effective requirements gathering is cost reduction. When requirements are well-defined and thoroughly understood at the beginning of a project, it minimizes the likelihood of costly changes and rework later in the development process.
* **Customer Satisfaction:**Clear and accurate requirements gathering directly contributes to customer satisfaction. When the end product aligns closely with the expectations and needs of the stakeholders, it enhances user experience and meets customer demands. This satisfaction is not only vital for the success of the current project but also contributes to positive relationships between the development team and clients, fostering trust and potential future collaborations.
* **Improved Communication:**Requirements gathering serves as a communication bridge between various stakeholders involved in a project, including developers, clients, users, and project managers. Miscommunication is a common source of project failures and delays. By clearly documenting and understanding requirements, the development team ensures that everyone involved has a shared vision of the project objectives, functionalities, and constraints.
* **Efficient Resource Utilization:** Thorough requirements gathering enables the efficient allocation and utilization of resources. Resources, including time, manpower, and technology, are finite and valuable. When requirements are well-defined, project teams can allocate resources more accurately, avoiding unnecessary expenditures or overcommitting resources to certain aspects of the project.
* **Enhanced Quality:** Well-documented requirements serve as the foundation for quality assurance throughout the development process. When the project team has a clear understanding of what needs to be achieved, they can establish quality standards and criteria from the outset. This clarity enables the implementation of effective testing strategies, ensuring that each aspect of the system is thoroughly evaluated against the specified requirements.
* **Risk Management:**Requirements gathering is a crucial component of effective risk management. By identifying potential risks early in the project, stakeholders can proactively address ambiguities, conflicting requirements, and other challenges that could pose a threat to the project’s success.
* **Accurate Planning:**Accurate project planning is dependent on a clear understanding of project requirements. When requirements are well-documented, project managers can create realistic schedules, milestones, and deliverables. This accurate planning is crucial for setting expectations, managing stakeholder timelines, and ensuring that the project progresses according to the established timeline.

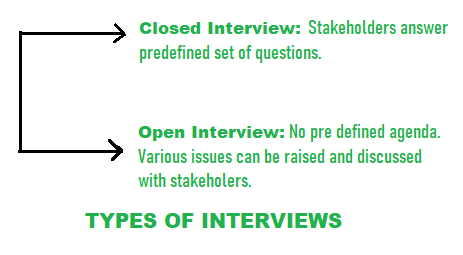
**Common Obstacles in Software Requirements Gathering:**

Common obstacles in software requirements gathering include:

* **Unclear Objectives:** Lack of clear project objectives can hinder requirements gathering. When stakeholders are unsure about what they want to achieve, it becomes challenging to define and prioritize requirements effectively. This can lead to confusion, scope creep, and difficulties in meeting project goals.
* **Ambiguous Requirements:**Ambiguities in requirements, such as vague language or conflicting statements, can create misunderstandings among stakeholders and the development team. Ambiguous requirements may result in deliverables that do not meet expectations and may require extensive rework.
* **Poor Stakeholder Involvement:** Insufficient involvement or engagement of key stakeholders can impede the requirements gathering process. When essential stakeholders are not actively participating or providing input, there is a risk of missing critical requirements or making decisions that do not align with the needs of the end-users.
* **Changing Requirements:** Requirements that undergo frequent changes during the development process, often referred to as “scope creep,” can lead to project delays, increased costs, and challenges in maintaining project focus. It is essential to manage and control changes to prevent unnecessary disruptions.
* **Communication Barriers:**Communication challenges, such as language barriers, misinterpretations, or inadequate channels for information exchange, can hinder effective requirements gathering. It is crucial to establish clear communication channels and ensure that all stakeholders have a shared understanding of the terminology used in the project.
* **Overreliance on Documentation:** Depending solely on documentation without active collaboration and communication can lead to misunderstandings. Written requirements may not capture the complete context or evolving needs, making it essential to complement documentation with interactive processes like workshops and interviews.
* **Lack of User Involvement:**Users are often the ultimate beneficiaries of the system, and their input is critical. Lack of user involvement or representation can result in systems that do not effectively meet their needs. It is important to actively involve end-users in the requirements gathering process to ensure the system’s usability and acceptance.

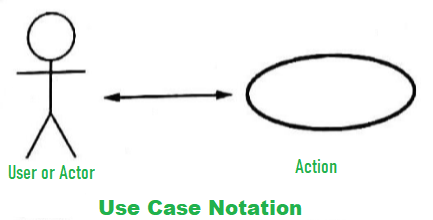
**The requirement gathering is process** of requirements discovery or generating list of requirements or collecting many requirements as possible by stakeholders. It is also called as requirements elicitation or requirement capture.

Various **Dimensions of requirements gathering** are:

1. **Stakeholders –**  
   The stakeholder means person with interest or concern in outcome of project who is affected by system.  
   For example- end-user, system maintenance engineer or administrator, software developer, direct user, indirect user, senior manager, etc. By collecting requirements from these stakeholders, understanding system requirements can be very easy.
2. **Interviewing –**  
   Interviewing is important and very effective method of requirement gathering. Different questions are being asked about system and its uses to stakeholders by team of requirement engineering so that identification of requirements can be done using these answers. There are two types of interviewers:
3. 

Interviewers are useful for understanding stakeholders but they are not much useful for understanding application domain.

1. **Use Case Modeling –**  
   Use cases are fundamental units of modeling language, in which functionalities are distinctly presented. The use case is scenario-based technique. Individual interactions with system can be identified using or with help of Use case model. Using of use case model is extremely done for requirements gathering. Important system requirements can be traced or identified by designing proper use cases for different scenarios. In use cases, typical notations used are:



The use case for ATM system is as shown below:

A diagram of a system

Description automatically generated

1. **Facility Application Specification Technique (FAST) –**  
   Facility Application Specification Technique is approach in which joint team of customers and developers work together to identify problem, propose elements of solution, negotiate different approaches and prepare specification for preliminary set of solutions requirements. Preparing- list of objects, list of services, and list of constraints are always asked each FAST attendee in every FAST meeting. Constraints of system such as cost, rules, memory requirement, speed accuracy, etc. are mentioned in list of constraints. As soon as FAST meeting begins, need and justification for new product is first issue that arises at point of discussion. Every participant has to present his lists once everyone agrees upon fact that product is justified.

**Functional Requirements**

These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract.

These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.

**Example:**

* What are the features that we need to design for this system?
* What are the edge cases we need to consider, if any, in our design?

**Non-Functional Requirements**

These are the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to another. They are also called non-behavioral requirements. They deal with issues like:

* Portability
* Security
* Maintainability
* Reliability
* Scalability
* Performance
* Reusability
* Flexibility

**Example:**

* Each request should be processed with the minimum latency?
* System should be highly valuable.

**Extended Requirements**

These are basically “nice to have” requirements that might be out of the scope of the System.

**Example:**

* Our system should record metrices and analytics.
* Service heath and performance monitoring.

**Difference between Functional Requirements and Non-Functional Requirements:**

| **Functional Requirements** | **Non Functional Requirements** |
| --- | --- |
| A functional requirement defines a system or its component. | A non-functional requirement defines the quality attribute of a software system. |
| It specifies “What should the software system do?” | It places constraints on “How should the software system fulfill the functional requirements?” |
| Functional requirement is specified by User. | Non-functional requirement is specified by technical peoples e.g. Architect, Technical leaders and software developers. |
| It is mandatory. | It is not mandatory. |
| It is captured in use case. | It is captured as a quality attribute. |
| Defined at a component level. | Applied to a system as a whole. |
| Helps you verify the functionality of the software. | Helps you to verify the performance of the software. |
| Functional Testing like System, Integration, End to End, API testing, etc are done. | Non-Functional Testing like Performance, Stress, Usability, Security testing, etc are done. |
| Usually easy to define. | Usually more difficult to define. |
| Example 1) Authentication of user whenever he/she logs into the system. 2) System shutdown in case of a cyber attack. 3) A Verification email is sent to user whenever he/she registers for the first time on some software system. | Example 1) Emails should be sent with a latency of no greater than 12 hours from such an activity. 2) The processing of each request should be done within 10 seconds 3) The site should load in 3 seconds when the number of simultaneous users are > 10000 |

**Shrink-wrapped software** typically refers to commercially available, pre-packaged software that is sold in physical packaging, such as a box or shrink-wrapped plastic. Here are some key differences and characteristics compared to other types of software distribution:

Distribution and Sales

Physical Distribution:

* Shrink-Wrapped: Sold in physical retail stores, online marketplaces, and other outlets in boxes or plastic wrapping.
* Other Types (e.g., Downloadable Software): Distributed digitally through the internet, allowing instant download and installation.

Packaging:

* Shrink-Wrapped: Comes with physical packaging, including a CD/DVD or USB stick, printed manuals, and possibly other materials like registration cards.
* Other Types: Typically, no physical packaging; users receive digital manuals and support documents.

Installation and Accessibility

Installation:

* Shrink-Wrapped: Requires physical media (CD/DVD/USB) for installation on a computer. May require an optical drive or USB port.
* Other Types: Installation files are downloaded directly from the internet. No need for physical media.

Updates and Patches:

* Shrink-Wrapped: May require manual installation of updates and patches via new physical media or downloadable updates.
* Other Types: Often features automatic updates, where the software checks for and installs updates from the internet seamlessly.

Licensing and Usage

License Management:

* Shrink-Wrapped: Typically involves a one-time purchase with a physical license key included in the packaging. License management is manual.
* Other Types: May use online activation, subscription models, or account-based licensing, allowing for easier management and renewal.

Copy Protection:

* Shrink-Wrapped: Often uses physical copy protection methods such as CD keys or hardware dongles.
* Other Types: Uses digital rights management (DRM) systems and online activation to prevent unauthorized copying.

Customer Experience

Immediate Access:

* Shrink-Wrapped: Users must go to a store or wait for shipping to receive the software. Installation can be delayed.
* Other Types: Immediate access to the software upon purchase and download.

Support and Documentation:

* Shrink-Wrapped: Includes physical manuals and support materials, which some users find more convenient.
* Other Types: Relies on online documentation, FAQs, and digital support, which may be more frequently updated.

Production Costs:

* Shrink-Wrapped: Higher production costs due to physical media, packaging, and distribution logistics.
* Other Types: Lower production costs as there are no physical materials; distribution is via digital download.

Environmental Impact:

* Shrink-Wrapped: Higher environmental impact due to production, packaging, and transportation.
* Other Types: Lower environmental impact as there is no physical waste or transportation required.

Market Trends:

* Shrink-Wrapped: Traditional model, still in use but declining as digital distribution becomes more prevalent.
* Other Types: Increasingly popular, especially with the rise of broadband internet, cloud computing, and software-as-a-service (SaaS) models.

Target Audience:

* Shrink-Wrapped: Often targets users who prefer physical copies, such as collectors or those in areas with limited internet access.
* Other Types: Targets users who prefer convenience, instant access, and the latest updates.

**Metrics for requirements phase**

Metrics for the requirements phase of a software development project are essential to ensure that the requirements are well-defined, understood, and feasible. Here are some key metrics that can be used to measure the effectiveness and quality of the requirements phase:

1. Requirement Quality Metrics

* Completeness: Measures the extent to which all necessary requirements have been identified and documented. Example Metric: Percentage of total requirements identified vs. those that should have been identified.
* Clarity: Assesses how clearly the requirements are stated, ensuring they are understandable by all stakeholders. Example Metric: Number of requirements needing clarification or rework.
* Consistency: Checks for contradictions or inconsistencies within the requirements. Example Metric: Number of conflicting requirements identified.
* Feasibility: Evaluates whether the requirements can be realistically implemented within the project constraints (time, budget, technology). Example Metric: Percentage of requirements deemed feasible after initial assessment.

2. Requirement Coverage Metrics

* Traceability: Measures the ability to trace each requirement through its lifecycle from origin to implementation. Example Metric: Percentage of requirements with full traceability (linked to business objectives, test cases, design elements).
* Coverage: Ensures all aspects of the project scope are covered by the requirements. Example Metric: Percentage of functional/non-functional requirements identified vs. total possible requirements.

3. Requirement Change Metrics

* Volatility: Tracks the frequency and impact of changes to the requirements during the project. Example Metric: Number of requirement changes per month.
* Stability: Assesses how stable the requirements are over time. Example Metric: Percentage of requirements that remain unchanged after initial approval.

4. Stakeholder Involvement Metrics

* Engagement: Measures the level of involvement and participation from stakeholders in the requirements gathering process. Example Metric: Number of stakeholder review meetings attended vs. total planned meetings.
* Satisfaction: Evaluates the satisfaction of stakeholders with the requirements gathering process and the final set of requirements. Example Metric: Stakeholder satisfaction score (collected via surveys).

5. Documentation Quality Metrics

* Accuracy: Ensures that the requirements document accurately reflects the stakeholders' needs and expectations. Example Metric: Number of errors or omissions found in requirements documents.
* Detail Level: Measures whether the requirements are detailed enough for the development team to start design and implementation. Example Metric: Percentage of requirements detailed to an acceptable level of granularity.

6. Time and Effort Metrics

* Time Spent: Tracks the amount of time spent on the requirements phase. Example Metric: Total hours spent on requirements elicitation, analysis, and documentation.
* Effort: Measures the effort required to gather, analyze, and document the requirements. Example Metric: Total person-days or person-hours dedicated to the requirements phase.

7. Risk Metrics

* Risk Identification: Evaluates the number of risks identified related to the requirements. Example Metric: Number of requirements-related risks identified.
* Risk Mitigation: Measures the effectiveness of risk mitigation strategies related to requirements. Example Metric: Percentage of identified risks that have mitigation plans.

8. Review and Approval Metrics

* Review Cycle Time: Tracks the time taken to review and approve the requirements. Example Metric: Average time taken for each review cycle.
* Approval Rate: Measures the rate at which requirements are approved by stakeholders. Example Metric: Percentage of requirements approved after first review.

9. Defect Metrics

* Defect Density: Measures the number of defects found in the requirements documents. Example Metric: Number of defects per requirement or per page of the requirements document.
* Defect Resolution Time: Tracks the time taken to resolve defects found in the requirements. Example Metric: Average time to resolve requirement defects.

**Skill sets required during requirements phase**

The requirements phase of a software development project involves gathering, analyzing, documenting, and validating the needs and expectations of stakeholders. This phase requires a diverse set of skills to ensure that requirements are accurately captured and understood. Here are the key skill sets needed during the requirements phase:

1. Communication Skills

* Active Listening: Ability to attentively listen to stakeholders, understand their needs, and ask clarifying questions.
* Verbal Communication: Clear and concise communication of ideas and questions during interviews, meetings, and workshops.
* Written Communication: Proficiency in writing clear, unambiguous, and detailed requirement documents, user stories, and use cases.

2. Interpersonal Skills

* Facilitation: Leading and managing requirement gathering sessions, workshops, and focus groups effectively.
* Negotiation: Balancing conflicting requirements and negotiating priorities among stakeholders to reach a consensus.
* Empathy: Understanding and addressing the concerns and perspectives of different stakeholders.

3. Analytical Skills

* Critical Thinking: Evaluating information and identifying the most relevant and important requirements.
* Problem-Solving: Identifying gaps, inconsistencies, and ambiguities in requirements and finding solutions to resolve them.
* Detail-Oriented: Ensuring all aspects of the requirements are thoroughly considered and documented.

4. Technical Skills

* Domain Knowledge: Understanding the specific industry or domain for which the software is being developed, including relevant standards and regulations.
* Technical Understanding: Basic understanding of the technical environment and constraints to assess feasibility and impact of requirements.
* Modeling Techniques: Using diagrams such as flowcharts, use case diagrams, entity-relationship diagrams (ERDs), and UML diagrams to visualize and analyze requirements.

5. Research Skills

* Data Gathering: Conducting interviews, surveys, observations, and document analysis to collect requirement information.
* Market Analysis: Understanding market trends, competitor products, and user expectations to inform requirements.

6. Documentation Skills

* Requirements Specification: Writing comprehensive requirement specifications that include functional, non-functional, and technical requirements.
* User Stories and Use Cases: Developing user stories and use cases to describe how users will interact with the system.

7. Validation and Verification Skills

* Requirements Validation: Ensuring that the documented requirements accurately reflect the stakeholders' needs.
* Quality Assurance: Conducting reviews and inspections to verify the completeness, clarity, and feasibility of requirements.

8. Project Management Skills

* Time Management: Managing time effectively to meet deadlines and deliverables during the requirements phase.
* Risk Management: Identifying and mitigating risks associated with requirement gathering and analysis.
* Prioritization: Prioritizing requirements based on business value, feasibility, and stakeholder input.

9. Tool Proficiency

* Requirements Management Tools: Using tools like Jira, Confluence, IBM Rational DOORS, or Microsoft Azure DevOps for tracking and managing requirements.
* Collaboration Tools: Proficiency with tools like Microsoft Teams, Slack, Zoom, or Google Workspace for communication and collaboration.
* Documentation Tools: Using word processors, spreadsheets, and diagramming tools such as Microsoft Word, Excel, Visio, Lucidchart, or similar.