

The background features abstract, overlapping green geometric shapes, primarily triangles and polygons, in various shades of green, creating a modern and dynamic look. The shapes are layered, with some appearing more prominent than others, and they extend from the edges towards the center of the slide.

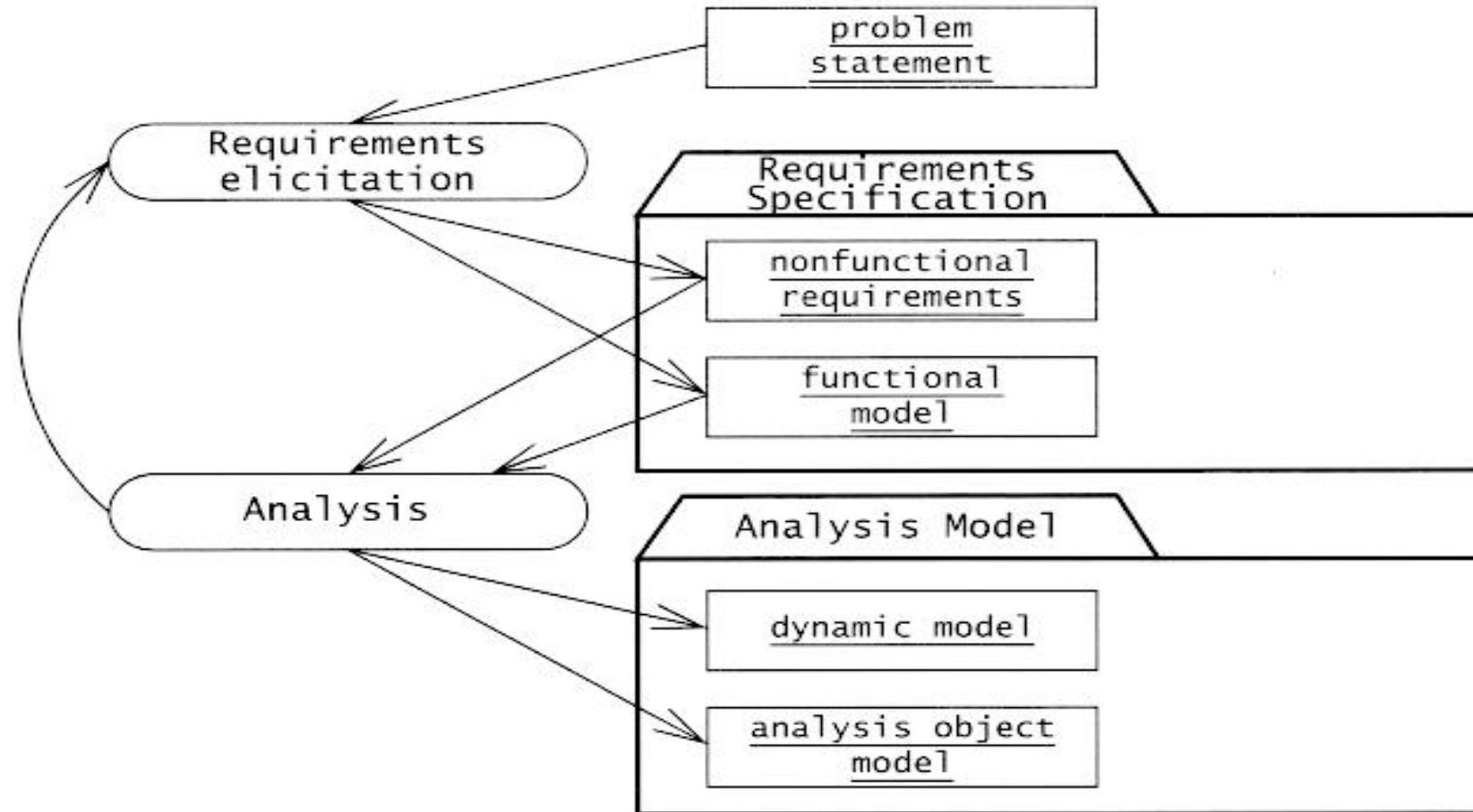
Unit 2

Requirement Elicitation

- ▶ Requirements elicitation focuses on describing the purpose of the system.
- ▶ The client, the developers, and the users identify a problem area and define a system that addresses the problem. Such a definition is called a requirements specification and serves as a contract between the client and the developers.
- ▶ The requirements specification is structured and formalized during analysis to produce an analysis model.
- ▶ Both requirements specification and analysis model represent the same information.
- ▶ They differ only in the language and notation they use; the requirements specification is written in natural language, whereas the analysis model is usually expressed in a formal or semiformal notation.
- ▶ The requirements specification supports the communication with the client and users.
- ▶ The analysis model supports the communication among developers.
- ▶ They are both models of the system in the sense that they attempt to represent accurately the external aspects of the system.
- ▶ Given that both models represent the same aspects of the system, requirements elicitation and analysis occur concurrently and iteratively.

- ▶ Requirements elicitation and analysis focus only on the user's view of the system.
- ▶ For example, the system functionality, the interaction between the user and the system, the errors that the system can detect and handle, and the environmental conditions in which the system functions are part of the requirements.
- ▶ The system structure, the implementation technology selected to build the system, the system design, the development methodology, and other aspects not directly visible to the user are not part of the requirements.

Products of requirements elicitation and analysis



Requirement Elicitation activities

► Requirements elicitation includes the following activities:

- Identifying actors
- Identifying scenarios.
- Identifying use cases.
- Refining use cases.
- Identifying relationships among use cases
- Identifying nonfunctional requirements.

Identifying actors

- ▶ During this activity, developers identify the different types of users the future system will support.
- ▶ Questions for identifying actors
 - ▶ Which user groups are supported by the system to perform their work?
 - ▶ Which user groups execute the system's main functions?
 - ▶ Which user groups perform secondary functions, such as maintenance and administration?
 - ▶ With what external hardware or software system will the system interact?

Identifying scenarios

- ▶ During this activity, developers observe users and develop a set of detailed scenarios for typical functionality provided by the future system.
- ▶ Scenarios are concrete examples of the future system in use.
- ▶ Developers use these scenarios to communicate with the user and deepen their understanding of the application domain

Types of scenarios

- ▶ As-is scenarios describe a current situation. During reengineering, for example, the current system is understood by observing users and describing their actions as scenarios. These scenarios can then be validated for correctness and accuracy with the users.
- ▶ Visionary scenarios describe a future system. Visionary scenarios are used both as a point in the modeling space by developers as they refine their ideas of the future system and as a communication medium to elicit requirements from users. Visionary scenarios can be viewed as an inexpensive prototype.
- ▶ Evaluation scenarios describe user tasks against which the system is to be evaluated. The collaborative development of evaluation scenarios by users and developers also improves the definition of the functionality tested by these scenarios.
- ▶ Training scenarios are tutorials used for introducing new users to the system. These are step-by-step instructions designed to hand-hold the user through common tasks.

Questions for identifying scenarios

- ▶ What are the tasks that the actor wants the system to perform?
- ▶ What information does the actor access? Who creates that data? Can it be modified or removed? By whom?
- ▶ Which external changes does the actor need to inform the system about? How often? When?
- ▶ Which events does the system need to inform the actor about? With what latency

Identifying use cases

- ▶ Once developers and users agree on a set of scenarios, developers derive from the scenarios a set of use cases that completely represent the future system.
- ▶ Whereas scenarios are concrete examples illustrating a single case, use cases are abstractions describing all possible cases.
- ▶ When describing use cases, developers determine the scope of the system.

Simple Use Case Writing Guide

- ▶ Use cases should be named with verb phrases
- ▶ Actors should be named with noun phrases
- ▶ The boundary of the system should be clear. Steps accomplished by the actor and steps accomplished by the system should be distinguished
- ▶ Use case steps in the flow of events should be phrased in the active voice. This makes it explicit who accomplished the step.
- ▶ The causal relationship between successive steps should be clear.
- ▶ A use case should describe a complete user transaction
- ▶ Exceptions should be described separately
- ▶ A use case should not describe the user interface of the system. This takes away the focus from the actual steps accomplished by the user and is better addressed with visual mock-ups
- ▶ A use case should not exceed two or three pages in length. Otherwise, use include and extend relationships to decompose it in smaller use cases,

Refining use cases

- ▶ During this activity, developers ensure that the requirements specification is complete by detailing each use case and describing the behavior of the system in the presence of errors and exceptional conditions

Identifying relationships among use cases

- ▶ During this activity, developers identify dependencies among use cases.
- ▶ They also consolidate the use case model by factoring out common functionality.
- ▶ This ensures that the requirements specification is consistent

Heuristics for extend and include relationships

- ▶ Use extend relationships for exceptional, optional, or seldom-occurring behavior. An example of seldom-occurring behavior is the breakdown of a resource .An example of optional behavior is the notification of nearby resources responding to an unrelated incident.
- ▶ Use include relationships for behavior that is shared across two or more use cases.

Identifying nonfunctional requirement

- ▶ During this activity, developers, users, and clients agree on aspects that are visible to the user, but not directly related to functionality.
- ▶ These include constraints on the performance of the system, its documentation, the resources it consumes, its security, and its quality

Two methods for eliciting information

- ▶ During requirements elicitation, developers access many different sources of information, including client-supplied documents about the application domain, manuals and technical documentation of legacy systems that the future system will replace, and most important, the users and clients themselves.
- ▶ Developers interact the most with users and clients during requirements elicitation.
- ▶ We focus on two methods for eliciting information, making decisions with users and clients, and managing dependencies among requirements and other artifacts:
 - ▶ Joint Application Design (JAD) focuses on building consensus among developers, users, and clients by jointly developing the requirements specification
 - ▶ Traceability focuses on recording, structuring, linking, grouping, and maintaining dependencies among requirements and between requirements and other work products

Types of Requirements

- ▶ Functional Requirements
- ▶ Non functional requirements

Functional Requirements

- ▶ Functional requirements describe the interactions between the system and its environment independent of its implementation.
- ▶ The environment includes the user and any other external system with which the system interacts

Nonfunctional Requirements

- ▶ Nonfunctional requirements describe aspects of the system that are not directly related to the functional behavior of the system.
- ▶ Nonfunctional requirements include a broad variety of requirements that apply to many different aspects of the system, from usability to performance.

Categories of nonfunctional requirements(as per The FURPS+ model)

- ▶ Usability
- ▶ Reliability
- ▶ Performance
- ▶ Supportability

Usability

- ▶ Usability is the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component.
- ▶ Usability requirements include, for example, conventions adopted by the user interface, the scope of online help, and the level of user documentation.
- ▶ Often, clients address usability issues by requiring the developer to follow user interface guidelines on color schemes, logos, and fonts

Reliability

- ▶ Reliability is the ability of a system or component to perform its required functions under stated conditions for a specified period of time.
- ▶ Reliability requirements include, for example, an acceptable mean time to failure and the ability to detect specified faults or to withstand specified security attacks.
- ▶ More recently, this category is often replaced by dependability, which is the property of a computer system such that reliance can justifiably be placed on the service it delivers.
- ▶ Dependability includes reliability, robustness (the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environment conditions), and safety (a measure of the absence of catastrophic consequences to the environment).

Performance

- ▶ Performance requirements are concerned with quantifiable attributes of the system, such as
 - ▶ response time (how quickly the system reacts to a user input)
 - ▶ throughput (how much work the system can accomplish within a specified amount of time)
 - ▶ availability (the degree to which a system or component is operational and accessible when required for use)
 - ▶ accuracy.

Supportability

- ▶ Supportability requirements are concerned with the ease of changes to the system after deployment
- ▶ For example,
 - ▶ adaptability (the ability to change the system to deal with additional application domain concepts)
 - ▶ maintainability (the ability to change the system to deal with new technology or to fix defects)
 - ▶ internationalization (the ability to change the system to deal with additional international conventions, such as languages, units, and number formats).

Additional categories of nonfunctional requirements

- ▶ Implementation requirements are constraints on the implementation of the system, including the use of specific tools, programming languages, or hardware platforms.
- ▶ Interface requirements are constraints imposed by external systems, including legacy systems and interchange formats.
- ▶ Operations requirements are constraints on the administration and management of the system in the operational setting.
- ▶ Packaging requirements are constraints on the actual delivery of the system (e.g., constraints on the installation media for setting up the software).
- ▶ Legal requirements are concerned with licensing, regulation, and certification issues.

How to identify non functional requirements?

| Category | Example questions |
|---|--|
| Usability | <ul style="list-style-type: none">• What is the level of expertise of the user?• What user interface standards are familiar to the user?• What documentation should be provided to the user? |
| Reliability <i>(including robustness, safety, and security)</i> | <ul style="list-style-type: none">• How reliable, available, and robust should the system be?• Is restarting the system acceptable in the event of a failure?• How much data can the system lose?• How should the system handle exceptions?• Are there safety requirements of the system?• Are there security requirements of the system? |
| Performance | <ul style="list-style-type: none">• How responsive should the system be?• Are any user tasks time critical?• How many concurrent users should it support?• How large is a typical data store for comparable systems?• What is the worst latency that is acceptable to users? |
| Supportability <i>(including maintainability and portability)</i> | <ul style="list-style-type: none">• What are the foreseen extensions to the system?• Who maintains the system?• Are there plans to port the system to different software or hardware environments? |

How to identify non functional requirements?

| | |
|-----------------------|---|
| Implementation | <ul style="list-style-type: none">• Are there constraints on the hardware platform?• Are constraints imposed by the maintenance team?• Are constraints imposed by the testing team? |
| Interface | <ul style="list-style-type: none">• Should the system interact with any existing systems?• How are data exported/imported into the system?• What standards in use by the client should be supported by the system? |
| Operation | <ul style="list-style-type: none">• Who manages the running system? |
| Packaging | <ul style="list-style-type: none">• Who installs the system?• How many installations are foreseen?• Are there time constraints on the installation? |
| Legal | <ul style="list-style-type: none">• How should the system be licensed?• Are any liability issues associated with system failures?• Are any royalties or licensing fees incurred by using specific algorithms or components? |

Greenfield Engineering, Reengineering, and Interface Engineering

- ▶ Requirements elicitation activities can be classified into three categories, depending on the source of the requirements
 - ▶ Greenfield Engineering
 - ▶ Reengineering
 - ▶ Interface Engineering

Greenfield Engineering

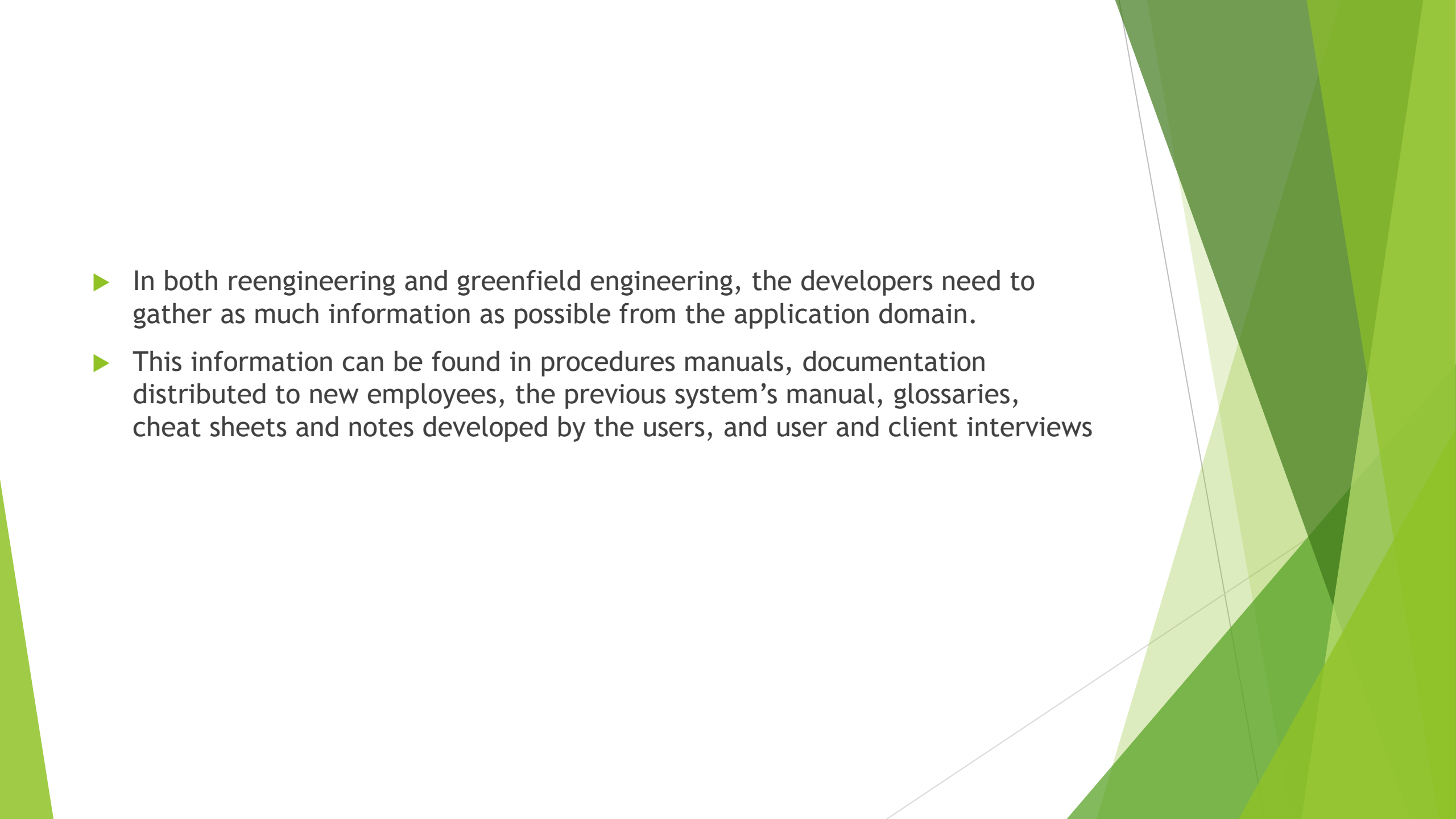
- ▶ In greenfield engineering, the development starts from scratch—no prior system exists—so the requirements are extracted from the users and the client.
- ▶ A greenfield engineering project is triggered by a user need or the creation of a new market.

Reengineering

- ▶ A reengineering project is the redesign and reimplementation of an existing system triggered by technology enablers or by business processes [Hammer & Champy, 1993].
- ▶ Sometimes, the functionality of the new system is extended, but the essential purpose of the system remains the same.
- ▶ The requirements of the new system are extracted from an existing system

Interface engineering

- ▶ An interface engineering project is the redesign of the user interface of an existing system.
- ▶ The legacy system is left untouched except for its interface, which is redesigned and reimplemented. T
- ▶ his type of project is a reengineering project in which the legacy system cannot be discarded without entailing high costs

- 
- The background of the slide features abstract, overlapping green geometric shapes, primarily triangles and polygons, in various shades of green, creating a modern and dynamic visual effect.
- ▶ In both reengineering and greenfield engineering, the developers need to gather as much information as possible from the application domain.
 - ▶ This information can be found in procedures manuals, documentation distributed to new employees, the previous system's manual, glossaries, cheat sheets and notes developed by the users, and user and client interviews

Managing Requirement Elicitation

- ▶ Negotiating Specifications with Clients: Joint Application Design
- ▶ Maintaining Traceability
- ▶ Documenting Requirements Elicitation

Negotiating Specifications with Clients: Joint Application Design

- ▶ Joint Application Design (JAD) is a requirements method developed at IBM at the end of the 1970s.
- ▶ Its effectiveness lies in that the requirements elicitation work is done in one single workshop session in which all stakeholders participate.
- ▶ Users, clients, developers, and a trained session leader sit together in one room to present their viewpoints, listen to other viewpoints, negotiate, and come to a mutually acceptable solution.
- ▶ The outcome of the workshop, the final JAD document, is a complete requirements specification document that includes definitions of data elements, work flows, and interface screens.
- ▶ Because the final document is jointly developed by the stakeholders the final JAD document represents an agreement among users, clients, and developers, and thus minimizes requirements changes later in the development process.


JAD is composed of five activities

- ▶ 1. Project definition: During this activity, the JAD facilitator interviews the project manager and the client to determine the objectives and the scope of the project. The findings from the interviews are collected in the Management Definition Guide.
- ▶ 2. Research: During this activity, the JAD facilitator interviews present and future users, gathers information about the application domain, and describes a first set of high-level use cases. The JAD facilitator also starts a list of problems to be addressed during the session. The results of this activity are a Session Agenda and a Preliminary Specification listing work flow and system information.
- ▶ 3. Preparation: During this activity, the JAD facilitator prepares the session. The JAD facilitator creates a Working Document,

- ▶ 4. Session: During this activity, the JAD facilitator guides the team in creating the requirements specification. A JAD session lasts for 3 to 5 days. The team defines and agrees on the scenarios, use cases, and user interface mock-ups. All decisions are documented by a scribe.
- ▶ 5. Final document: The JAD facilitator prepares the Final Document, revising the working document to include all decisions made during the session. The Final Document represents a complete specification of the system agreed on during the session. The Final Document is distributed to the session participants for review. The participants then attend a 1- to 2-hour meeting to discuss the reviews and finalize the document.

Maintaining Traceability

- ▶ Traceability is the ability to follow the life of a requirement.
- ▶ This includes tracing where the requirements came from (e.g., who originated it, which client need does it address) to which aspects of the system and the project it affects (e.g., which components realize the requirement, which test case checks its realization).
- ▶ Traceability enables developers to show that the system is complete, testers to show that the system complies with its requirements, designers to record the rationale behind the system, and maintainers to assess the impact of change

- 
- ▶ Traceability would enable us to answer the following questions:
 - ▶ Who originated the two-line display requirement?
 - ▶ Did any implicit constraints mandate this requirement?
 - ▶ Which components must be changed because of the additional button and display?
 - ▶ Which test cases must be changed?

Documenting Requirements Elicitation

- ▶ The results of the requirements elicitation and the analysis activities are documented in the Requirements Analysis Document (RAD).
- ▶ This document completely describes the system in terms of functional and nonfunctional requirements.
- ▶ The audience for the RAD includes the client, the users, the project management, the system analysts (i.e., the developers who participate in the requirements), and the system designers (i.e., the developers who participate in the system design).
- ▶ The first part of the document, including use cases and nonfunctional requirements, is written during requirements elicitation. The formalization of the specification in terms of object models is written during analysis.