ID 1.1.1.1 - Frequently encountered misconceptions

Meta data

	EN	DE
Status	DRAFT	
Title	Frequently encountered misconceptions on requirements engineering	Häufig anzutreffende Missverständnisse über Requirements Engineering
Tags		
Estimated Duration	5 min	
Level	☑ BEGINNER ☐ INTERMEDIATE ☐ EXPERT	
External ressources		
Dependencies		
Next Item	ID 1.1.1.2 - What is requirements engineering?	
Goals (Teaching perspective)	To clarify misconceptions and make a clear distinction between the myths and realities of requirements engineering in order to provide an interesting introduction into the following topics.	
Content description (Teaching perspective)	Explore common misunderstandings in the field of requirements engineering, such as	

ENGLISCH

[ENGLISCH] DESCRIPTION

In this learning nugget, we will explore common misconceptions about Requirements Engineering (RE). We will understand what RE is and how it's already being done in organisations, even if they're not aware of it. Additionally, we will expose common myths that can hinder effective project management and development.

🎯 [ENGLISCH] GOALS AND VALUE

In this learning nugget you will …

- Get an idea why requirements engineering is more than just collecting wishes before implementing solutions.
- Understand the dynamic nature of requirements and how to deal with change.
- Explore the need for interdisciplinary collaboration in RE processes.
- Recongise that good RE practices alone do not automatically lead to project success.

[ENGLISCH] CONTENT

> We don’t do requirements engineering

We often come across various statements such as: *"We don't do Requirements Engineering (RE), we do SCRUM"*. **RE, however, is generally seen as the process of identifying what the product should look like, what features it should have.** Even when organisations follow agile principles that do not explicitly prescribe what is perceived as RE activities, they still create, prioritise and track requirements as product backlog items or user stories. In the following paragraphs we will describe and refute other common misconceptions about RE.

Misconception 1: "RE is just writing down wants and needs".

RE is more than just listing wishes. It involves analysis, negotiation, and prioritisation of requirements to ensure that they are clear, measurable, and actionable (and testable!). A profound understanding of stakeholder needs and technical/business conditions is essential to this process. A clear set of requirements serves as a tool to create a common understanding between all stakeholders. This misconception will be addressed in more detail following unit.

Misconception 2: "Requirements and implementation are one and the same".

Contrary to popular belief, gathering requirements is not the same as implementing solutions. Requirements engineering involves eliciting, analyzing, documenting, and managing requirements, while implementation translates requirements into tangible solutions. Recognizing this difference is a critical determinant for success. Jumping directly to solutions without understanding the problem first - commonly referred to as solution orientation -- often results in a loss of time and effort, in the worst-case to a project failure. In more details this question would be addressed in the later in the learning path

Misconception 3: "A one-size-fits-all approach is sufficient for requirements engineering".

Every project is unique, with different stakeholders, objectives and even the chosen software process (e.g. agile) will affect the way we handle requirements. Fact is, that there is no one-size-fits-all approach to requirements engineering. Even though RE offers a wide spectrum of approaches, their customization to specific project needs is essential for effective requirements gathering and management. On this platform, we will therefore focus on the essence of RE - focussing on what matters most - and show how it can be used in different contexts.

Misconception 4: "Requirements are static and immutable".

It is important to recognise that requirements are dynamic and may change due to evolving stakeholder needs, changes in market conditions and advances in technology. It is important to adapt these requirements in the later stages of a project, either by reformulating, deleting or introducing new requirements. Adopting this approach is essential to maintaining project responsiveness and achieving success.

Misconception 5: "Requirements engineering is a one-time activity".

Every project begins with the need to properly understand the requirements. However, as requirements evolve over time, the process does not end once the requirements have been established. It is an iterative process that continues throughout the project lifecycle. As understanding deepens and stakeholder needs evolve, requirements may need to be refined, added or removed. This iterative approach promotes project flexibility.

Misconception 6: "Requirements engineering is the sole responsibility of business analysts".

While business analysts play a critical role, requirements engineering is a collaborative effort involving multiple stakeholders. Project managers, developers, testers and end users all bring unique perspectives that enrich the understanding and management of requirements. Consensus among all stakeholders should confirm the need to implement a particular requirement.

Misconception 7: “A perfect RE guarantees a successful project”.
Although an effective RE is critical to the success of the project, it does not guarantee it. The success of a project depends on a variety of factors, including project management, team dynamics, resource availability and implementation skills. RE is a critical component, but only one part of a comprehensive project management process.

💡 [ENGLISCH] KEY TAKEAWAY

- Every software project is subject to, and performs requirements engineering.

- Requirements Engineering goes beyond documentation; it involves analysis, negotiation and prioritisation of requirements.
- Requirements gathering and implementation are different processes, and RE should involve all stakeholders for effective project management.
- It's important to tailor the approach for Requirements Engineering to the unique needs of each project.
- While effective RE is critical, it does not guarantee project success; several other factors play an important role.

GERMAN

[GERMAN] DESCRIPTION

In diesem Lernabschnitt befassen wir uns mit gängigen Missverständnissen über Requirements Engineering (RE). Wir verstehen, was RE ist und wie es in Organisationen bereits praktiziert wird, auch wenn sie sich dessen nicht bewusst sind. Darüber hinaus entlarven wir gängige Mythen, die ein effektives Projektmanagement und eine effektive Entwicklung behindern können.

[GERMAN] GOALS AND VALUE

In diesem Lernmodul erfahren Sie ...

- Warum Requirements Engineering mehr ist als das Sammeln von Wünschen vor der Implementierung von Lösungen.
- Wie dynamisch Anforderungen sind und wie man mit Veränderungen umgeht.
- Dass interdisziplinäre Zusammenarbeit in RE-Prozessen notwendig ist.
- Dass gute RE-Praktiken allein nicht automatisch zum Projekterfolg führen.

[GERMAN] CONTENT

> Wir machen kein Requirements Engineering

Wir stoßen oft auf Aussagen wie: „Wir machen kein Requirements Engineering (RE), wir machen SCRUM". Aber **RE wird im Allgemeinen als der Prozess gesehen, in dem festgelegt wird, wie das Produkt aussehen soll und welche Eigenschaften es haben soll.** Selbst wenn Organisationen agilen Prinzipien folgen, die nicht ausdrücklich vorschreiben, was als RE-Aktivitäten angesehen wird, erstellen, priorisieren und verfolgen sie Anforderungen als Product Backlog Items oder User Stories. In den folgenden Abschnitten werden wir weitere gängige Missverständnisse über RE beschreiben und widerlegen.

Irrtum 1: „RE ist nur das Aufschreiben von Wünschen und Bedürfnissen".

RE ist mehr als nur die Auflistung von Wünschen. Es umfasst Analyse, Verhandlung und Priorisierung, um sicherzustellen, dass die Anforderungen klar, messbar und umsetzbar (und testbar!) sind. Ein tiefes Verständnis der Bedürfnisse der Interessengruppen und der technischen/wirtschaftlichen Bedingungen ist für diesen Prozess unerlässlich. Ein klarer Anforderungskatalog dient als Instrument zur Schaffung eines gemeinsamen Verständnisses zwischen allen Beteiligten. Auf dieses Missverständnis wird in der folgenden Einheit näher eingegangen.

Irrtum 2: „Anforderungen und Umsetzung sind ein und dasselbe".

Entgegen der landläufigen Meinung ist das Sammeln von Anforderungen nicht dasselbe wie die Implementierung von Lösungen. Beim Requirements Engineering geht es um das Erfassen, Analysieren, Dokumentieren und Verwalten von Anforderungen, während bei der Implementierung die Anforderungen in konkrete Lösungen umgesetzt werden. Das Erkennen dieses Unterschieds ist entscheidend für ein erfolgreiches Projektmanagement. Sich direkt auf die Lösungen zu stürzen, ohne das Problem zuerst zu verstehen, führt oft zu einem Verlust an Zeit und Aufwand, im schlimmsten Fall zu einem Scheitern des Projekts. Auf diese Frage wird im weiteren Verlauf des Lernpfads näher eingegangen.

Irrglaube 3: „Ein einheitlicher Ansatz ist für das Requirements Engineering ausreichend".

Jedes Projekt ist einzigartig, mit unterschiedlichen Interessengruppen und Zielen. Auch der gewählte Softwareprozess (z.B. agil) beeinflusst die Art und Weise, wie wir mit Anforderungen umgehen. Tatsache ist, dass es keine Einheitslösung für das Requirements

Engineering gibt. Auch wenn RE ein breites Spektrum an Ansätzen bietet, ist deren Anpassung an die spezifischen Projektbedürfnisse für eine effektive Anforderungserhebung und -verwaltung unerlässlich. Auf dieser Plattform werden wir uns daher auf das Wesentliche des RE konzentrieren und zeigen, wie es in unterschiedlichen Kontexten eingesetzt werden kann.

Irrglaube 4: „Anforderungen sind statisch und unveränderlich“.

Es ist wichtig zu erkennen, dass die Anforderungen dynamisch sind und sich aufgrund der sich entwickelnden Bedürfnisse der Beteiligten, der veränderten Marktbedingungen und des technischen Fortschritts ändern können. Diese Anforderungen müssen in den späteren Phasen eines Projekts angepasst werden, entweder durch Neuformulierung, Streichung oder Einführung neuer Anforderungen. Diesen Ansatz anzuwenden ist eine wesentliche Voraussetzung, um die Reaktionsfähigkeit und den Erfolg eines Projekts aufrechtzuerhalten.

Irrglaube 5: „Requirements Engineering ist eine einmalige Aktivität“.

Jedes Projekt beginnt mit der Notwendigkeit, die Anforderungen richtig zu verstehen. Da sich die Anforderungen jedoch im Laufe der Zeit weiterentwickeln, endet der Prozess nicht, sobald die Anforderungen festgelegt sind. Es handelt sich um einen iterativen Prozess, der während des gesamten Projektlebenszyklus fortgesetzt wird. Wenn sich das Verständnis vertieft und die Bedürfnisse der Stakeholder weiterentwickelt haben, müssen die Anforderungen möglicherweise verfeinert, hinzugefügt oder entfernt werden. Dieser iterative Ansatz fördert die Flexibilität des Projekts.

Irrtum 6: „Die Anforderungserstellung ist allein Sache der Business-Analysten“.

Während Business-Analysten eine entscheidende Rolle spielen, ist die Anforderungserstellung ein gemeinschaftliches Unterfangen, an dem mehrere Interessengruppen beteiligt sind. Projektmanager, Entwickler, Tester und Endbenutzer bringen alle einzigartige Perspektiven ein, die das Verständnis und die Verwaltung der Anforderungen bereichern. Der Konsens aller Beteiligten sollte die Notwendigkeit der Umsetzung einer bestimmten Anforderung bestätigen.

Irrglaube 7: „Ein perfektes RE garantiert ein erfolgreiches Projekt“.

Ein wirksames RE ist zwar entscheidend für den Projekterfolg, garantiert ihn aber nicht. Der Erfolg eines Projekts hängt von einer Vielzahl von Faktoren ab, darunter Projektmanagement, Teamdynamik, Verfügbarkeit von Ressourcen und Umsetzungsfähigkeiten. RE ist eine wichtige Komponente, aber nur ein Teil eines umfassenden Projektmanagementprozesses.

💡 [GERMAN] KEY TAKEAWAY

- Jedes Softwareprojekt unterliegt einem Requirements Engineering und führt dieses durch.
- Requirements Engineering geht über die Dokumentation hinaus; es umfasst die Analyse, Verhandlung und Priorisierung von Anforderungen.
- Anforderungserfassung und Implementierung sind unterschiedliche Prozesse. RE sollte alle Beteiligten für ein effektives Projektmanagement einbeziehen.
- Es ist wichtig, den Ansatz für die Anforderungsanalyse auf die besonderen Bedürfnisse jedes Projekts abzustimmen.
- Ein effektives RE ist zwar entscheidend, aber keine Garantie für den Projekterfolg; mehrere andere Faktoren spielen eine wichtige Rolle.

ID 1.1.1.2 - What is requirements engineering?

Meta data

	EN	DE
Status	DRAFT	
Title	What is Requirements Engineering?	Was ist Requirements Engineering überhaupt?
Tags		
Estimated Duration	5 min	
Level	☑ BEGINNER ☐ INTERMEDIATE ☐ EXPERT	
External ressources	Research describing the importance of RE	
Dependencies	ID 1.1.1.1 - Frequently encountered misconceptions	
Next Item	ID 1.1.1.3 - What is a requirement?	
Goals (Teaching perspective)	To provide an understanding of the role and significance of requirements engineering in software development	
Content description (Teaching perspective)	Introduction to the discipline of requirements engineering, focusing on its importance, impact and outcomes for projects	

ENGLISCH

[ENGLISCH] DESCRIPTION

In this Learning Nugget we will explore Requirements Engineering (RE), an essential component of successful software development. We will look at its formal definition, key activities and the benefits it provides for software projects.

[ENGLISCH] GOALS AND VALUE

In this learning nugget you will …

- Understand the definition of Requirements Engineering.
- Identify the key activities involved in the RE process.
- Evaluate the impact of RE on the project success.

[ENGLISCH] CONTENT

Introduction to Requirements Engineering

Requirements Engineering (RE) is a fundamental process in software development that focuses on eliciting, analysing, documenting and validating the needs and expectations of stakeholders. It is the foundation upon which successful software projects are built.

Formal definition and key activities

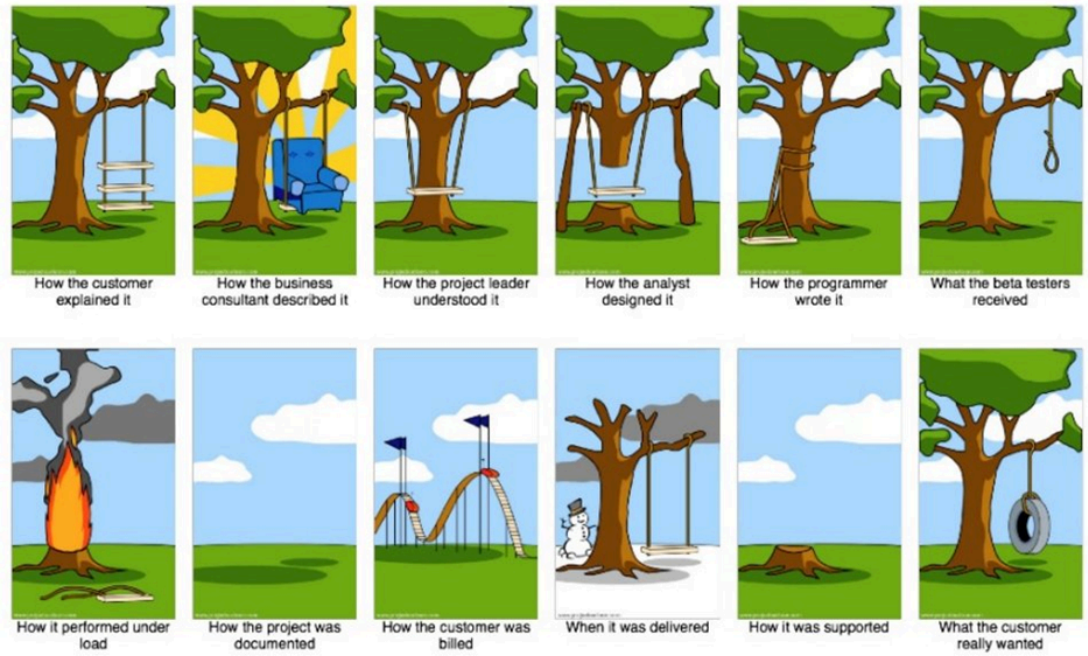
> **Requirements Engineering (RE)** is the systematic, iterative, and disciplined approach to develop explicit requirements and system specification that all stakeholders agree upon.

Elementary tasks include, but are not limited to:

- Requirements elicitation – identifying all relevant stakeholders and their requirements.
- Requirements analysis – understanding requirements and achieving consensus among stakeholders.
- Requirements specification – structuring, modelling, documenting requirements.
- Requirements validation and verification – ensuring validity and quality of created requirements.

Impact on Project Success

Effective Requirements Engineering is critical to successful software development. Research shows that 33% of errors in the software development lifecycle are rooted in insufficient Requirements Engineering, and 36% of these errors lead to project failure. Requirements Engineering is therefore both inherently difficult and a critical determinent for success. Common reasons for project failure include nowaways incomplete, hidden and unclear requirements, and communication problems with customers. A lack of common understanding between stakeholders is a major challenge, as illustrated in the following picture.



What are the benefits of applying Requirements Engineering?

- **Alignment with stakeholder needs:** Requirements engineering facilitates clear communication between stakeholders, ensuring that the final product meets their expectations and increasing user satisfaction.
- **Improved communication and collaboration:** Requirements engineering fosters collaboration across cross-functional teams, aligning stakeholders and promoting a common project vision.
- **Reduced rework and costs:** Well-defined requirements reduce misunderstandings and costly rework, saving time and resources in later stages of development.
- **Improved quality and reliability:** Clear requirements act as a blueprint, leading to more robust, reliable and maintainable software systems, improving overall quality.

- **Risk Management:** Early identification and management of requirements-related risks minimises disruptions and delays, promoting smoother project execution.
- **Traceability and change management:** Clear requirements enable systematic change management and maintain the integrity of the software system as requirements evolve.
- **Regulatory and standards compliance:** Requirements engineering ensures compliance with regulatory requirements and industry standards, mitigating the risk of non-compliance during development.
- **Accurate project planning:** Knowing the key problems in advance is a foundation for an effective and efficient development and delivering products that the stakeholders really need

[ENGLISCH] KEY TAKEAWAY

- Requirements Engineering is a systematic approach to developing clear and comprehensive requirements for software development projects.
- RE helps to create a common understanding among involved stakeholders
- Key activities in RE include elicitation, analysis, specification and validation/verification.
- Clear requirements mitigates risk, facilitate communication, traceability and alignment with stakeholder needs.
- RE affects system functionality, quality, cost, benefits and complexity, and has a significant impact on project success.

GERMAN

[GERMAN] DESCRIPTION

In diesem Lernabschnitt befassen wir uns mit dem Requirements Engineering (RE), einer wesentlichen Komponente der erfolgreichen Softwareentwicklung. Wir betrachten seine formale Definition, die wichtigsten Aktivitäten und die Vorteile, die es für Softwareprojekte bietet.

[GERMAN] GOALS AND VALUE

In diesem Lernmodul erfahren Sie ...

- Wie Requirements Engineering definiert wird.
- Was die wichtigsten Aktivitäten im RE-Prozess sind.
- Wie RE den Projekterfolg beeinflusst.

[GERMAN] CONTENT

Einführung in das Requirements Engineering

Requirements Engineering (RE) ist ein grundlegender Prozess in der Softwareentwicklung, der sich auf das Erheben, Analysieren, Dokumentieren und Validieren der Bedürfnisse und Erwartungen der Beteiligten konzentriert. Es ist die Grundlage, auf der erfolgreiche Softwareprojekte aufgebaut werden.

Formale Definition und Schlüsselaktivitäten

> **Requirements Engineering (RE)** ist der systematische, iterative und disziplinierte Ansatz zur Entwicklung einer expliziten Anforderungs- und Systemspezifikation, der alle Beteiligten zustimmen.

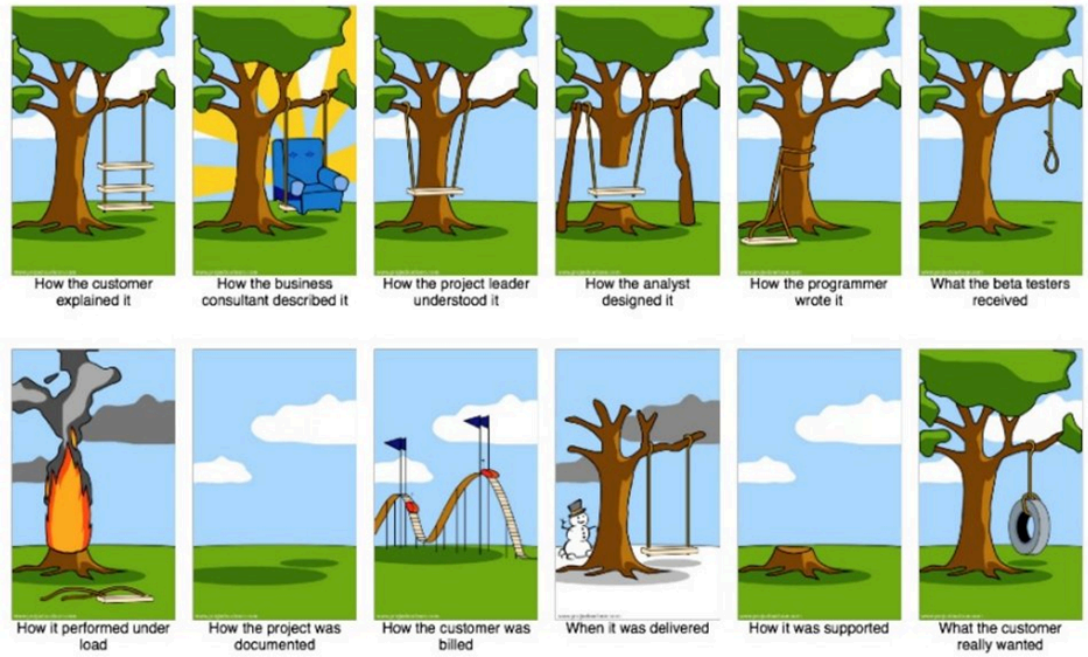
Zu den grundlegenden Aufgaben gehören unter anderem:

- Anforderungserhebung - Identifizierung aller relevanten Interessengruppen und ihrer Anforderungen.
- Anforderungsanalyse - Verstehen der Anforderungen und Erreichen eines Konsenses zwischen den Stakelholdern.
- Anforderungsspezifikation - Strukturierung, Modellierung und Dokumentation von Anforderungen.

- Validierung und Verifizierung von Anforderungen - Sicherstellung der Gültigkeit und Qualität der erstellten Anforderungen.

Auswirkungen auf den Projekterfolg

Effektives Requirements Engineering ist entscheidend für eine erfolgreiche Softwareentwicklung. Untersuchungen zeigen, dass 33 % der Fehler in dieser Phase auftreten, und 36 % dieser Fehler führen zum Scheitern des Projekts. Häufige Gründe für das Scheitern sind unvollständige, versteckte und unklare Anforderungen sowie Kommunikationsprobleme mit den Kunden. Ein mangelndes gemeinsames Verständnis zwischen den Beteiligten ist eine große Herausforderung, wie die folgende Abbildung zeigt.



Was sind die Vorteile, Requirements Engineering anzuwenden?

- **Anpassung an die Bedürfnisse der Beteiligten:** Das Requirements Engineering erleichtert die klare Kommunikation zwischen den Stakeholdern und stellt sicher, dass das Endprodukt ihren Erwartungen entspricht und die Zufriedenheit der Nutzer erhöht.
- **Verbesserte Kommunikation und Zusammenarbeit:** Requirements Engineering fördert die Zusammenarbeit zwischen funktionsübergreifenden Teams, bringt die beteiligten Interessensgruppen zusammen und fördert eine gemeinsame Projektvision.
- **Geringere Nacharbeit und Kosten:** Gut definierte Anforderungen verringern Missverständnisse und kostspielige Nacharbeiten und sparen so Zeit und Ressourcen in späteren Entwicklungsphasen.
- **Verbesserte Qualität und Zuverlässigkeit:** Klare Anforderungen wirken wie eine Blaupause, die zu robusteren, zuverlässigeren und wartungsfähigeren Softwaresystemen führt und die Qualität insgesamt verbessert.
- **Risikomanagement:** Durch die frühzeitige Identifizierung und das Management anforderungsbezogener Risiken werden Störungen und Verzögerungen minimiert, was einen reibungsloseren Projektablauf fördert.
- **Nachvollziehbarkeit und Änderungsmanagement:** Klare Anforderungen ermöglichen ein systematisches Änderungsmanagement und erhalten die Integrität des Softwaresystems, wenn sich die Anforderungen weiterentwickeln.
- **Einhaltung von Vorschriften und Standards:** Das Requirements Engineering gewährleistet die Einhaltung gesetzlicher Vorschriften und Industriestandards und mindert das Risiko einer Nichteinhaltung während der Entwicklung.
- **Genaue Projektplanung:** Die Kenntnis der wichtigsten Probleme im Voraus ist die Grundlage für eine effektive und effiziente Entwicklung und die Bereitstellung von Produkten, die die Beteiligten wirklich benötigen.

💡 [GERMAN] KEY TAKEAWAY

- Requirements Engineering ist ein systematischer Ansatz zur Entwicklung klarer und umfassender Anforderungen für Softwareentwicklungsprojekte.
- RE hilft dabei, ein gemeinsames Verständnis zwischen den beteiligten Interessengruppen zu schaffen.
- Zu den wichtigsten Aktivitäten im RE gehören die Erhebung, Analyse, Spezifikation und Validierung/Verifizierung.
- Klare Anforderungen mindern das Risiko, erleichtern die Kommunikation, die Rückverfolgbarkeit und die Anpassung an die Bedürfnisse der Stakeholder.
- RE wirkt sich auf die Systemfunktionalität, die Qualität, die Kosten, den Nutzen und die Komplexität aus und hat einen erheblichen Einfluss auf den Projekterfolg.

ID 1.1.1.3 - What is a requirement?

Meta data

	EN	DE
Status	DRAFT	
Title	What is a requirement?	Was ist eine Anforderung?
Tags	• Requirement definition	• Anforderungsdefinition
Estimated Duration	5 min	
Level	☑ BEGINNER ☐ INTERMEDIATE ☐ EXPERT	
External ressources	• List of Part of Speech and related Components	
Dependencies		
Next Item	ID 1.1.1.4 - Problem space vs solution space	
Goals (Teaching perspective)	To ensure learners can identify requirements and understand their impact on projects.	
Content description (Teaching perspective)	Define what constitutes a requirement within the context of system and software engineering, including its attributes and how it guides the development process	

ENGLISCH

[ENGLISCH] DESCRIPTION

In this learning nugget we will take a deep dive into the world of requirements in software engineering. By understanding what requirements are, and how they shape the development process, learners will gain practical insights into how to effectively identify, manage, and leverage requirements for successful project outcomes.

🎯 [ENGLISCH] GOALS AND VALUE

In this Learning Nugget you will …

- Understand what a requirement is.
- Get an initial understanding of how to derive requirements from the system description.
- See how requirements affects software development.

[ENGLISCH] CONTENT

What is a requirement?

In systems and software engineering, requirements play a crucial role in project development, serving as a fundamental aspect that guides the design, implementation and validation processes. The term “Requirement“ is often used to describe different aspects of a project's needs or specifications, such as

> 1. **Need or constraint:** A requirement can be a stakeholder need or an imposed constraint.
> 2. **Capability or property:** It defines what a system should be able to do or what properties it should have.

Examples:

1. **Constraint**: The system shall comply with the Payment Card Industry Data Security Standard (PCI DSS) for handling payment information.
2. **Property**: The system shall encrypt all sensitive user data (such as credit card information) to ensure PCI DSS compliance.

The role of requirements

Requirements play various roles at different stages of software development. As requirements evolve and become clearer, they take on new functions.

- Requirement as a **basic idea**:
  When a requirement is first formulated, it helps structure the design and behaviour of a software system and improves communication. It acts as a basic blueprint, outlining the essential functionality and features needed for the project to succeed.
- Requirements as a **highl-level need**:
  Requirements also represent a high-level (business) need. They help to establish a clear understanding of the project objectives and align development efforts with overarching goals.
- Requirement as a **documented need, function or constraint**:
  In later stages of requirements development (analysis and refinement), requirements serve as the cornerstone for all project activities and decisions. They set standards for implementation and help with quality assurance.

Derive Requirements from the System Description

At the beginning of product development, stakeholders often have an abstract idea of the system/service. Let us explore together how we get from an abstract idea to concrete requirements using the example of an online shopping platform.

For that purpose, we apply Abbotts textual analysis technique and analyze the system description for nouns, verbs and adjectives (see here for a full list of part of speech and their meaning). Nouns describe objects relevant for our online shapping platform. Verbs describe interactions, we have to support with those objects, and adjectives hint towards how such interactions have to happen. Especially verbs and adjectives are of interest: the first provide valuable indicators for indentifying functional requirements and the latter for identifying non-functional requirements. In the following, we exemplarly highlighted a couple of relevant terms and derived requirements from them.

System Description:

The **Online Shopping Platform** is a web-based application designed to facilitate users in purchasing products conveniently over the internet. The platform offers a wide range of products across different categories such as electronics, clothing, home appliances, and more. Users can **browse** through the catalog, **search** for specific items, and **add** them to their shopping cart. Upon **checkout**, users can **select** their preferred **payment method**, including credit/debit card, PayPal, or other online payment options. The system ensures **secure transactions** by encrypting sensitive information and **providing** a confirmation email for each purchase. Additionally, users have the option to **track** their **orders** and view their **purchase history**. The platform also **features** a user-friendly interface with intuitive **navigation** and **responsive design** for optimal viewing on various devices, including desktops, laptops, tablets, and smartphones.

Requirements:

- **Browse** functionality for users to navigate through product categories.

- **Search** feature enabling users to find specific items efficiently.
- **Add** to cart functionality to allow users to gather items for purchase.
- **Checkout** process for users to complete transactions.
- Support for multiple **payment methods** such as credit/debit cards, PayPal, etc.
- **Secure transactions** through encryption and confirmation emails.
- **Order tracking** feature for users to monitor the status of their purchases.
- **Purchase history** functionality for users to review past orders.
- User-friendly **interface** with intuitive navigation.
- **Responsive design** for optimal viewing across devices.

[ENGLISCH] KEY TAKEAWAY

- Requirements in systems and software engineering cover stakeholder needs, constraints, capabilities and characteristics.
- Throughout the software development lifecycle, requirements act as blueprints, representing business needs and guiding project activities.
- Derived from system descriptions, requirements move from abstract ideas to concrete specifications.
- Clear and detailed requirements are essential for effective project development, ensuring stakeholder alignment and guiding implementation and quality assurance efforts.

GERMAN

[GERMAN] DESCRIPTION

In diesem Lernabschnitt tauchen wir tief in die Welt der Anforderungen in der Softwareentwicklung ein. Durch das Verständnis, was Anforderungen sind und wie sie den Entwicklungsprozess prägen, erhalten die Lernenden praktische Einblicke in die effektive Ermittlung, Verwaltung und Nutzung von Anforderungen für erfolgreiche Projektergebnisse.

[GERMAN] GOALS AND VALUE

In diesem Lernmodul erfahren Sie ...

- Was eine Anforderung ist.
- Wie man Anforderungen aus der Systembeschreibung ableitet.
- Wie Anforderungen die Softwareentwicklung beeinflussen.

[GERMAN] CONTENT

Was ist eine Anforderung?

Im System- und Software-Engineering spielen die Anforderungen eine entscheidende Rolle bei der Projektentwicklung und dienen als grundlegender Aspekt, der den Entwurfs-, Implementierungs- und Validierungsprozess leitet. Der Begriff „Anforderung“ wird häufig verwendet, um verschiedene Aspekte der Bedarfe oder Spezifikationen eines Projekts zu beschreiben, z. B.

> 1. **Bedarf oder Einschränkung:** Eine Anforderung kann ein Bedürfnis der Interessengruppen oder eine auferlegte Einschränkung sein.
> 2. **Fähigkeit oder Eigenschaft:** Sie definiert, was ein System können sollte oder welche Eigenschaften es haben sollte.

Beispiele:

1. **Einschränkung:** Das System muss den Payment Card Industry Data Security Standard (PCI DSS) für den Umgang mit Zahlungsinformationen einhalten.

2. **Eigenschaft:** Das System verschlüsselt alle sensiblen Benutzerdaten (wie z. B. Kreditkarteninformationen), um die Einhaltung des PCI DSS zu gewährleisten.

Die Rolle der Anforderungen

Anforderungen spielen in den verschiedenen Phasen der Softwareentwicklung unterschiedliche Rollen. Wenn sich die Anforderungen weiterentwickeln und klarer werden, nehmen sie neue Funktionen an.

- Anforderungen als **Grundidee**: Wenn eine Anforderung zum ersten Mal formuliert wird, hilft sie, das Design und Verhalten eines Softwaresystems zu strukturieren und die Kommunikation zu verbessern. Sie fungieren als grundlegende Blaupause, die die wesentlichen Funktionen und Merkmale umreißt, die für den Erfolg des Projekts erforderlich sind.
- Anforderungen als **Geschäftsbedürfnis**: Anforderungen stellen auch ein Geschäftsbedürfnis dar. Sie helfen dabei, ein klares Verständnis der Projektziele zu schaffen und die Entwicklungsbemühungen auf die übergeordneten Unternehmensziele auszurichten.
- Anforderungen als **dokumentierter Bedarf, Funktion oder Einschränkung**: In späteren Phasen der Anforderungsentwicklung dienen die Anforderungen als Grundstein für alle Projektaktivitäten und -entscheidungen. Sie setzen Standards für die Implementierung und helfen bei der Qualitätssicherung.

Ableitung der Anforderungen aus der Systembeschreibung

Zu Beginn der Produktentwicklung haben die Beteiligten oft eine abstrakte Vorstellung von dem System/der Dienstleistung. Lassen Sie uns gemeinsam am Beispiel einer Online-Einkaufsplattform untersuchen, wie wir von einer abstrakten Idee zu konkreten Anforderungen gelangen.
Hierfür wenden wir Abbotts Textanalysetechnik an und analysieren die Systembeschreibung auf Nomen, Verben und Adjektive (siehe hier für eine komplete Liste von Termen und deren Bedeutung). Nomen beschreiben relevante Objekte für unsere Online-Einkaufsplattform, Verben beschreiben Interaktionen mit diesen Objekten, die es zu unterstützen gilt, und Adjektive geben Hinweise darauf wie diese Interaktionen zu erfolgen haben. Im Folgenden haben wir exemplarisch ein paar der relevanten Terme hervorgehoben und Anforderungen daraus erstellt.

Systembeschreibung:

Die **Online-Einkaufsplattform** ist eine webbasierte Anwendung, die den Benutzern den bequemen Einkauf von Produkten über das Internet erleichtern soll. Die Plattform bietet eine breite Palette von Produkten in verschiedenen Kategorien wie Elektronik, Kleidung, Haushaltsgeräte und mehr. Die Nutzer können den Katalog **durchstöbern**, nach bestimmten Artikeln **suchen** und diese in den Warenkorb **legen**. An der Kasse können die Nutzer ihre **bevorzugte** Zahlungsmethode **auswählen**, darunter Kredit-/Debitkarte, PayPal oder andere Online-Zahlungsoptionen. Das System gewährleistet **sichere Transaktionen**, indem es sensible Daten verschlüsselt und für jeden Kauf eine Bestätigungs-E-Mail **versendet**. Darüber hinaus haben die Nutzer die Möglichkeit, ihre **Bestellungen** zu **verfolgen** und ihre **Kaufhistorie** einzusehen. Die Plattform **verfügt** außerdem über eine benutzerfreundliche Oberfläche mit intuitiver **Navigation** und **responsivem Design** für eine optimale Anzeige auf verschiedenen Geräten, einschließlich Desktops, Laptops, Tablets und Smartphones.

Anforderungen:

- **Blättern**-Funktionalität für Benutzer, um durch Produktkategorien zu navigieren.
- **Suchfunktion**, die es den Nutzern ermöglicht, bestimmte Artikel effizient zu finden.
- Die **Funktion** „In den Warenkorb“ ermöglicht es den Nutzern, Artikel zum Kauf zusammenzustellen.
- **Checkout**-Prozess für Benutzer, um Transaktionen abzuschließen.
- **Unterstützung** für mehrere Zahlungsarten wie Kredit-/Debitkarten, PayPal, usw.
- **Sichere Transaktionen** durch Verschlüsselung und Bestätigungs-E-Mails.
- **Auftragsverfolgungsfunktion** für Benutzer zur Überwachung des Status ihrer Einkäufe.
- **Kaufverlaufsfunktion** für Benutzer, um frühere Bestellungen zu überprüfen.
- Benutzerfreundliche **Oberfläche** mit intuitiver Navigation.

- **Responsive Design** für optimale Anzeige auf allen Geräten.

💡 [GERMAN] KEY TAKEAWAY

- Anforderungen in der System- und Softwareentwicklung umfassen die Bedürfnisse, Einschränkungen, Fähigkeiten und Eigenschaften der Beteiligten.
- Während des gesamten Lebenszyklus der Softwareentwicklung fungieren Anforderungen als Blaupausen, die Geschäftsziele darstellen und die Projektaktivitäten leiten.
- Abgeleitet von Systembeschreibungen gehen Anforderungen von abstrakten Ideen zu konkreten Spezifikationen über.
- Klare und detaillierte Anforderungen sind für eine effektive Projektentwicklung unerlässlich, da sie die Abstimmung mit den Interessensgruppen gewährleisten und die Implementierung und Qualitätssicherung steuern.

ID 1.1.1.4 - Problem space vs solution space

Meta data

	EN	DE
Status	DRAFT	
Title	Problem space vs solution space	Problemdomäne vs Lösungsdomäne
Tags	• problem space • solution space	• problemraum • lösungsraum
Estimated Duration	5 min	
Level	☑ BEGINNER ☐ INTERMEDIATE ☐ EXPERT	
External ressources		
Dependencies		
Next Item	ID 1.1.1.5 - Types of Requirements	
Goals (Teaching perspective)	To teach the importance of clearly defining the problem before jumping to solutions.	
Content description (Teaching perspective)	Differentiation between problem space, which focuses on identifying and understanding the problem, and solution space, which is about formulating solutions	

ENGLISCH

[ENGLISCH] DESCRIPTION

In this Learning Nugget we will discuss the distinction between the problem space and the solution space in project development. We will emphasise why it is important to define a problem accurately before considering ways to solve it.

[ENGLISCH] GOALS AND VALUE

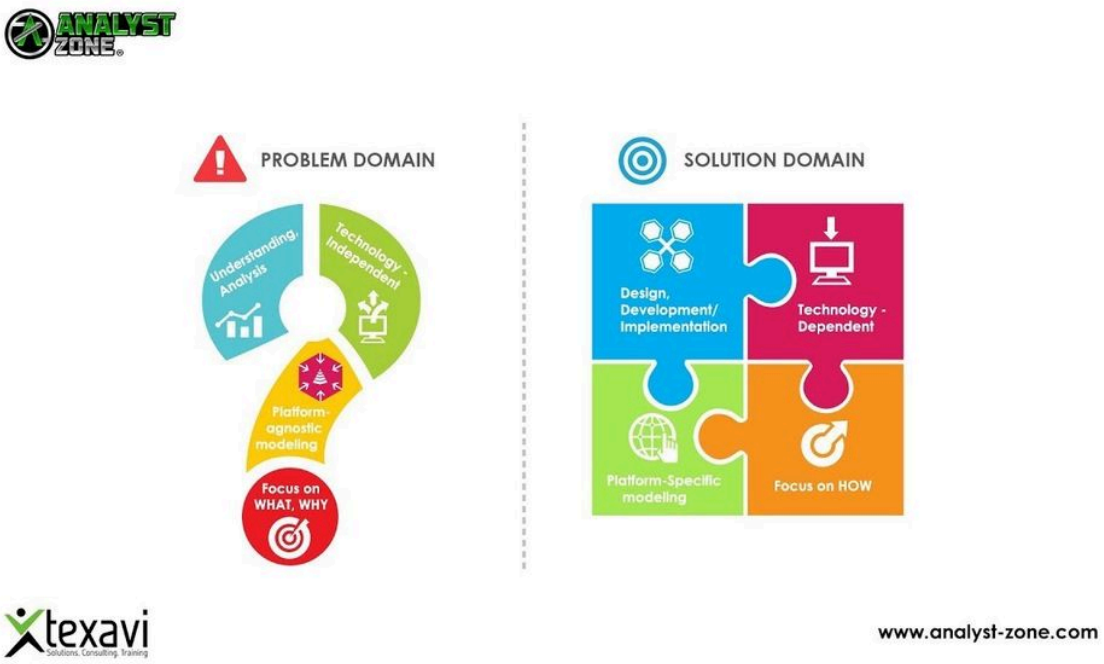
In this learning nugget you will …

- Understand the distinction between problem space and solution space in project development.
- Recognise common confusion that occurs when stakeholders focus prematurely on implementation details.
- Gain strategies for successfully navigating the process.

[ENGLISCH] CONTENT

It is important to separate the questions of WHAT problem to solve and HOW to solve it, as this reduces confusion, provides greater clarity and opens up more options. The WHAT (to do) is what we often refer to as the problem space and the HOW (to do it) is what we often refer to as the solution space. The first is in scope of requirements while the second is in scope of subsequent activities. We will now look at these concepts in more detail.

Understand the problem



Before jumping to solutions, it's important to have a clear understanding of the problem. Understanding the problem first ensures that efforts are focused in the right direction and avoids confusion and inefficiency. It also gives a certain and important degree of freedom for solution providers to develop products and services without unnecessary restrictions.

Requirements: What is the problem to be solved?

Requirements outline what a product or service must do, or the constraints within which it must operate. We define the system's needs and goals by understanding user needs, business objectives or market demands. We identify the concise description of the intended functions and behaviour of the system. Here, we take an external view on the system as a whole with no care for how it is internally realised. That is, at this point, we do not need to think about the specific technologies, platforms and other details of the implementation. We don't need to commit to anything specific. A concrete example is to analyze the user requirements without thinking of internals of a system respectively treating the system as a blackbox to solve the requirements.

Example

A new requirement for an online shopping platform might be that users must log in to the system in order to place an order. This login process should be quick, easy and secure.

Crafting solutions

Once the problem is well understood, it's time to think about how to solve it.

Solutions: How to achieve it

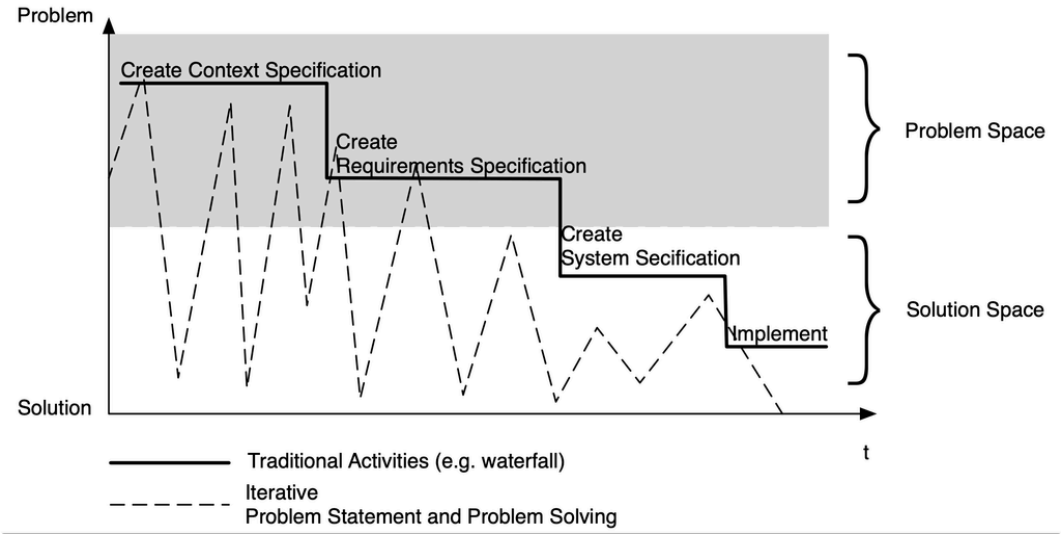
Solutions address the "how" by detailing the implementation specifics needed to meet the requirements. Thinking about solutions involves considering different methods, designs, architectures and implementation specifics. Here we think abou the internal realisation of how the system should operate to satisfy the requirements.

Example

A solution to the above requirement might involve selecting an authentication protocol, designing a login interface, and implementing a secure password storage mechanism.

Problem and Solution Space in Practice

In practice, the differentiation between problem and solution space is important, but seldomly a linear process but rather a back and forth as also reflected by modern software development methodologies such as Scrum. It can be described as a continuous back and forth between problem space and solution space. That is, with every requirement, we narrow down the solution scope and with every design decision, we exclude further potential requirements. For example, once we estblish security-relevant requirements (for example log-in mechanisms), we narrow down the options in the solution and once we make design decisions to implement the security requirements, we may narrow down the options for other requirements (such as usability-relevant ones).



Problem and Solution Space interactions in Waterfall and Iterative Software Development

From Business Need to Implementation

As we have already seen, requirements engineering starts with the problem space by answering the WHAT (to do) and later proceeds to the solution space covering the HOW (to do it). This also reflects how requirements emerge. At first, a business need or constraint is formulated describing the WHAT on a high-level focusing on the WHY certain things are important. From this business need or constraint, more specific requirements are derived describing in detail the WHAT. Once the WHAT is described in sufficient detail, we are able to proceed to the HOW. We represent these different abstractions in three layers:

Context Layer (“Why“)

Captures goals and the operational context of a system with no immediate relationship to the actual system and its usage. Here, we aim at understanding typical processes and concepts of the domain or tasks that are carried out in an organisation for which we need to develop (or replace) a system. A profound understanding of the operational context sets the foundation for identifying relevant requirements and prioritising them, as we understanding why those requirements are important.

Requirements Layer (“What”)

Captures user-visible system behaviour with immediate, measurable (testable) relationship to the actual system. Here, we take a black-box view on the entire system with its intended behavior from a user perspective, building the intersection between problem and solution domain. We pay little to no attention to how the system is internally realised.

System Layer (“How“)

Captures a glass-box view of the system including functions and their internal realisation. Here, we may outline how the software system should be realized in terms of interactions by describing the solution domain.

While we will see those abstraction layers again in ID 1.1.1.6 - Functional Requirements and ID 1.1.1.7 - Non-functional Requirements, we may already note here that only the first two layers are in scope of Requirements Engineering as we describe the problem space by answering what shall be done and why it shall be done.

Common confusion and pitfalls

Confusion often arises when stakeholders focus on implementation details before fully understanding and agreeing the requirements. This can lead to a number of problems:

- **Overlooking stakeholder needs:** Discussing solutions prematurely can cause stakeholders to miss the real needs, resulting in solutions that don't address the right problems.
- **Gold plating:** Additional functionality may be added without being justified by precise requirements, increasing complexity and cost (unnecessarily).
- **Inadequate solutions:** Even technically sound solutions can fall short if the underlying requirements are unclear or undefined.
- **Missed opportunities:** Prematurely jumping to solutions can also lead to missed opportunities to explore alternative approaches that might better meet the requirements.

Example

Think of two-factor authentication (2FA) as you have already seen on several websites, a security process where users provide two different authentication factors to verify their identity. For example, the first step requires you to provide a password, and the second step requires a security token generated with a special application.

If you haven't thought about your users needs in the 'problem space', and you just decide that 2FA is what you need for your site, you might overlook the fact that your users usually order some groceries with an average cheque size of €30 which is paid with an external service (having its own security mechanisms). So even though the security issue might be very important, these render usability of the system cumbersome. And if you bore your users too often with complex login processes that eventually are not as relevant, they might end up simply going to your competitors.

[ENGLISCH] KEY TAKEAWAY

- **Prioritise problem definition:** Before jumping into solutions, it is crucial to clearly understand the problem space. This will significantly contribute to the success of the project.
- **Requirements drive solutions:** defining requirements and thinking about the user needs is necessary. You will receive the best solution based on what is required for this project.
- **Avoid premature solution discussions:** stakeholders should focus on thorough requirements elicitation and validation. This approach will help avoid common pitfalls, prevent overcomplicating the project, and create inadequate solutions.

GERMAN

[GERMAN] DESCRIPTION

In diesem Learningabschnitt diskutieren wir die Unterscheidung zwischen dem Problemraum und dem Lösungsraum in der Projektentwicklung. Wir lernen, warum es wichtig ist, ein Problem genau zu definieren, bevor man über Lösungsmöglichkeiten nachdenkt.

[GERMAN] GOALS AND VALUE

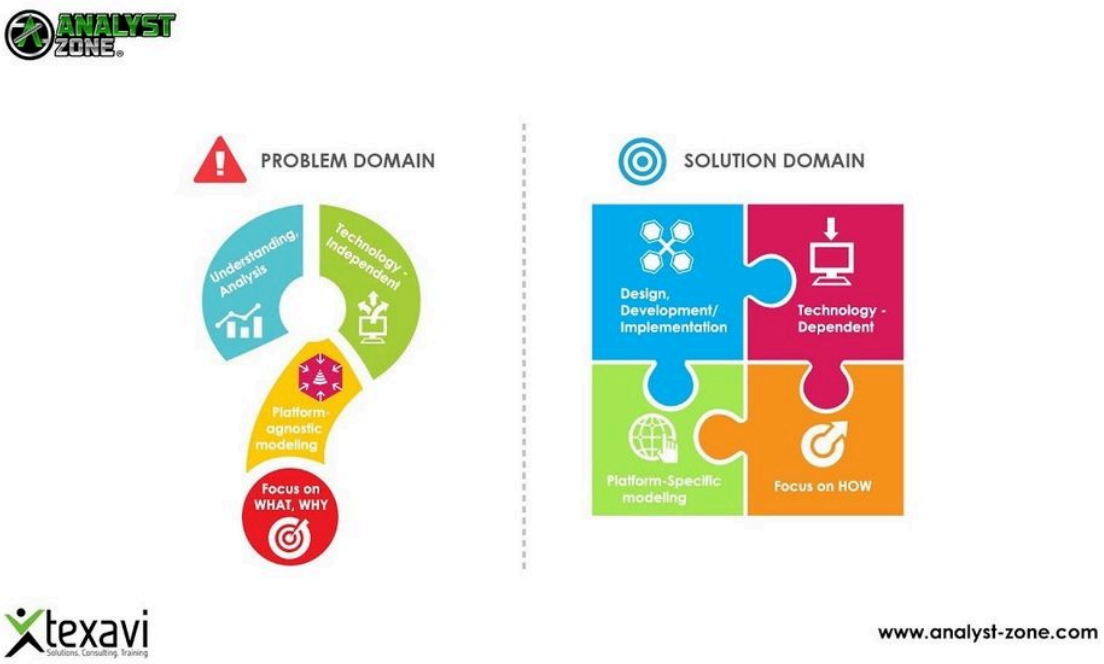
In diesem Lernmodul erfahren Sie ...

- Wie sich Problemraum und Lösungsraum in der Projektentwicklung unterscheiden.
- Wie eine vorschnelle Konzentration auf Umsetzungsdetails zu Verwirrung führen kann.
- Welche Strategien Sie zur erfolgreichen Navigation durch den Prozess verwenden können.

[GERMAN] CONTENT

Es ist wichtig, die Frage, WAS für ein Problem gelöst werden soll, von der Frage, WIE es gelöst werden soll, zu trennen. Durch diese Trennung schaffen wir mehr Klarheit und eröffnen mehr Optionen. Wir werden uns diese Konzepte nun genauer ansehen.

Verstehen des Problems



Bevor man sich auf Lösungen stürzt, ist es wichtig, dass man das Problem genau kennt. Wenn man das Problem zuerst versteht, ist sichergestellt, dass die Bemühungen in die richtige Richtung gelenkt werden und Verwirrung und Ineffizienz vermieden werden.

Anforderungen: Was ist das zu lösende Problem?

Anforderungen beschreiben, was ein Produkt oder eine Dienstleistung leisten muss oder innerhalb welcher Grenzen sie funktionieren müssen. Wir definieren die Bedürfnisse und Ziele des Systems, indem wir die Bedürfnisse der Benutzer, die Unternehmensziele oder die Marktanforderungen verstehen. Wir ermitteln die prägnante Beschreibung der beabsichtigten Funktionen und des Verhaltens des Systems. Zu diesem Zeitpunkt müssen wir uns noch keine Gedanken über die spezifischen Technologien, Plattformen und anderen Details der Implementierung machen. Wir müssen uns nicht auf etwas Bestimmtes festlegen. Ein konkretes Beispiel ist die Analyse der Benutzeranforderungen, ohne an die Interna eines Systems zu denken bzw. das System als Blackbox zu behandeln, um die Anforderungen zu lösen.

Beispiel

Eine neue Anforderung an eine Online-Einkaufsplattform könnte darin bestehen, dass sich die Benutzer bei dem System anmelden müssen, um eine Bestellung aufzugeben. Dieser Anmeldevorgang sollte schnell, einfach und sicher sein.

Erarbeitung von Lösungen

Sobald das Problem verstanden ist, ist es an der Zeit, darüber nachzudenken, wie man es lösen kann.

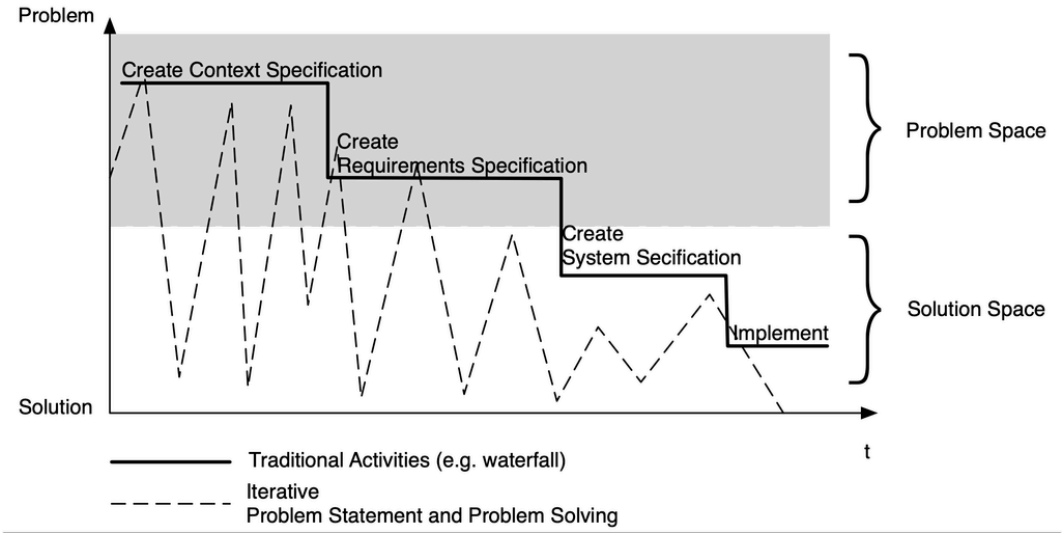
Lösungen: Wie man es erreicht

Lösungen befassen sich mit dem „Wie“, indem sie die Implementierungsspezifika beschreiben, die zur Erfüllung der Anforderungen erforderlich sind. Beim Nachdenken über Lösungen werden verschiedene Methoden, Designs, Architekturen und Implementierungen in Betracht gezogen. Hier denken wir über die interne Umsetzung nach, wie das System funktionieren sollte, um die Anforderungen zu erfüllen.

> Beispiel
>
> Eine Lösung für die obige Anforderung könnte die Auswahl eines Authentifizierungsprotokolls, den Entwurf einer Anmeldeschnittstelle und die Implementierung eines sicheren Mechanismus zur Speicherung von Passwörtern umfassen.

Problem- und Lösungsraum in der Praxis

In der Praxis ist die Unterscheidung zwischen Problem- und Lösungsraum wichtig, aber selten ein linearer Prozess, wie moderne Softwareentwicklungsmethoden wie Scrum zeigen. Vielmehr lässt er sich als ein ständiges Hin und Her zwischen Problemraum und Lösungsraum beschreiben. Das heißt, mit jeder Anforderung grenzen wir den Lösungsraum ein und mit jeder Designentscheidung schließen wir weitere potenzielle Anforderungen aus.



Interaktionen zwischen Problem- und Lösungsraum in der Wasserfall- und iterativen Softwareentwicklung

Vom Geschäftsbedarf zur Implementierung

Wie wir bereits gesehen haben, beginnt das Requirements Engineering im Problemraum mit der Beantwortung des WAS und geht später in den Lösungsraum über, der das WIE umfasst. Dies spiegelt auch wider, wie Anforderungen entstehen. Zunächst wird ein Geschäftsbedarf oder eine Einschränkung formuliert, der/die das WAS auf einer hohen Ebene beschreibt und sich auf das WARUM konzentriert. Aus diesem geschäftlichen Bedarf oder Zwang werden spezifischere Anforderungen abgeleitet, die das WAS im Detail beschreiben. Sobald das WAS hinreichend detailliert beschrieben ist, gehen wir zum WIE über. Wir stellen diese verschiedenen Abstraktionen in drei Ebenen dar:

Kontextebene („Warum“)

Erfasst die Ziele und den betrieblichen Kontext ohne unmittelbaren Bezug zum eigentlichen System und dessen Nutzung. Beschreibt das Verständnis von Prozessen und Aufgaben im Problemraum.

Anforderungsebene („Was“)

Erfasst das für den Benutzer sichtbare Systemverhalten mit unmittelbarem, messbarem (testbarem) Bezug zum eigentlichen System. Beschreibt das Black-Box-Verhalten des Systems und bildet die Schnittmenge zwischen Problem- und Lösungsraum.

Systemebene („Wie“)

Erfasst die Systemfunktionen und ihre interne Realisierung. Umreißt, wie das Softwaresystem in Form von Interaktionen realisiert werden sollte, indem die Lösungsdomäne beschrieben wird.

Wir werden diese Abstraktionsebenen wieder sehen in ID 1.1.1.6 - Functional Requirements und ID 1.1.1.7 - Non-functional Requirements.

Häufige Unklarheiten und Fallstricke

Verwirrung entsteht oft, wenn sich die Beteiligten auf die Einzelheiten der Umsetzung konzentrieren, bevor sie die Anforderungen vollständig verstanden und vereinbart haben. Dies kann zu einer Reihe von Problemen führen:

- **Übersehen der Bedürfnisse der Stakeholder:** Eine verfrühte Diskussion von Lösungen kann dazu führen, dass die Beteiligten die tatsächlichen Bedürfnisse nicht erkennen, was zu Lösungen führt, die nicht die richtigen Probleme angehen.
- **Ausufernder Umfang:** Zusätzliche Funktionen können ohne genaue Anforderungen hinzugefügt werden, was die Komplexität und die Kosten erhöht.
- **Unzureichende Lösungen:** Selbst technisch solide Lösungen können unzureichend sein, wenn die zugrunde liegenden Anforderungen unklar oder undefiniert sind.
- **Verpasste Gelegenheiten:** Wenn man sich vorschnell auf eine Lösung festlegt, kann man auch die Gelegenheit verpassen, alternative Ansätze zu erkunden, die den Anforderungen besser gerecht werden könnten.

Beispiel

Stellen Sie sich vor, Sie haben auf mehreren Websites die Zwei-Faktor-Authentifizierung (2FA) gesehen, ein Sicherheitsverfahren, bei dem Benutzer zwei verschiedene Authentifizierungsfaktoren angeben, um ihre Identität zu überprüfen. Wenn Sie sich keine Gedanken über die Bedürfnisse Ihrer Benutzer in diesem „Problembereich“ gemacht haben und einfach entscheiden, dass 2FA das ist, was Sie für Ihre Website brauchen, könnten Sie die Tatsache übersehen, dass Ihre Benutzer normalerweise Lebensmittel mit einem durchschnittlichen Wert von 30 € bestellen. Obwohl das Thema Sicherheit also sehr wichtig ist, bezweifeln Sie, dass die Nutzer zu vorsichtig damit umgehen werden. Und wenn Sie Ihre Nutzer zu oft mit komplizierten Anmeldeverfahren langweilen, werden sie einfach zu Ihren Konkurrenten gehen.

[GERMAN] KEY TAKEAWAY

- **Der Problemdefinition Vorrang einräumen:** Bevor man sich auf Lösungen stürzt, ist es von entscheidender Bedeutung, den Problembereich klar zu verstehen. Dies wird wesentlich zum Erfolg des Projekts beitragen.
- **Anforderungen treiben Lösungen voran:** Es ist notwendig, Anforderungen zu definieren und über die Bedürfnisse der Nutzer nachzudenken. Sie erhalten die beste Lösung auf der Grundlage dessen, was für dieses Projekt erforderlich ist.
- **Vermeiden Sie voreilige Lösungsdiskussionen:** Die Beteiligten sollten sich auf eine gründliche Anforderungsermittlung und -validierung konzentrieren. Auf diese Weise lassen sich häufige Fallstricke vermeiden und verhindern, dass das Projekt zu kompliziert wird und unzureichende Lösungen entstehen.

ID 1.1.1.5 - Types of Requirements

Meta data

	EN	DE
Status	DRAFT	
Title	Types of Requirements	Anforderungstypen
Tags	• Functional Requirements • Non-functional Requirements • Business Constraints	• Funktionale Anforderung • Nicht-funktionale Anforderung
Estimated Duration	5 min	
Level	☑ BEGINNER ☐ INTERMEDIATE ☐ EXPERT	
External ressources		
Dependencies		
Next Item	ID 1.1.1.6 - Functional Requirements ID 1.1.1.7 - Non-functional Requirements	
Goals (Teaching perspective)	To familiarize learners with the various categories of requirements and their respective characteristics.	
Content description (Teaching perspective)	Overview of different types of requirements such as business, functional and non-functionalrequirements.	

ENGLISCH

[ENGLISCH] DESCRIPTION

In this learning nugget we will get an overview of the different types of requirements such as business, functional, and non-functional requirements.

[ENGLISCH] GOALS AND VALUE

In this Learning Nugget you will...

- Learn about key project development documents and their purpose.
- Understand the different types of requirements.

[ENGLISCH] CONTENT

Common Requirements Engineering Documents

Before we look at the specific types of requirements, it's important to understand the various documents that are typically used to store these requirements during software development. Please note that this is not an exhaustive list and may vary depending on the domain, project type, etc. However, you may have come across some of the following documents in your own projects. Important to understand is that these documents may well be related to the previously introduced problem and solution space.

1. **Requirements Specification Document (Lastenheft):** This document outlines client expectations, serving as a guiding compass for project development. It details desired functionalities, features, and constraints, ensuring alignment among stakeholders and clarity in project objectives. It typically reflects both the WHAT and the WHY.
2. **Requirements Analysis (Anforderungsanalyse)**: This document focuses on analyzing and prioritizing requirements, often involving stakeholders to ensure alignment with business goals and user needs. Typically, this docuement concentrates on the WHAT:
3. **Functional Specification (Pflichtenheft)**: This document details the functional requirements of a project, often derived from the Requirements Specification. It describes HOW the system or product should behave or perform to meet the needs outlined in the Requirements Specification.
4. **System Specification (Systemspezifikation)**: This document provides a more technical perspective, outlining the system architecture, interfaces, and technical constraints. It serves as a blueprint for developers and engineers to understand the technical aspects of the project.
5. **Change Documentation (Änderungsdokumentation)**: Throughout the project lifecycle, changes to requirements are inevitable. This document tracks and documents any changes made to the initial requirements specifications, ensuring transparency and traceability.

Some of the documents emphasize high-level objectives and goals, whereas others detail the system's functionalities or the quality of these functions and the development processes. Typically, these requirements are documented using various methods and separated into different types of requirements. The ones that we will explore further. In later learning nuggets, we will introduce a clear structure (in the form of general artefacts) that is well reflected by the above documents.

Types of Requirements

1. High-level Goals:

High-level goals and constraints represent the collective needs and desires of stakeholders, reflecting their strategic vision and operational objectives. They serve as a blueprint for the ultimate scope of a project, outlining overall goals, specific objectives and expected benefits (giving answers to the WHY questions). They provide guidance for decision making and resource allocation, and align the project trajectory with the broader strategy. They navigate the project team to meet overall business objectives and stakeholder expectations, ensuring tangible value and business growth.

Examples:

- Market expansion: The system should support multiple languages to enable international customers to access and use the platform seamlessly.
- Customer support: A dedicated customer support portal must be integrated into the system, allowing users to submit queries and receive timely responses.
- Compliance: The system should comply with GDPR regulations, ensuring that user data is collected and processed in a lawful and transparent manner.

2. Functional Requirements:

Functional requirements describe WHAT the system should do and what features it should have. They specify the actions or behaviour expected of the system. Ideally, functional requirements are rooted in goals while this type of requirement should be clearly measurabe and later tested. Here are some examples:

Examples:

- Data processing: The system should be able to sort and filter large data sets of up to 1 million records within 5 seconds.
- User authentication: Users should be able to log in using either their email address or username, with a minimum password length of 8 characters.
- Reporting: The system should generate monthly financial reports in PDF format detailing revenues, expenses and profit margins.

3. Non-functional requirements:

Non-functional requirements (NFRs) complement functional requirements with a set of criteria that defines how a system should behave and how the development process should be carried out. Unlike functional requirements, which specify 'what' the system should do, NFRs focus on 'how' the system should behave and what qualities it should have. They are primarily concerned with the system's performance, constraints and process qualities, ensuring that the system is efficient, operable, secure and maintainable. In later learning nuggets, we will differentiate NFRs further as they may well address very different concerns.

Examples:

- Scalability: The system should be scalable to accommodate a 50% increase in user traffic without performance degradation.
- Testing: A comprehensive suite of automated tests covering at least 80% of the code must be run before any deployment to production.
- Dependency: The software must be compatible with the latest version of the company's standard database management system

Quiz

What type of requirement is this?

1. "The system should be able to process 1000 transactions per second".
2. "All user input should be validated to prevent incorrect data entry".
3. "The system should allow users to search for products by category and filter the results by price, brand and customer reviews."
4. "The project team should adhere to agile development methods, including sprints and daily stand-up meetings."
5. "The new website should increase online sales by 25% within the first quarter after launch."

Answers:

1. Non-functional requirement
2. Functional requirement
3. Functional requirement
4. Non-functional requirement
5. Business requirement

[ENGLISCH] KEY TAKEAWAY

- Product development uses a variety of documents to guide the direction of the project, define the behaviour of a system, outline technical aspects, and track changes in requirements for transparency.
- Business requirements outline the overall goals and expected benefits from a business perspective, providing context for the project objectives.
- It is important to differentiate between functional and non-functional requirements. One critically important reason is that different types of requirements are handled very differently (and they are verified differently).
- Functional requirements define the specific actions or behaviours expected of the system during project development.
- Non-functional requirements include quality attributes and process rules, emphasising characteristics beyond mere functionality.

GERMAN

[GERMAN] DESCRIPTION

In dieser Lernabschnitt erhalten wir einen Überblick über die verschiedenen Arten von Anforderungen wie geschäftliche, funktionale und nicht-funktionale Anforderungen.

[GERMAN] GOALS AND VALUE

In diesem Lernmodul erfahren Sie ...

- Welchen Zweck die wichtigsten Projektentwicklungsdokumente haben.
- Welche verschiedenen Arten von Anforderungen es gibt.

[GERMAN] CONTENT

Allgemeine Dokumente in der Anforderungsentwicklung

Bevor wir uns mit den spezifischen Arten von Anforderungen beschäftigen, ist es wichtig, die verschiedenen Dokumente zu verstehen, die typischerweise verwendet werden, um diese Anforderungen während der Softwareentwicklung zu dokumentieren. Bitte beachten Sie, dass diese Liste nicht vollständig ist und je nach Bereich, Projekttyp usw. variieren kann. Möglicherweise sind Sie jedoch in Ihren eigenen Projekten bereits auf einige der folgenden Dokumente gestoßen:

1. **Anforderungsspezifikationsdokument (Lastenheft):** Dieses Dokument umreißt die Erwartungen des Kunden und dient als Kompass für die Projektentwicklung. Es beschreibt die gewünschten Funktionalitäten, Merkmale und Einschränkungen und gewährleistet die Abstimmung zwischen den Beteiligten und die Klarheit der Projektziele.
2. **Anforderungsanalyse (Requirements Analysis):** Dieses Dokument konzentriert sich auf die Analyse und Priorisierung von Anforderungen, hierfür werden häufig alle Beteiligten einbezogen, um die Übereinstimmung mit den Unternehmenszielen und den Benutzeranforderungen sicherzustellen.
3. **Funktionale Spezifikation (Pflichtenheft):** In diesem Dokument werden die funktionalen Anforderungen eines Projekts detailliert beschrieben, oft abgeleitet von der Anforderungsspezifikation. Es beschreibt, wie sich das System oder das Produkt verhalten oder funktionieren soll, um die in der Anforderungsspezifikation genannten Anforderungen zu erfüllen.
4. **System-Spezifikation (Systemspezifikation):** Dieses Dokument bietet eine eher technische Perspektive und beschreibt die Systemarchitektur, die Schnittstellen und die technischen Beschränkungen. Es dient als Vorlage für Entwickler und Ingenieure, um die technischen Aspekte des Projekts zu verstehen.
5. **Änderungsdokumentation (Change Documentation):** Während des gesamten Projektlebenszyklus sind Änderungen an den Anforderungen unvermeidlich. Dieses Dokument verfolgt und dokumentiert alle Änderungen an den ursprünglichen Spezifikationen und gewährleistet so Transparenz und Nachvollziehbarkeit.

Einige dieser Dokumente betonen die Unternehmensziele, während andere die Funktionalitäten des Systems oder die Qualität dieser Funktionen und die Entwicklungsprozesse detailliert beschreiben. In der Regel werden diese Anforderungen mit verschiedenen Methoden dokumentiert und in verschiedene Arten von Anforderungen unterteilt. Diese werden wir nun näher untersuchen.

Arten von Anforderungen

1. Geschäftsziele:

Geschäftsziele sind die kollektiven Bedürfnisse und Wünsche der Unternehmensbeteiligten und spiegeln deren strategische Vision und operative Ziele wider. Sie dienen als Blaupause für ein Projekt, indem sie Gesamtziele, spezifische Ziele und erwartete Vorteile umreißen. Sie sind Richtschnur für die Entscheidungsfindung und die Ressourcenzuweisung und stimmen den Projektverlauf mit der allgemeinen Unternehmensstrategie ab. Sie steuern das Projektteam so, dass es übergreifende Erwartungen der Stackeholder und des Unternehmens erfüllt und einen konkreten Nutzen sowie Wachstum für das Unternehmen gewährleistet.

Beispiele:

- **Markterweiterung:** Das System sollte mehrere Sprachen unterstützen, damit internationale Kunden nahtlos auf die Plattform zugreifen und sie nutzen können.
- **Kundenbetreuung:** Ein spezielles Kundensupport-Portal muss in das System integriert werden, damit die Benutzer Anfragen stellen und zeitnahe Antworten erhalten können.
- **Einhaltung der Vorschriften:** Das System sollte den GDPR-Vorschriften entsprechen und sicherstellen, dass Nutzerdaten auf rechtmäßige und transparente Weise erfasst und verarbeitet werden.

2. Funktionale Anforderungen:

Funktionale Anforderungen beschreiben, was das System tun soll und welche Eigenschaften es haben soll. Sie spezifizieren die Aktionen oder das erwartete Verhalten des Systems. Normalerweise kann diese Art von Anforderungen gemessen und getestet werden. Hier sind einige Beispiele:

Beispiele:

- **Datenverarbeitung:** Das System sollte in der Lage sein, große Datensätze mit bis zu 1 Million Datensätzen innerhalb von 5 Sekunden zu sortieren und zu filtern.
- **Benutzerauthentifizierung:** Die Benutzer sollten sich entweder mit ihrer E-Mail-Adresse oder ihrem Benutzernamen anmelden können, wobei das Passwort mindestens 8 Zeichen lang sein muss.
- **Berichte:** Das System sollte monatliche Finanzberichte im PDF-Format mit detaillierten Angaben zu Einnahmen, Ausgaben und Gewinnspannen erstellen.

3. Nicht-funktionale Anforderungen:

Nicht-funktionale Anforderungen (NFR) sind eine Reihe von Kriterien, die festlegen, wie sich ein System verhalten soll und wie der Entwicklungsprozess ablaufen soll. Im Gegensatz zu den funktionalen Anforderungen, die festlegen, „was“ das System tun soll, konzentrieren sich die NFR darauf, „wie“ sich das System verhalten soll und welche Eigenschaften es haben soll. Sie befassen sich in erster Linie mit der Leistung, den Einschränkungen und den Prozessqualitäten des Systems und stellen sicher, dass das System effizient, funktionsfähig, sicher und wartbar ist.

Beispiele::

- Skalierbarkeit: Das System sollte so skalierbar sein, dass es eine 50%ige Zunahme des Benutzerverkehrs ohne Leistungseinbußen bewältigen kann.
- Testen: Eine umfassende Reihe von automatisierten Tests, die mindestens 80 % des Codes abdecken, muss vor dem Einsatz in der Produktion durchgeführt werden.
- Abhängigkeiten: Die Software muss mit der neuesten Version des Standard-Datenbankmanagementsystems des Unternehmens kompatibel sein.

Quiz

Welche Art von Anforderung ist dies?

1. „Das System sollte 1000 Transaktionen pro Sekunde verarbeiten können“.
2. „Alle Benutzereingaben sollten validiert werden, um eine falsche Dateneingabe zu verhindern“.
3. „Das System sollte es den Benutzern ermöglichen, Produkte nach Kategorien zu suchen und die Ergebnisse nach Preis, Marke und Kundenrezensionen zu filtern.
4. „Das Projektteam sollte sich an agile Entwicklungsmethoden halten, einschließlich Sprints und täglichen Stand-up-Meetings.
5. „Die neue Website sollte den Online-Umsatz innerhalb des ersten Quartals nach dem Start um 25 % steigern.“

Antworten:

1. Nicht-funktionale Anforderung
2. Funktionale Anforderung
3. Funktionale Anforderung
4. Nicht-funktionale Anforderung
5. Geschäftsziele

💡 [GERMAN] KEY TAKEAWAY

- Bei der Produktentwicklung wird eine Vielzahl von Dokumenten verwendet: um die Richtung des Projekts vorzugeben, das Verhalten eines Systems zu definieren, technische Aspekte zu umreißen und transparent Änderungen der Anforderungen nachzuverfolgen.
- Geschäftsziele umreißen die Gesamtziele und den erwarteten Nutzen aus geschäftlicher Sicht und liefern den Kontext für die Projektziele.
- Funktionale Anforderungen definieren die spezifischen Aktionen oder Verhaltensweisen, die von dem System während der Projektentwicklung erwartet werden.
- Zu den nichtfunktionalen Anforderungen gehören Qualitätsattribute und Prozessregeln, die Merkmale hervorheben, die über die reine Funktionalität hinausgehen.

ID 1.1.1.6 - Functional Requirements

Meta data

	EN	DE
Status	DRAFT	
Title	Functional Requirements	Funktionale Anforderungen
Tags	• functional requirement	• funktionale anforderung
Estimated Duration	10 min	
Level	☑ BEGINNER ☐ INTERMEDIATE ☐ EXPERT	
External ressources	• Formulation Guide (Guide towards defining requirements in natural language)	• Formulation Guide (Auf Englisch)
Dependencies	ID 1.1.1.5 - Types of Requirements	
Next Item	ID 1.1.1.7 - Non-functional Requirements	
Goals (Teaching perspective)	To understand what functional requirements are and how they integrate into requirement engineering	
Content description (Teaching perspective)	• Definition of a functional requirement • Introduction of functional behaviour abstractions • Set functional requirements in the context of use cases and scenarios • Describe how to formulate functional requirements • Detail functional behavior in agile software engineering	

ENGLISCH

[ENGLISCH] DESCRIPTION

In this learning nugget we will investigate the notion of a functional requirement, discuss its relevance in requirements engineering, and see examples of functional requirements.

[ENGLISCH] GOALS AND VALUE

In this learning nugget you will …

- Understand what a functional requirement is.
- Explore how functional requirement relate to business goals and constraints.
- Get insights into how functional requirements are represented in agile software development.

[ENGLISCH] CONTENT

Definition

> A functional requirement describes the external (user-visible) system behaviour, characterised by a stimulus (input) and a reaction (output).
>
> Source: A. Davis: Software Requirements: Objects, Functions & States

Functional requirements answer the question of what the software is supposed to do by **specifying** the **systems behaviour**. Often this represented as

“When [condition], the system shall do [xyz]“

e.g.

When the customer initiates to withdraw money, the ATM shall ask for a debit card and a PIN for authentication.

Functional requirements exist in the context of the functional behaviour of the system and can be specified on the previously introduced three layers of abstractions, each tailored towards the perspectives of the needs of different stakeholders (e.g. from high-level goals and constraints to low-level feature descriptions). The three layers are are described below. We set functional requirements into the context of those abstraction layers, and highlight how the functional system behaviour is documented in agile software engineering.

Functional Requirements from Business goals to Implementation

When we dicussed the ID 1.1.1.4 - Problem space vs solution space, you already got acquinted with the different abstraction layers of requirements. Like other requirements, the functional behaviour of a system can be traced from business goals to implementation through these layers. In the following, we showcase how the functional behavior can be traced from the WHY to the HOW.

Context Layer (“Why“)
At the highest abstraction layer of the needs and constraints, stakeholders such as business analyst define business goals, processes, constraints.
Example Requirement: Our customers should be able to manage their money through an ATM.

> Example
>
> Our customers should be able to manage their money through an ATM.

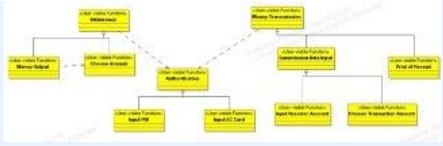
Requirements Layer (“What”)
Once the why has been established through goals, processes and constraints, we should tackle the question of what to implement. Typically, this results (among others) in use cases, user stories, and other sets of functional requirements.

> Example
>
> If the card is inserted, the ATM shall ask the customer for the PIN of their card.

System Layer (“How“)

On the lowest level, system architect or developers will formalize the features, internal function specification, and logical component architectures - to name a few.

> Example
>
> At this level, we may begin with designing which functions the system shall offer with which components, what the interfaces are and how the system will be internally organised via a component architecture.



Functional Requirements Context

To better understand functional requirements, we take a closer look at the requirements layer and its related outcomes: use cases, scenarios and functional requirements. These are all different (and complementary) ways of describing functional requirements at different levels of abstraction and granularity.

Use Cases integrate all possible scenarios, which might occur when an actor tries to achieve a specific usage goal in alignment with the business goals. An example of a use case is:

Example

Use Case Title: ATM Cash Withdrawal

Sequence:

1. The customer inserts their EC into the ATM.
2. The ATM checks the card and shows the available functions to the customer.
3. The ATM then prompts the customer to enter their PIN.
4. Once the customer inserts their PIN, the ATM checks whether the entered PIN is correct.
5. If the PIN is wrong, the ATM prints an error message, retains the card temporarily, and then returns the card to the customer.
6. If the correct PIN is entered, the customer is then prompted to choose the withdrawal amount.
7. The ATM checks the customer's account balance to ensure sufficient funds are available for the withdrawal.
8. If the balance is insufficient, the ATM will not proceed with the transaction and will likely notify the customer.
9. If there is a sufficient balance, the ATM dispenses the requested amount of money along with the customer's EC card.
10. The process ends once the customer takes the money and their card.

Scenarios describe an ordered set of interactions (i.e. functional requirements) between the system and a set of external actors (such as users) or other systems. As such, it describes one possible sequence of a use case. A scenario of the use case before looks like:

Example

Scenario Title: ATM Cash Withdrawal Authentication Error

Sequence:

1. The customer inserts their EC into the ATM.
2. The ATM checks the card and shows the available functions to the customer.
3. The ATM then prompts the customer to enter their PIN.
4. Once the customer inserts their PIN, the ATM checks whether the entered PIN is correct.
5. If the PIN is wrong, the ATM prints an error message, retains the card temporarily, and then returns the card to the customer.

Functional requirements are single (atomic) statements describing isolated functional properties. Functional requirements derived from the scenario above are:

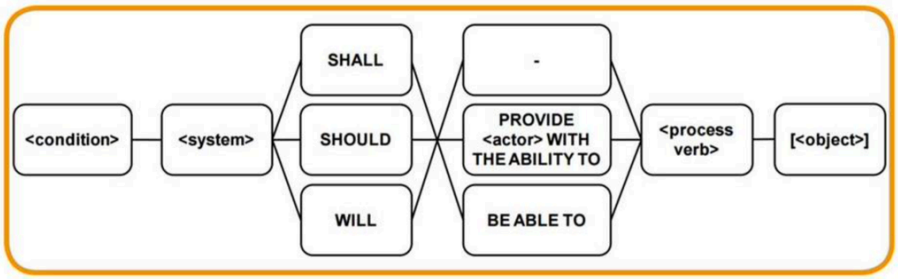
Example

- The ATM system shall allow the customer to enter their Personal Identification Number after inserting their card.

- Upon PIN entry, the ATM system shall validate the correctness of the PIN against the customer's encrypted PIN stored within the bank's secure database.
- If the entered PIN is incorrect, the system shall prompt the user to try again up to a specified number of attempts.

Formulation

There is no one only way to formulate functional requirements, let alone requirements. The Sophist group nonetheless offers a helpful guide towards formulating single functional requirements in natural language:



Pattern towards describing functional requirements in natural language. From The REUSE Company

Note that each statement formulated by following the above pattern may represent one single interaction step in a scenario which each scenario may be part of a collection of scenarios, which in turn is grouped by a single use case. This shows, again, how functional requirements can be represented at different (complementary) levels of granularity.

Functional Requirements in Agile

A common misconception (based on the principle “working software over comprehensive documentation”) is that agile software development neglects documentation and as such the trace from goals and constraints to more technical requirements is missing. However, given that the typical requirements artifacts in agile software development are

- Themes: Top-level objectives
- Epics: Higher-level functionality represented by a group of user stories
- User Stories: Brief statement of intent when interacting with the system
- Tasks: Implementation unit for developers/testers

we can well show a clear maping to goals and technical requirements:

- Themes describe the goals and requirements (Problem Space)
- User Stories can be considered as equivalent to Use Cases, both aggregating multiple system interactions into a higher-level functionality. However, given that user stories emphasize the intent of the interaction with the system, use cases are of higher granularity and they provide far more details (for which testers will likely be thankful). (Problem Space)
- Tasks result from user stories and as such can be guided by scenarios and implement functional requirements (Solution Space)

💡 [ENGLISCH] KEY TAKEAWAY

- Functional requirements exist in the context of the functional behavior of the system
- Functional requirements specify the systems observable behaviour as atomic statements prescribing isolated system properties
- Use cases comprise scenarios which contain functional requirements

GERMAN

[GERMAN] DESCRIPTION

In diesem Lernabschnitt untersuchen wir den Begriff der funktionalen Anforderung, diskutieren ihre Bedeutung für das Requirements Engineering und betrachten Beispiele.

[GERMAN] GOALS AND VALUE

In diesem Lernmodul erfahren Sie ...

- Was eine funktionale Anforderung ist.
- Wie sie mit den Geschäftszielen zusammenhängen.
- Wie funktionale Anforderungen in der agilen Softwareentwicklung dargestellt werden.

[GERMAN] CONTENT

Definition

> Eine funktionale Anforderung beschreibt das externe (für den Benutzer sichtbare) Systemverhalten, das durch einen Stimulus (Eingabe) und eine Reaktion (Ausgabe) gekennzeichnet ist.
>
> Quelle: A. Davis: Software Requirements: Objekte, Funktionen und Zuständefunctional

Funktionale Anforderungen beantworten die Frage, was die Software tun soll, indem sie das **Verhalten des Systems spezifizieren**. Häufig wird dies wie folgt dargestellt

„Wenn [Bedingung], soll das System [xyz] tun“

z.B..

Wenn der Kunde Geld abheben will, soll der Geldautomat nach einer Debitkarte und einer PIN zur Authentifizierung fragen.

Funktionale Anforderungen bestehen im Kontext des funktionalen Verhaltens des Systems und können auf drei Abstraktionsebenen spezifiziert werden, die jeweils auf die Perspektive der Bedürfnisse verschiedener Interessengruppen zugeschnitten sind (z. B. von High-Level-Geschäftszielen bis zu Low-Level-Funktionsbeschreibungen). Die drei Ebenen werden im Folgenden beschrieben. Wir stellen die funktionalen Anforderungen in den Kontext dieser Abstraktionsebenen und zeigen auf, wie das funktionale Systemverhalten in der agilen Softwareentwicklung dokumentiert wird.

Funktionale Anforderungen vom Geschäftsbedarf bis zur Implementierung

Im Learning Nugget ID 1.1.1.4 - Problem space vs solution space haben Sie die verschiedenen Abstraktionsebenen von Anforderungen kennengelernt. Wie andere Anforderungen kann auch das funktionale Verhalten eines Systems durch diese Schichten von unternehmerischen Vorgaben bis zur Implementierung verfolgt werden. Im Folgenden zeigen wir, wie sich das funktionale Verhalten vom WARUM zum WIE zurückverfolgen lässt.

Kontextebene („Warum“)

Auf der höchsten Abstraktionsebene der Anforderungen definieren Stakeholders wie Business Analysten Business Ziele, Prozesses, und Einschränkungen.

Example

Die Kunden sollen ihr Geld über einen Bankautomaten verwalten können.

Anforderungsebenes (“Was”)
Sobald das Warum durch Business Ziele, Prozesses und Einschränkungen formalisiert wurde beschäftigen sich Stakeholder wie

Requirements Engineers mit der Frage was es zu implementieren gilt. Typischerweise ist das Resultat (unter anderem) Use Cases, User Stories und funktionale Anforderungen.

Example

Wenn die Bankkarte in den Automat eingeführt wurde soll der Bankautomat den Kunden nach der PIN seiner Karte fragen.

Systemebene (“Wie“)

Auf dem niedrigsten Level, formalisieren Stakeholder wie Systemarchitekten oder Entwickler Features, interne Funktionsspezifikationen, und logische Komponentenarchitekturen - um ein paar zu nennen.

Example

Der Nutzer kann die letze Nummer seiner PIN während der Authentifizierung korrigieren.

Funktionale Anforderungen - Kontext

Um die funktionalen Anforderungen besser zu verstehen, werfen wir einen genaueren Blick auf die Anforderungsebene und die damit verbundenen Ergebnisse: Use Cases, Szenarien und funktionale Anforderungen.

Use Cases umfassen alle möglichen Szenarien, die auftreten können, wenn ein Akteur versucht, ein bestimmtes Nutzungsziel in Übereinstimmung mit den Geschäftszielen zu erreichen. Ein Beispiel für einen Use Case ist:

Beispiel

Use Case Titel: Bargeldabhebung am Geldautomaten

Ablauf:

1. Der Kunde führt seine EC-Karte in den Geldautomaten ein.
2. Der Geldautomat prüft die Karte und zeigt dem Kunden die verfügbaren Funktionen an.
3. Dann fordert der Geldautomat den Kunden auf, seine PIN einzugeben.
4. Sobald der Kunde seine PIN eingibt, prüft der Geldautomat, ob die eingegebene PIN korrekt ist.
5. Wenn die PIN falsch ist, druckt der Geldautomat eine Fehlermeldung aus, behält die Karte vorübergehend ein und gibt sie dann an den Kunden zurück.
6. Wenn die PIN richtig eingegeben wurde, wird der Kunde aufgefordert, den Abhebungsbetrag zu wählen.
7. Der Geldautomat überprüft den Kontostand des Kunden, um sicherzustellen, dass genügend Geld für die Abhebung vorhanden ist.
8. Wenn das Guthaben nicht ausreicht, führt der Geldautomat die Transaktion nicht durch und benachrichtigt den Kunden.
9. Ist ein ausreichendes Guthaben vorhanden, gibt der Geldautomat den angeforderten Geldbetrag zusammen mit der EC-Karte des Kunden aus.
10. Der Vorgang ist beendet, sobald der Kunde das Geld und seine Karte entnommen hat.

Szenarien beschreiben einen geordneten Satz von Interaktionen (d. h. funktionale Anforderungen) zwischen dem System und einer Reihe von externen Akteuren (z. B. Benutzern) oder anderen Systemen. Als solche beschreiben sie eine mögliche Abfolge eines Use Cases. Ein Szenario des vorherigen Use Cases sieht so aus:

Beispiel

Szenario Titel: Authentifizierungsfehler beim Abheben von Bargeld am Geldautomaten

Ablauf:

1. Der Kunde führt seine EC-Karte in den Geldautomaten ein.

2. Der Geldautomat prüft die Karte und zeigt dem Kunden die verfügbaren Funktionen an.
3. Dann fordert der Geldautomat den Kunden auf, seine PIN einzugeben.
4. Sobald der Kunde seine PIN eingibt, prüft der Geldautomat, ob die eingegebene PIN korrekt ist.
5. Wenn die PIN falsch ist, druckt der Geldautomat eine Fehlermeldung aus, behält die Karte vorübergehend ein und gibt sie dann an den Kunden zurück.

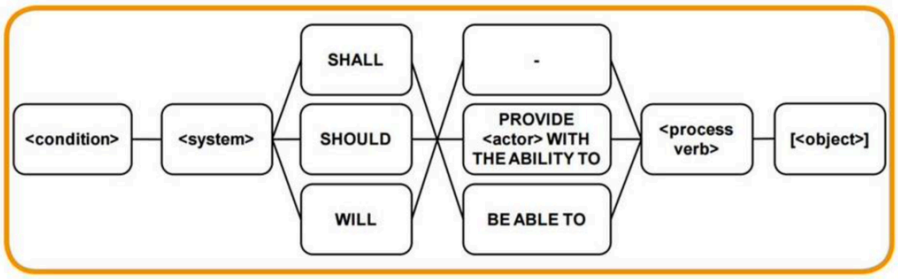
Funktionale Anforderungen sind einzelne (atomare) Aussagen, die isolierte funktionale Eigenschaften beschreiben. Funktionale Anforderungen, die aus dem obigen Szenario abgeleitet werden, sind:

Beispiel

- Das Geldautomatensystem muss es dem Kunden ermöglichen, nach dem Einstecken der Karte seine persönliche Identifikationsnummer einzugeben.
- Bei der PIN-Eingabe überprüft das Geldautomatensystem die Korrektheit der PIN anhand der verschlüsselten PIN des Kunden, die in der sicheren Datenbank der Bank gespeichert ist.
- Wenn die eingegebene PIN falsch ist, fordert das System den Benutzer auf, es bis zu einer bestimmten Anzahl von Versuchen erneut zu versuchen.

Formulierung

Es gibt nicht den einen richtigen Weg, um funktionale Anforderungen zu formulieren, geschweige denn Anforderungen. Die Sophist-Gruppe bietet jedoch eine hilfreiche Anleitung zur Formulierung von Anforderungen in natürlicher Sprache:



Muster für die Beschreibung von funktionalen Anforderungen in natürlicher Sprache. Von The REUSE Company

Funktionale Anforderungen in agiler Softwareentwicklung

Ein weit verbreitetes Missverständnis (basierend auf dem Prinzip „funktionierende Software vor umfassender Dokumentation“) ist, dass die agile Softwareentwicklung die Dokumentation vernachlässigt und somit die Nachvollziehbarkeit von den fachlichen Anforderungen zu den technischen Anforderungen fehlt. Wenn wir uns jedoch die typischen Anforderungsartefakte in der agilen Softwareentwicklung ansehen:

- Themes: Top-Level-Ziele
- Epics: Funktionalität auf höherer Ebene, dargestellt durch eine Gruppe von User Stories
- User Stories: Kurze Absichtserklärung bei Interaktion mit dem System
- Tasks: Implementierungseinheit für Entwickler/Tester,

können wir einen klaren Weg zwischen Geschäftszielen und technischen Anforderungen nachvollziehen:

- Themes beschreiben die geschäftlichen Ziele und Anforderungen (Problemraum)
- User Stories können als Äquivalent zu Use Cases betrachtet werden, da beide mehrere Systeminteraktionen zu einer übergeordneten Funktionalität zusammenfassen. Da User Stories jedoch die Absicht der Interaktion mit dem System betonen, sind

Use Cases von hoher Granularität (Problemraum)

- Tasks resultieren aus User Stories und können als solche von Szenarien abgeleitet werden und funktionale Anforderungen implementieren (Lösungsraum)

💡 [GERMAN] KEY TAKEAWAY

- Funktionale Anforderungen existieren im Kontext des funktionalen Verhaltens des Systems.
- Funktionale Anforderungen spezifizieren das beobachtbare Verhalten des Systems als atomare Aussagen, die isolierte Systemeigenschaften vorschreiben.
- Anwendungsfälle bestehen aus Szenarien, die funktionale Anforderungen enthalten.

ID 1.1.1.7 - Non-functional Requirements

Meta data

	EN	DE
Status	DRAFT	
Title	Non-functional Requirements	Nicht-funktionale Anforderungen
Tags	• Non-functional Requirements • Quality Requirements • Process Requirements • System Contraints	• Nicht funktionale anforderung • Qualitätsanforderung • Prozessanforderung • Systemeinschränkung
Estimated Duration	10 min	
Level	☐ BEGINNER ☑ INTERMEDIATE ☐ EXPERT	
External ressources	• Requirements Taxonomy following Glinz • Requirements Taxonomy according to ISO 25010	
Dependencies	ID 1.1.1.5 - Types of Requirements ID 1.1.1.6 - Functional Requirements	
Next Item	ID 1.1.1.8 - What makes a good requirement?	
Goals (Teaching perspective)	Give insights into non-functional requirements	
Content description (Teaching perspective)	• Characterize non-functional requirements • Insights into the diverse definitions of non-functional requirements and state NFR types • Trace of non-functional requirement from business need to technical requirements	

ENGLISCH

[ENGLISCH] DESCRIPTION

In this learning nugget we will discuss non-functional requirements, what they entail, and why measuring them is difficult.

[ENGLISCH] GOALS AND VALUE

In this learning nugget you will …

- Gain an understanding of non-functional requirements.
- Trace non-functional requirements from business needs to implementation tasks.
- Get first tips on how to define measurable non-functional requirements.

[ENGLISCH] CONTENT

Why non-functional Requirements?

Very often, you may encounter different systems that may be well comparable in terms of functionality. The key difference lies in more qualitative aspects that, in the end, may be seen as the final criterion for chosing one system over the other. This brings us to non-functional requirements. While functional requirements describe large parts of the intented system, other crucial needs, such as performance or maintainability of a system are not covered by them. This is where non-functional requirements come into play.

What are non-functional Requirements?

Non-funtional requirements aim to address all those needs that are not covered by functional requirements. Similar to functional requirements, non-functional requirements trace from high-level goals and constraints to more technical requirements prescribing their implementation. Note already here that the spectrum of what is covered by non-functional requirements is very broad covering both quality properties of the system under consideration and properties of the overall development process. While we may well differentiate better in later learning nuggets, here, for reasons of cimplicity, we focus on properties of the system itself.

To make things more complicated, there is, unfortunately, not one commonly accepted definition for what non-functional requirements eventually are. This is due to the fact that non-functional requirements cover a broad spectrum of needs and can overlap with some functionalities, such as system performance. Additionally, non-functional requirements might not always be isolated to individual functions (think of maintainability of a system). All these factors lead to different perspectives on non-functional requirements and have led in the past to a multiture of definitions, none of which may be considered very helpful.

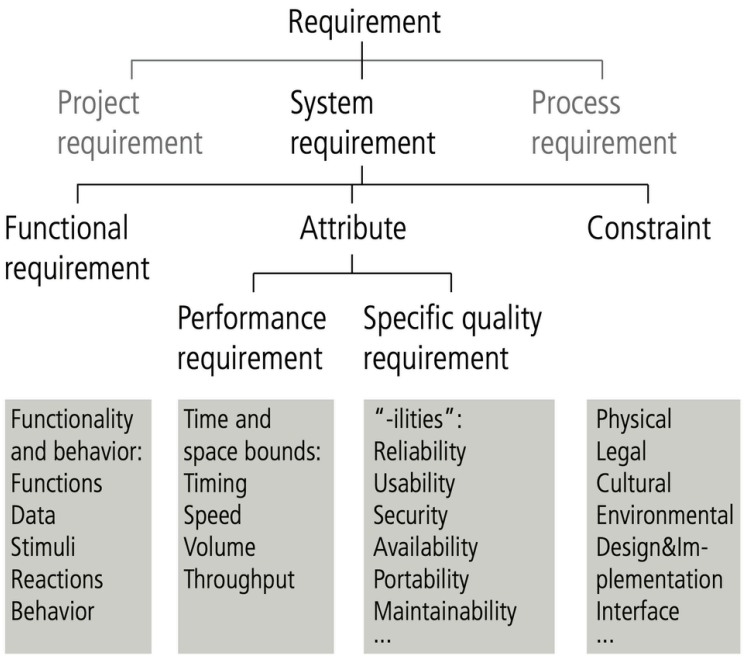
To give you a feeling of the different perspectives on non-funcational requirements, see below some of the definitions:

> Describe the nonbehavioral aspects of a system, capturing the properties and constraints under which a system must operate.
>
> *A. Antón (1997). Goal Identification and Refinement in the Specification of Information Systems. PhD Thesis, Georgia Institute of Technology*

> The behavioral properties that the specified functions must have, such as performance, usability.
>
> *C. Ncube (2000). A Requirements Engineering Method for COTS-Based Systems Development. PhD Thesis, City University London.*

> Requirements wich specify criteria that can be used to judge the operation of a system, rather than specific behaviors.
>
> *Wikipedia: Non-Functional Requirements* W Non-functional requirement

Available textual definitions can be vague and sometimes even contradict each other (e.g. behavioral vs. nonbehavior). Thus, a more explicit way of characterizing requirements is through taxonomies. In the following, we show two taxonomies that depict the boundaries between requirement types and characterize quality requirement types. In the end, we have to leave the decision which taxonomy to consider and how to adapt it to the individual organisations as the decision also comes along the operational context of the organisation and the application domain.



Requirements Taxonomy as defined by Glinz in https://doi.org/10.1109/RE2007.45

Software Product Quality Taxonomy as defined by the ISO 25010 ISO 25010

Types of non-functional Requirements

The taxonomies in the previous section emphasize the various categorizations of non-functional requirements, each with their own benefits. To give already here an initial and intuitive overview of different sub-classes of non-functional requirements, we categorize them into the following three types:

Quality Requirements (Summary of ID 1.1.1.5 - Types of Requirements)

Define quality characteristics and properties of a system that are (ideally) measurable regarding the system's behavior.

Example

- The system shall process and respond to 99% of user requests within 2 seconds under a load of 700 concurrent users.

Process Requirements (Summary of ID 1.1.1.5 - Types of Requirements)

Prescribe requirements for the development process and definitions or rules for the implementation.

Example

- The development process must be compliant to the IEC 62443-4-1 development standard.

System Constraints

These constraints concern restrictions on the solution space beyond what is necessary to meet functional, quality or process requirements. Examples can range from technical to legal aspects:

Example

- Technical: All interfaces must have a Javadoc-compatible documentation.
- Legal: System may not be available in Country XYZ dues to national regulations.

Non-functional Requirements from Goals to Implementation

Like functional requirements, non-functional requirements can be traced from business needs to fine-granular requirements. Taking the same layers as for the functional requirements, non-functional requirements can be traced as follows:

Context Layer (“Why”)

At the highest level, business stakeholders define the (abstract) needs for their systems and services. These needs set the values of a system and justify further requirements and their respective costs.

Example

- The system has to be maintainable.
- The system shall be secure.

Requirements Layer (“What”)

The abstract business needs are refined to describe the envision interactions, prescribing which activities shall be supported by the system, e.g. to enable maintenance. And which activities shall be prevented, e.g. to protect against attacks. The abstract business needs further enable definition of measurable quality characteristics of the system. Business needs can often be refined by asking “How/What”-questions?

Example

- What makes the system maintainable?
  - The documentation must be provided in english.
  - The user documentation must be provided in easy language.
- How should the system be secured?
  - User data must be encrypted following the recommendations of the BSI.
  - Data may only be transfered to authorized actors.

How to specify non-functional requirements?

Since functional requirements describe the (user) observable behavior of a system, these requirements can be tested in a straight-forward manner. The diverse spectrum of non-functional requirements, however, makes it far more difficult to measure the satisfaction of such requirements. Process requirements and system constraints are typically difficult to measure, however quality requirements are even harder to measure measure as the notion of quality is highly subjective, and it often lays in the eye of the beholder.

A few tips towards defining measurable non-functional requirements:

1. Trace requirements from abstract business needs to individual requirements.
   This way, the measurements can be easier defined and performed across all abstraction levels. E.g. the business need “The system shall be secure“ could be measured as the aggregation of non-functional requirements derived from it, such as “User data must be encrypted following the recommendations of the BSI”.

2. Refine requirements until they are actionable and measurable. E.g. the non-functional requirement “User data must be encrypted following the recommendations of the BSI“ needs further clarification and refinement e.g. regarding the type of user data or the encryption algorithms.

We will go into further details about requirements quality in ID 1.1.1.8 - What makes a good requirement?.

[ENGLISCH] KEY TAKEAWAY

- Non-functional requirements adress needs not coverd by functional requirements
- We differentiate into three non-functional requirement types
  - Quality requirements
  - Process requirements
  - System constraints
- Tracing and careful refining of non-functional requirements from business needs to implementation tasks facilitates measurement

GERMAN

[GERMAN] DESCRIPTION

In diesem Lernabschnitt sprechen wir über nicht-funktionale Anforderungen, was sie beinhalten und warum es schwierig ist, sie zu messen.

[GERMAN] GOALS AND VALUE

In diesem Lernmodul erfahren Sie ...

- Was nicht-funktionale Anforderungen sind
- Wie man nicht-funktionale Anforderungen von Geschäftszielen bis zu Implementierungs-Tasks verfolgt
- Tipps, wie man messbare nicht-funktionale Anforderungen definieren kann

[GERMAN] CONTENT

Warum nicht-funktionale Anforderungen?

Während funktionale Anforderungen große Teile des geplanten Systems beschreiben, werden entscheidende Anforderungen, wie z.B. die Leistungsfähigkeit oder Wartbarkeit eines Systems, von ihnen nicht abgedeckt. An dieser Stelle kommen nicht-funktionale Anforderungen ins Spiel.

Was sind nicht-funktionale Anforderungen?

Nicht-funktionale Anforderungen zielen darauf ab, all jene Bedürfnisse zu adressieren, die nicht durch funktionale Anforderungen abgedeckt sind. Ähnlich wie bei den funktionalen Anforderungen führen nicht-funktionale Anforderungen von den Geschäftsvorgaben zu den technischen Anforderungen, die deren Umsetzung vorschreiben.

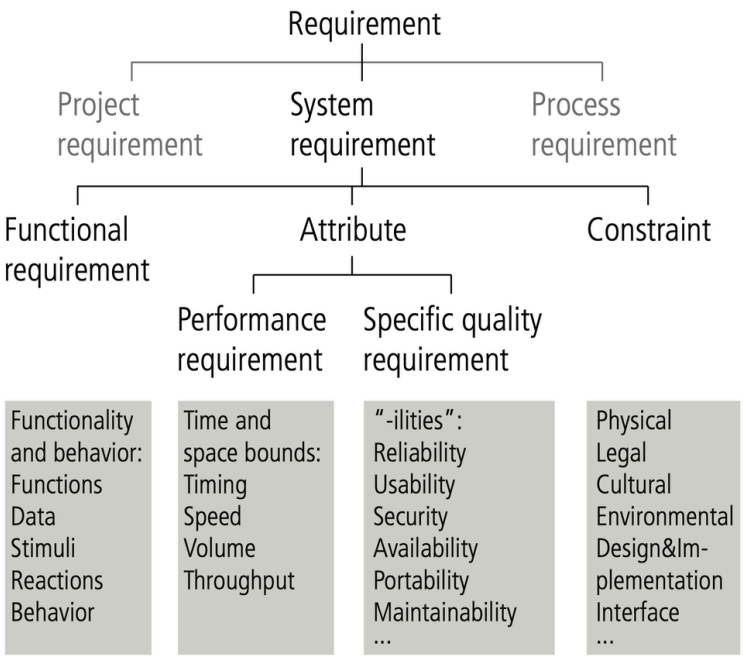
Leider gibt es nicht die eine, allgemein akzeptierte Definition. Nicht-funktionale Anforderungen decken ein breites Spektrum von Bedarfen ab und können sich mit einigen Funktionalitäten überschneiden, z. B. mit der Systemleistung. Außerdem lassen sich nicht-funktionale Anforderungen nicht immer auf einzelne Funktionen beschränken, z. B. auf die Wartbarkeit. All diese Faktoren führen zu unterschiedlichen Sichtweisen auf nicht-funktionale Anforderungen und zu einer Vielzahl von Definitionen.

Um Ihnen ein Gefühl für die verschiedenen Perspektiven zu nicht-funktionalen Anforderungen zu vermitteln, finden Sie hier einige Definitionen:

> Beschreibt die nicht-verhaltensbezogenen Aspekte eines Systems und erfasst die Eigenschaften und Beschränkungen, unter denen ein System funktionieren muss. (Übersetzt aus dem Englischen)
>
> A. Antón (1997). Goal Identification and Refinement in the Specification of Information Systems. Dissertation, Georgia Institute of Technology
>
> Die Verhaltenseigenschaften, die die spezifizierten Funktionen haben müssen, z. B. Leistung, Benutzerfreundlichkeit. (Übersetzt aus dem Englischen)
>
> C. Ncube (2000). A Requirements Engineering Method for COTS-Based Systems Development. Dissertation, City University London.

> Anforderungen, die Kriterien spezifizieren, die zur Beurteilung der Funktionsweise eines Systems herangezogen werden können, anstatt spezifischer Verhaltensweisen. (Übersetzt aus dem Englischen)
>
> Wikipedia: Nicht-funktionale Anforderungen W Non-functional requirement

Verfügbare Definitionen in Textform können vage sein und sich manchmal sogar widersprechen (z. B. Verhalten vs. Nicht-Verhalten). Eine explizitere Methode zur Charakterisierung von Anforderungen sind daher Taxonomien. Im Folgenden werden zwei Taxonomien vorgestellt, die die Grenzen zwischen Anforderungsarten darstellen und Qualitätsanforderungsarten charakterisieren.



Anforderungs-Taxonomie, wie sie von Glinz in https://doi.org/10.1109/RE2007.45 definiert wurde.

SOFTWARE PRODUCT QUALITY								
FUNCTIONAL SUITABILITY	PERFORMANCE EFFICIENCY	COMPATIBILITY	INTERACTION CAPABILITY	RELIABILITY	SECURITY	MAINTAINABILITY	FLEXIBILITY	SAFETY
FUNCTIONAL COMPLETENESS FUNCTIONAL CORRECTNESS FUNCTIONAL APPROPRIATENESS	TIME BEHAVIOUR RESOURCE UTILIZATION CAPACITY	CO-EXISTENCE INTEROPERABILITY	APPROPRIATENESS RECOGNIZABILITY LEARNABILITY OPERABILITY USER ERROR PROTECTION USER ENGAGEMENT INCLUSIVITY USER ASSISTANCE SELF-DESCRIPTIVENESS	FAULTLESSNESS AVAILABILITY FAULT TOLERANCE RECOVERABILITY	CONFIDENTIALITY INTEGRITY NON-REPUDIATION ACCOUNTABILITY AUTHENTICITY RESISTANCE	MODULARITY REUSABILITY ANALYSABILITY MODIFIABILITY TESTABILITY	ADAPTABILITY SCALABILITY INSTALLABILITY REPLACEABILITY	OPERATIONAL CONSTRAINT RISK IDENTIFICATION FAIL SAFE HAZARD WARNING SAFE INTEGRATION
iso25000.com								

Taxonomie der Software-Produktqualität gemäß der Definition in ISO 25010 ISO 25010

Arten von nicht-funktionalen Anforderungen

Die Taxonomien im vorangegangenen Abschnitt verdeutlichen die verschiedenen Kategorisierungen von nicht-funktionalen Anforderungen, die jeweils ihre eigenen Vorteile haben. Um einen intuitiven Überblick über nicht-funktionale Anforderungen zu geben, teilen wir sie in die folgenden drei Typen ein:

Qualitätsanforderungen (Zusammenfassung von ID 1.1.1.5 - Arten von Anforderungen)

Definieren Qualitätsmerkmale und Eigenschaften eines Systems, die (idealerweise) in Bezug auf das Verhalten des Systems messbar sind.

Beispiel

- Das System soll 99% der Nutzeranfragen innerhalb von 2 Sekunden unter einer Last von 700 gleichzeitigen Nutzenden verarbeiten und beantworten.

Prozessanforderungen (Zusammenfassung von ID 1.1.1.5 - Arten von Anforderungen)

Legen Anforderungen an den Entwicklungsprozess und Definitionen oder Regeln für die Implementierung fest.

Beispiel

- Der Entwicklungsprozess muss mit der Entwicklungsnorm IEC 62443-4-1 konform sein.

Systemeinschränkungen

Diese Einschränkungen betreffen den Lösungsraum, der über das hinausgeht, was zur Erfüllung von Funktions-, Qualitäts- oder Prozessanforderungen notwendig ist. Die Beispiele können von technischen bis zu rechtlichen Aspekten reichen:

Beispiel

- Technisch: Alle Schnittstellen müssen eine Javadoc-kompatible Dokumentation haben.
- Rechtlich: Das System darf in Land XYZ aufgrund nationaler Vorschriften nicht verfügbar sein.

Nicht-funktionale Anforderungen von den Geschäftszielen bis zur Implementierung

Wie die funktionalen Anforderungen lassen sich auch die nichtfunktionalen Anforderungen von den Geschäftszielen bis hin zu den feingranularen Anforderungen verfolgen. Ausgehend von denselben Ebenen wie bei den funktionalen Anforderungen können nichtfunktionale Anforderungen wie folgt verfolgt werden:

Kontextebene („Warum“)

Auf der obersten Ebene definieren die Geschäftsinteressenten die (abstrakten) Bedürfnisse für ihre Systeme und Dienste. Diese Bedürfnisse legen die Werte eines Systems fest und rechtfertigen weitere Anforderungen und ihre jeweiligen Kosten.

Beispiel

- Das System muss wartbar sein.
- Das System soll sicher sein.

Anforderungsebene („Was“)

Die abstrakten Geschäftsziele werden verfeinert, um die vorgesehenen Interaktionen zu beschreiben, d. h. welche Aktivitäten vom System unterstützt werden sollen, z. B. um die Wartung zu ermöglichen. Und welche Aktivitäten verhindert werden sollen, z. B. zum Schutz vor Angriffen. Die abstrakten Geschäftsziele ermöglichen darüber hinaus die Definition von messbaren Qualitätsmerkmalen des Systems. Diese Ziele können oft durch „Wie/Was“-Fragen verfeinert werden.

Beispiel

- Was macht das System wartbar?
  - Die Dokumentation muss in englischer Sprache erstellt werden.
  - Die Benutzerdokumentation muss in einfacher Sprache erstellt werden.
- Wie soll das System abgesichert werden?
  - Die Benutzerdaten müssen nach den Empfehlungen des BSI verschlüsselt werden.
  - Daten dürfen nur an autorisierte Akteure weitergegeben werden.

Wie spezifiziert man nicht-funktionale Anforderungen?

Da funktionale Anforderungen das beobachtbare Verhalten eines Systems beschreiben, können diese Anforderungen leicht getestet werden. Der vielfältige Bereich der nicht-funktionalen Anforderungen macht es schwierig, die Erfüllung solcher Anforderungen zu messen. Prozessanforderungen und Systembeschränkungen sind schwierig zu messen, Qualitätsanforderungen sind noch schwieriger zu messen, da der Begriff der Qualität sehr subjektiv ist und oft im Auge des Betrachters liegt.

Einige Tipps zur Definition messbarer nicht-funktionaler Anforderungen

1. Verfolgen Sie die Anforderungen von den abstrakten Geschäftsbedürfnissen bis hin zu den einzelnen Anforderungen; auf diese Weise lassen sich die Messungen über alle Abstraktionsebenen hinweg leichter definieren und durchführen. So könnte z.B. das Geschäftsziel „Das System muss sicher sein“ als Aggregation von daraus abgeleiteten nicht-funktionalen Anforderungen gemessen werden, wie z.B. „Die Benutzerdaten müssen entsprechend den Empfehlungen des BSI verschlüsselt werden“.
2. Verfeinern Sie die Anforderungen, bis sie umsetzbar und messbar sind. Die nichtfunktionale Anforderung „Die Nutzerdaten müssen entsprechend den Empfehlungen des BSI verschlüsselt werden“ muss z. B. hinsichtlich der Art der Nutzerdaten oder der Verschlüsselungsalgorithmen weiter präzisiert und verfeinert werden.

Weitere Details zur Anforderungsqualität werden wir in ID 1.1.1.8 - What makes a good requirement? behandeln.

[GERMAN] KEY TAKEAWAY

- Nicht-funktionale Anforderungen beziehen sich auf Bedürfnisse, die nicht durch funktionale Anforderungen abgedeckt werden.
- Wir unterscheiden drei Arten von nicht-funktionalen Anforderungen:
  - Qualitätsanforderungen
  - Prozessanforderungen
  - Systemeinschränkungen.
- Die Verfolgung und sorgfältige Verfeinerung der nicht-funktionalen Anforderungen von den Geschäftszielen bis zu den Implementierungstasks erleichtert die Messung.

ID 1.1.1.8 - What makes a good requirement?

Meta data

	EN	DE
Status	DRAFT	
Title	What makes a good requirement?	Was macht eine gute Anforderung aus?
Tags	• requirements quality	• Anforderungsqualität
Estimated Duration	10 min	
Level	☑ BEGINNER ☐ INTERMEDIATE ☐ EXPERT	
External ressources	• International standards for Systems and software Engineering • Requirements Engineering Quick Check • Requirements Smells	
Dependencies		
Next Item	ID 1.1.1.9 - Form of Requirements	
Goals (Teaching perspective)	Give a first impression on requirements quality (from a textual perspective)	
Content description (Teaching perspective)	Characteristics of a good requirement Characteristics of a good set of requirements Language recommendations for textual formulations of requirements	

ENGLISCH

[ENGLISCH] DESCRIPTION

In this learning nugget we will explore what makes a good requirement, refine a bad requirement into a good one, look at the set of requirements together.

[ENGLISCH] GOALS AND VALUE

In this learning nugget you will …

- Understand what characteristics the good requirement should have.
- What characteristics should the set of requirements have together.
- What language should be used to formulate a requirement.

[ENGLISCH] CONTENT

What makes a good requirement?

A good requirement is one that accurately captures the needs and expectations of stakeholders, is clear and unambiguous, and can be effectively implemented and verified. It should provide a solid foundation for the development process and facilitate communication between stakeholders and development teams. Further we will break down all characteristics of requirements.

Characteristics of "good" requirement

We will use the international standards for Systems and software Engineering to define the necessary properties of good requirements.

1. **Necessity**: The requirement should define an important characteristic that the system must have.
2. **Appropriateness**: The requirement should be specific to the entity and should not impose unnecessary constraints on the design.
3. **Unambiguity:** Each requirement must be clear and have a single interpretation.
4. **Completeness**: It should be able to explain the required functionality or constraint without requiring further detail to understand.
5. **Singularity**: It should define only one characteristic or constraint.
6. **Feasibility:** Each requirement should be achievable within the constraints of the project.
7. **Verifiability:** It should be possible to verify that a requirement has been met.
8. **Correctness:** It should accurately reflect the need of stakeholder.
9. **Conformity**: Each written requirement should follow the predefined structure.

It is important to note that not properties can be formally ensured; for instance, there is no universal approach to ensure that the requirements are correct, but we may be well able to discover through discussions that requirements are not correct, same as we may never be able to formally verify that the requirements are complet. It's therefore vital to take a pragmatic perspective here and ensure that each stakeholder agrees at least on the wording and essence of a requirement to ensure that it is consistent with all specified characteristics.

Let us look at the above properties in a concrete example and refine a bad requirement together.

Bad requirement: The system should be user-friendly.

1. **Necessity:** This requirement is too vague. It's unclear what "user-friendly" means and whether it's truly essential to the system. Let's refine it.

   Refined Requirement: The system must provide clear and intuitive navigation for users to perform common tasks efficiently.
2. **Appropriateness:** While the refined requirement is clearer, it may still be too broad and open to interpretation. Let's make it more specific.

   Refined Requirement: The system must include a menu bar with easily recognisable icons and labels for navigation, providing direct access to frequently used functions.
3. **Unambiguity:** The refined requirement is clearer, but it could still be interpreted differently by different stakeholders. Let's clarify it further.

   Refined Requirement: The system must display a fixed menu bar at the top of the interface, containing universally understood icons (e.g., home, search, settings) with accompanying text labels for each function.
4. **Completeness:** While the refined requirement explains the need for a menu bar, it doesn't address other aspects of usability, such as feedback mechanisms or error handling. Let's expand it.

   Refined Requirement: The system must include a fixed menu bar at the top of the interface, containing universally understood icons with accompanying text labels for each function. In addition, it should provide contextual tooltips for each icon on hover, providing guidance on their respective functions.
5. **Singularity:** The refined requirement covers several aspects of usability. Let's break it down into separate requirements.

   Refined Requirement (User Navigation): The system must include a fixed menu bar at the top of the interface, containing universally understood icons with accompanying text labels for each function.

New Requirement (Contextual Help): The system must provide contextual tooltips for each icon on hover, providing guidance on their respective functions.

6. **Feasibility:** The refined requirements seem feasible within the constraints of most projects. Let's leave them as they are.
7. **Verifiability:** It should be possible to verify that the system includes the specified menu bar with icons and labels, and that contextual tooltips are provided with simple tests during the development phase.
8. **Correctness:** The stakeholder has formally agreed to the refined requirements and we may therefore assume that it accurately reflect the stakeholders' need for a user-friendly interface. We consider it therefore to be (sufficiently) correct.
9. **Conformity:** Both refined requirements follow a predefined structure, specifying the desired feature and how it should be implemented.

What characteristics should the set of requirements have together?

When considering requirements, it is important to look at them as a whole rather than individually. Collections of requirements should have certain characteristics:

1. **Completeness**: Taken together, the requirements should cover all the necessary aspects of the system.
2. **Consistency**: They should not contradict each other.
3. **Feasibility**: The set of requirements should be achievable within the constraints of the project.
4. **Comprehensibility**. The set of requirements makes it clear how the system would work and what it would look like.
5. **Validation**: If the set of requirements is fulfilled, it should lead to the system that is expected.

What language should be used to formulate a requirement?

Requirements should be clear and focused on describing the necessary components, without going into technical details or specific solutions. They should state "what" is needed rather than "how" it should be achieved. Based on the Standards for Software and Systems Engineering we introduced the term Requirements Smells was developed. Similar to the “code smells”, it can be defined as an indicator of a quality violation in requirement representation, which may lead to a defect. Those are the main smells

- **Subjective Language:** Terms whose meaning cannot be objectively determined, such as "user-friendly", "easy to use" and "cost-effective".
- **Ambiguous Adverbs and Adjectives:** adjectives and adverbs that are by definition too general and not provide clarity, like “nearly always”, “considerable”, and “minimal”.
- **Loopholes:** Expressions stating that the following criteria must be met only in a certain vague degree, e.g. “if applicable”, “as far as practicable”.
- **Open-ended, Non-verifiable Terms:** terms that allow to have a variability in implementation, such as “*sufficient measurement”, “provide support”.*
- **Superlatives:** requirements that describe a quality of a system compared to all other systems, such as “the most secure” and “highest resolution*”.*
- **Comparatives:** terms that describe the quality of the system in relation to concrete systems or previous circumstances, like “higher than”, “better than”
- **Negative Statements:** declarations that specify what the system does not do, such as "The system must not sign off users due to timeouts" specify what the system does not do.
- **Vague Pronouns:** use of pronouns such as "this," "which," and so forth.
- **Incomplete References:** failure to provide accurate citations and references, which complicates searches.

💡 [ENGLISCH] KEY TAKEAWAY

- Effective requirements accurately capture stakeholder needs, adhere to international standards, and are clear, unambiguous and feasible within the project constraints, ensuring successful software development.
- A comprehensive set of requirements should be complete, consistent and feasible.

- The choice of language when formulating requirements should prioritise clarity, precision and objectivity.
- Tools such as fortiss' Requirements Engineering Quick Check can help benchmark and improve requirements processes.

GERMAN

[GERMAN] DESCRIPTION

In diesem Lernabschnitt untersuchen wir, was eine gute Anforderung ausmacht, wandeln eine schlechte Anforderung in eine gute um und beschäftigen uns mit der Gruppe der Anforderungen.

[GERMAN] GOALS AND VALUE

In diesem Lernmodul erfahren Sie ...

- Welche Eigenschaften eine gute Anforderung haben sollte.
- Welche Eigenschaften die Menge der Anforderungen zusammen haben sollte.
- Welche Sprache zur Formulierung einer Anforderung verwendet werden sollte.

[GERMAN] CONTENT

Was macht eine gute Anforderung aus?

Eine gute Anforderung ist eine Anforderung, die die Bedürfnisse und Erwartungen der Beteiligten genau erfasst, die klar und eindeutig ist und die effektiv implementiert und überprüft werden kann. Sie sollte eine solide Grundlage für den Entwicklungsprozess bilden und die Kommunikation zwischen den Beteiligten und den Entwicklungsteams erleichtern. Im Folgenden werden wir alle Merkmale von Anforderungen aufschlüsseln.

Merkmale einer „guten“ Anforderung

Wir werden die internationalen Standards für Systems und Software Engineering verwenden, um die notwendigen Eigenschaften guter Anforderungen zu definieren.

1. **Erforderlichkeit:** Die Anforderung sollte eine wichtige Eigenschaft definieren, die das System haben muss.
2. **Angemessenheit:** Die Anforderung sollte spezifisch für die Entität sein und dem Entwurf keine unnötigen Beschränkungen auferlegen.
3. **Eindeutigkeit:** Jede Anforderung muss eindeutig sein und eine einzige Interpretation zulassen.
4. **Vollständigkeit:** Sie sollte die geforderte Funktionalität oder Einschränkung erklären können, ohne dass weitere Details zum Verständnis erforderlich sind.
5. **Singularität:** Sie sollte nur ein Merkmal oder eine Einschränkung definieren.
6. **Durchführbarkeit:** Jede Anforderung sollte im Rahmen der Projektvorgaben realisierbar sein.
7. **Überprüfbarkeit:** Es sollte möglich sein zu überprüfen, ob eine Anforderung erfüllt wurde.
8. **Korrektheit:** Sie sollte die Bedürfnisse der Beteiligten genau widerspiegeln.
9. **Konformität:** Jede schriftliche Anforderung sollte der vordefinierten Struktur folgen.

Es ist von entscheidender Bedeutung, dass sich alle Stakeholder über den Wortlaut und das Wesen einer Anforderung einig sind, um sicherzustellen, dass sie mit allen festgelegten Merkmalen übereinstimmt.

Schauen wir uns diese Eigenschaften an einem konkreten Beispiel an und verfeinern wir gemeinsam eine schlechte Anforderung.

Schlechte Voraussetzung: Das System sollte benutzerfreundlich sein.

1. **Erforderlichkeit:** Diese Anforderung ist zu vage. Es ist unklar, was „benutzerfreundlich“ bedeutet und ob es wirklich wesentlich für das System ist. Wir sollten sie verfeinern.

Verfeinerte Anforderung: Das System muss den Benutzern eine klare und intuitive Navigation bieten, damit sie gängige Aufgaben effizient ausführen können.

2. **Angemessenheit:** Die verfeinerte Anforderung ist zwar klarer, aber möglicherweise immer noch zu weit gefasst und offen für Interpretationen. Wir sollten sie genauer formulieren.

   Verfeinerte Anforderung: Das System muss eine Menüleiste mit leicht erkennbaren Symbolen und Bezeichnungen für die Navigation enthalten, die einen direkten Zugriff auf häufig verwendete Funktionen ermöglicht.

3. **Eindeutigkeit:** Die verfeinerte Anforderung ist klarer, könnte aber immer noch von verschiedenen Beteiligten unterschiedlich interpretiert werden. Lassen Sie sie uns weiter verdeutlichen.

   Verfeinerte Anforderung: Das System muss eine feste Menüleiste am oberen Rand der Benutzeroberfläche anzeigen, die allgemein verständliche Symbole (z. B. Home, Suche, Einstellungen) mit begleitenden Textbeschriftungen für jede Funktion enthält.

4. **Vollständigkeit:** Die verfeinerte Anforderung erklärt zwar die Notwendigkeit einer Menüleiste, geht aber nicht auf andere Aspekte der Benutzerfreundlichkeit ein, wie etwa Feedback-Mechanismen oder Fehlerbehandlung. Erweitern wir sie.

   Verfeinerte Anforderung: Das System muss eine feste Menüleiste am oberen Rand der Benutzeroberfläche enthalten, die allgemein verständliche Symbole mit begleitenden Textbeschriftungen für jede Funktion enthält. Darüber hinaus sollte es kontextbezogene Tooltipps für jedes Icon bereitstellen, wenn man den Mauszeiger darüber bewegt, um die jeweiligen Funktionen zu erläutern.

5. **Einzigartigkeit:** Die verfeinerte Anforderung umfasst mehrere Aspekte der Benutzerfreundlichkeit. Lassen Sie uns diese in einzelne Anforderungen aufteilen.

   Verfeinerte Anforderung (Benutzernavigation): Das System muss eine feste Menüleiste am oberen Rand der Benutzeroberfläche enthalten, die allgemein verständliche Symbole mit begleitenden Textbeschriftungen für jede Funktion enthält.

   Neue Anforderung (kontextbezogene Hilfe): Das System muss kontextbezogene Tooltipps für jedes Icon bereitstellen, wenn der Mauszeiger über das Icon bewegt wird, die Hinweise zu den jeweiligen Funktionen geben.

6. **Durchführbarkeit:** Die verfeinerten Anforderungen scheinen im Rahmen der meisten Projekte machbar zu sein. Wir sollten sie so belassen, wie sie sind.
7. **Überprüfbarkeit:** Es sollte möglich sein, während der Entwicklungsphase mit einfachen Tests zu überprüfen, ob das System die angegebene Menüleiste mit Symbolen und Beschriftungen enthält und ob kontextbezogene Tooltipps bereitgestellt werden.
8. **Korrektheit:** Die verfeinerten Anforderungen spiegeln den Bedarf der Beteiligten an einer benutzerfreundlichen Schnittstelle korrekt wider.
9. **Konformität:** Beide verfeinerten Anforderungen folgen einer vordefinierten Struktur, in der die gewünschte Funktion und die Art und Weise, wie sie implementiert werden soll, festgelegt sind.

Welche Merkmale sollten die Anforderungen insgesamt aufweisen?

Bei der Betrachtung von Anforderungen ist es wichtig, sie in ihrer Gesamtheit und nicht einzeln zu betrachten. Sammlungen von Anforderungen sollten bestimmte Merkmale aufweisen:

1. **Vollständigkeit:** In ihrer Gesamtheit sollten die Anforderungen alle notwendigen Aspekte des Systems abdecken.
2. **Konsistenz:** Sie sollten sich nicht gegenseitig widersprechen.
3. **Durchführbarkeit:** Der Anforderungskatalog sollte im Rahmen des Projekts realisierbar sein.
4. **Nachvollziehbarkeit**. Aus den Anforderungen geht klar hervor, wie das System funktioniert und wie es aussieht.
5. **Validierung**: Wenn das Anforderungspaket erfüllt ist, sollte es zu dem System führen, das erwartet wird.

In welcher Sprache sollte eine Anforderung formuliert werden?

Anforderungen sollten klar sein und sich auf die Beschreibung der notwendigen Komponenten konzentrieren, ohne auf technische Details oder spezifische Lösungen einzugehen. Sie sollten angeben, „was“ benötigt wird und nicht „wie“ es erreicht werden soll. In Anlehnung an die Standards for Software and Systems Engineering wurde der Begriff Requirements Smells entwickelt. Ähnlich wie bei den „Code Smells“ kann er als Indikator für eine Qualitätsverletzung in der Anforderungsdarstellung definiert werden, die zu einem Fehler führen kann. Dies sind die wichtigsten Smells:

- **Subjektive Sprache:** Begriffe, deren Bedeutung nicht objektiv bestimmt werden kann, wie z. B. „benutzerfreundlich“, „leicht zu bedienen“ und „kostengünstig“.
- **Mehrdeutige Adverbien und Adjektive:** Adjektive und Adverbien, die per definitionem zu allgemein sind und keine Klarheit schaffen, wie „fast immer“, „erheblich“ und „minimal“.
- **Schlupflöcher:** Ausdrücke, die besagen, dass die folgenden Kriterien nur in einem bestimmten vagen Ausmaß erfüllt werden müssen, z. B. „falls zutreffend“, „soweit durchführbar“.
- **Offene, nicht überprüfbare Begriffe:** Begriffe, die eine Variabilität in der Umsetzung zulassen, z. B. „ausreichende Messung“, „Unterstützung bieten“.
- **Superlative**: Anforderungen, die die Qualität eines Systems im Vergleich zu allen anderen Systemen beschreiben, z. B. „das sicherste“ und „höchste Auflösung“.
- **Komparative:** Begriffe, die die Qualität des Systems in Bezug auf konkrete Systeme oder frühere Umstände beschreiben, wie „höher als“, „besser als“
- **Negative Aussagen:** Erklärungen, die angeben, was das System nicht tut, wie z. B. „Das System darf Benutzer nicht aufgrund von Zeitüberschreitungen abmelden“.
- **Vage Pronomen:** Verwendung von Pronomen wie „dies“, „welches“ usw.
- **Unvollständige Verweise:** Fehlen von genauen Zitaten und Verweisen, was die Suche erschwert.

💡 [GERMAN] KEY TAKEAWAY

- Wirksame Anforderungen erfassen die Bedürfnisse der Beteiligten genau, entsprechen internationalen Standards, sind klar, eindeutig und im Rahmen der Projektvorgaben realisierbar und gewährleisten so eine erfolgreiche Softwareentwicklung.
- Ein umfassender Anforderungssatz sollte vollständig, konsistent und realisierbar sein.
- Bei der Wahl der Sprache für die Formulierung von Anforderungen sollten Klarheit, Präzision und Objektivität im Vordergrund stehen.
- Tools wie der fortiss' Requirements Engineering Quick Check können helfen, Anforderungsprozesse zu bewerten und zu verbessern.

ID 1.1.1.9 - Form of Requirements

Meta data

	EN	DE
Status	DRAFT	
Title	Form of Requirements	Requirementsformen
Tags	• natural language • structured language • model-base requirements	• Natürliche Sprache • Strukturierte Sprache • Model-basierte Anforderungen
Estimated Duration	5 min	
Level	☑ BEGINNER ☐ INTERMEDIATE ☐ EXPERT	
External ressources		
Dependencies		
Next Item	ID 1.1.1.10 - Wrapup	
Goals (Teaching perspective)	To understand that there a varying forms of requirements, and each serve a specific purpose	
Content description (Teaching perspective)	• Natural language requirements • Structured language requirements • Model-based requirements • Advantages and disadvantages of each form • Examples for each form	

ENGLISCH

[ENGLISCH] DESCRIPTION

In this learning nugget we will explore different ways of describing requirements: natural language, structured language, and model-based requirements.

🎯 [ENGLISCH] GOALS AND VALUE

In this learning nugget you will …

- Get to know the main three types of requirements forms.
- Understand the advantages and disadvantages of each requirements form.

[ENGLISCH] CONTENT

Goals, needs, and constraints can be expressed in various forms. In this part, we'll look into the three most common ways to express them: natural language requirements, structured language requirements, and model-based requirements.

Natural Language Requirements

All examples you have seen so far use natural language to convey the essence of the requirement. This is traditionally the most commonly used format to communicate requirements. The simple reason is that all stakeholders understand natural language. The main advantage of textual requirements is therefore the accessibility. Every stakeholder can read, understand and contribute to the requirements (given enough context knowledge). This renders natural language requirements the easiest employable technique to convey requirements between stakeholders of diverse backgrounds (e.g. business analysts and developers).

Example

- The system must provide a search function that allows users to find products by name or category.
- Users must be able to export their data reports in CSV format directly from their dashboard.

The main disadvantage of natural language requirements is ambiguity. Natural language is often subject to interpretation, especially when stakeholders have different backgrounds. Ambiguities can severly impact the project success. The earlier ambiguities are eliminated, the less sever the impact.

Example

- *Authorized users should be able to access advanced features depending on their preferences.*
  - This requirement does not specify what "advanced features" are, leaving it's interpretation to the respective stakeholder.
- *The system should respond quickly to user input.*
  - This requirements does not specify what "quickly" means, again leaving it to the stakeholders interpretation.

Structured Language Requirements

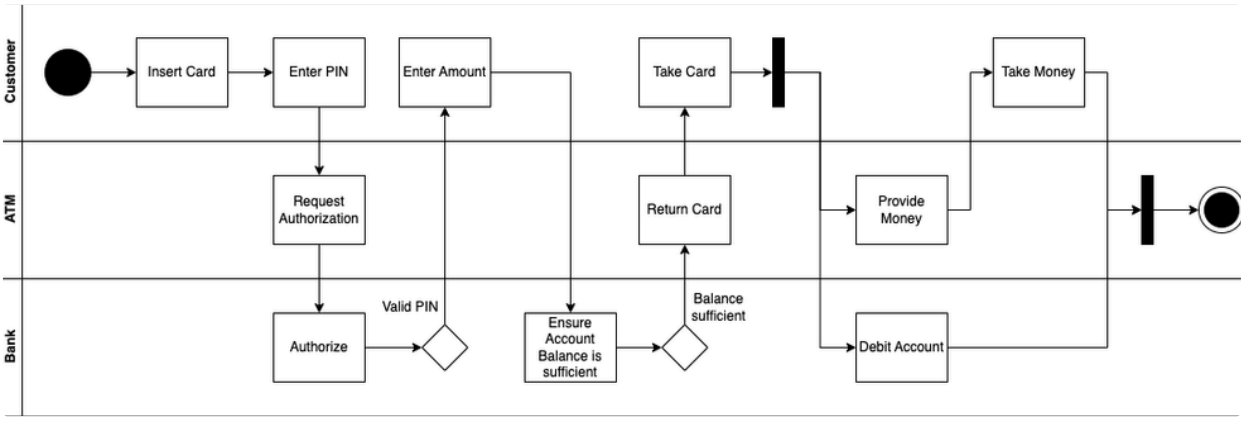
Structured language requirement are one way to tackle the issues of ambiguity in natural language requirements at least to some extent. Requirements are structured in templates prescribing the content. Structured language can for example be used to unambiguously convey an order between activities, as for example with a use case template (see example below). At the same time, this strict form can hinder creative requirements elicitation, is more difficult to apply to abstract requirements (e.g. at the business goals), and requires some practice to be effective. In essence, while structured language requirements maybe be a bit cumbersome to write down and read (often resembling the user-friendly charme of an instruction manual to a tax declaration form), they offer the benefit of clear structure and some basic quality criteria that make them easier to interpret (implement, and test).

USE Case #	**<the name is the goal as a short active verb phrase>**
Context of Use	<a longer statement of the context of use if needed>
Scope	<what system is being considered black box under design>
Level	<one of summary, primary taks, subfunction>
Primary Actor	<a role name for the primary actor, or a description>
Stakeholder and Interests	Stakeholder Interest <stakeholder name> <put here the interest of the stakeholder>

	<stakeholder name>	<put here the interest of the stakeholder>
Preconditions	<what we expect is already the state of the world>	
Minimal Guarantees	<the interests as protected on any exit>	
Success Guarantees	<the interests as satisfied on a successful ending>	
Trigger	<the action upon the system that starts the use case>	
Description	Step	Action
	1	<put here the steps of the scenario from trigger to goal delivery and any cleanup after>
	2	<…>
Extensions	Step	Branching Action
	1a	<condition causing branching>: <action or name of sub use case>
Technology and Data Variations	1	<list of variations>

Model-based Requirements

Another common form of documenting requirements is through models, with the most adopted ones being UML and BPMN. We may note here that other far more formalised and mathematically oriented models are in use as well, depending on the domain, but focus on the above notations given their wide adoption. Depending on the type, models can cover various concepts, from stakeholders over system boundaries to activity sequence flows. The main advantage of models is their clarity, at least from an engineering perspective. They are build with one goal in mind (e.g. to describe the activity flow) and unambiguous express the related concepts. However, models come with a few disadvantages. First, learning to read and create model comes with a steep learning curve, often excluding crucial stakeholders with no modeling knowledge from the discussion (such as end users). Seconds, models are build on modeling languages and their concepts, which are limited to certain goals. No one model language can cover all required concepts; that is, no modelling approach fits all purposes. Hence, their application is context dependent and requires individual analysis for each project's needs.



Scenario of succesful Withdrawal at ATM as UML activity diagram

[ENGLISCH] KEY TAKEAWAY

- The three most common forms of requirements are natural language, structured language, and models
- Natural language requirements are easily utilized across all stakeholders at the cost of ambiguity
- Structured language requirements prescribe the content and forms of a requirement, impacting flexibility in requirements specification
- Model-based requirements are precise, but need knowledge about the modeling language excluding many stakeholders
- Each requirements form has its one advantages and disadvantages and their usage depends on the situation

GERMAN

[GERMAN] DESCRIPTION

In diesem Lernabschnitt werden wir verschiedenen Arten erforschen, Anforderungen zu beschreiben: Natürliche Sprache, strukturierte Sprache, und modellbasierte Anforderungen.

[GERMAN] GOALS AND VALUE

In diesem Lernmodul erfahren Sie ...

- Mit welchen drei Hauptformen Anforderungen beschrieben werden.
- Was deren Vor- und Nachteile sind.

[GERMAN] CONTENT

Ziele, Bedarfe und Einschränkungen können in verschiedensten Formen repräsentiert werden. In dieser Lerneinheit fokussieren wir uns auf die am häufigsten vorkommenden Formen: Anforderungen spezifiert in natürlicher Sprache, in strukturierte Sprache und Anforderungen basierend auf Modellen.

Anforderungen spezifiziert in natürlicher Sprache

Alle Beispiele, die wir bisher gesehen haben, nutzen natürliche Sprache, um den Kernpunkte der Anforderungen zu vermitteln. Diese Form ist traditionell die am häufigsten verwendete Art, um Anforderungen zu kommunizieren. Der Hauptvorteil textueller Anforderungen ist die einfache Verständlichkeit. Jeder involvierte Stakeholder kann diese Art von Anforderung lesen, versehen und verbessern (unter der Voraussetzung von Domänenwissen). Das macht Anforderungen in natürlicher Sprache zu der einfachsten Technik, um Anforderungen zwischen verschiedenen Stakeholdern mit diversen Hintergründen (z.B. Business Analysts und Entwickler) zu kommunizieren.

Beispiel

- Das System stellt eine Suchfunktion bereit, die es Nutzenden ermöglicht, Produkte über den Namen oder die Kategorie zu finden.
- Nutzende können ihren Datenreport im CSV Format direkt aus dem Dashboard exportieren.

Der größte Nachteil dieser Art vom Anforderungen ist ihre Unklarheit beziehungsweise Mehrdeutigkeit. Anforderungen, die in natürlicher Sprache spezifiziert sind, sind häufig anfällig für subjektive Interpretationen, besonders wenn Stakeholder unterschiedliche Hintergründe haben. Diese Mehrdeutigkeit kann den Projekterfolg erheblich beeinflussen. Je früher im Projektprozess Unklarheiten und Mehrdeutigkeiten gelöst werden, desto geringer der Einfluss auf das gesamte Projekt.

Beispiel

- *Authorisierte Nutzende haben die Möglichkeit, erweiterte Funktionen basierend auf ihren Präferenzen zu nutzen.*
  - Diese Anforderung spezifiert nicht, was diese “erweitere Funktionen“ sind, und überlässt deren Interpretation dem jeweiligen Stakeholder.
- *Das System soll schnell auf Nutzereingaben reagieren.*
  - Die Anforderung spezifiert nicht, was “schnell“ bedeutet, und überlässt dies dem jeweiligen Stakeholder.

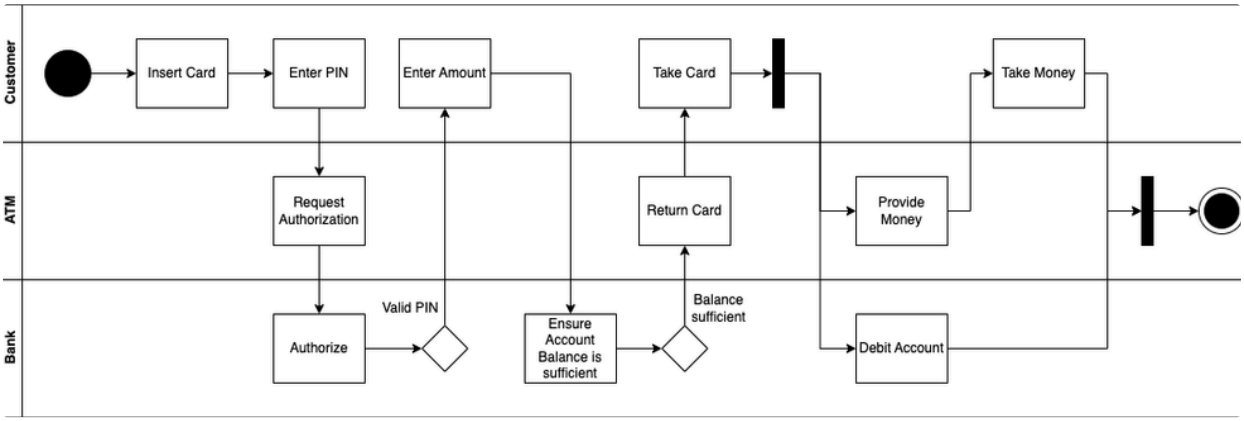
Anforderungen spezifiert in strukturierter Sprache

Anforderungen, die in strukturierter Sprache spezifiziert werden, sind eine Art um das Problem der Uneindeutigkeit der natürlichen Sprache zu verbessern. Solche Anforderungen sind nach Vorlagen strukturiert, die den Inhalt vorgeben. Durch die Struktur können Teile der Anforderungen eindeutig vorgegeben und kommuniziert werden. Ein Beispiel ist die Reihenfolge von Aktivititäten in einer Use Case-Vorlage (siehe Beispiel unten). Gleichzeitig hat diese strikte Form negative Konsequenzen für eine schnelle und kreative Anforderungserhebung. Zusätzlich ist es schwierig, abstrakte Anforderungen wie Unternehmensziele darin zu kommunizieren. Und schließlich benötigt diese Form der Anforderungsspezifizierung einiges an Übung für eine effektiven Erstellung und Nutzung.

USE Case #	**<Der Name ist das Ziel als kurzen Satz mit aktivem Verb>**
Kontext des Nutzen	<Ein längeres Statement zum Kontext des Nutzen, falls nötig>
Umfang	<Welches System wird als Blackbox im Entwurf betrachtet>
Level	<Eines von: Zusammenfassung, Primäraufgabe, Unterfunktion>
Hauptakteur	<Ein Rollenname für den Hauptnutzenden, oder eine Beschreibung>
Stakeholder und Ihre Interessen	Stakeholder — Interesse <Stakeholder Name> — <Interesse des Stakeholders> <Stakeholder Name> — <Interesse des Stakeholders>
Voraussetzungen	<Was wir als gegeben über die Welt annehmen>
Minimale Zusicherungen	<Die bei jedem Ergebnis geschützten Interessen>
Erfolgszusicherungen	<Die Interessen, die bei erfolgreichem Ergebnis befriedigt werden>
Trigger	<Die Aktion auf dem System, die den Anwendungsfall startet>
Beschreibung	Schritt — Aktion 1 — <die Schritte des Szenarios vom Auslöser bis zur Übergabe des Ziels und eventuelle Bereinigung> 2 — <…>
Erweiterungen	Schritt — Abzweigungsaktion 1a — <Bedingung für die Abzweigung>: <Aktion oder Name des untergeordneten Use Cases>
Technologien and Daten Variationen	1 — <Liste von Variationen>

Modellbasierte Anforderungen

Eine weitere häufig vorkommende Form der Dokumentation von Anforderungen ist durch Modelle. Die am weitest verbreiteten Modellierungssprachen hierfür sind UML und BPMN. Je nach Art können Modelle verschiedene Konzepte abdecken, von Stakeholdern über Systemgrenzen bis hin zu Aktivitätsabläufen. Der Hauptvorteil von Modellen ist ihre Klarheit. Sie werden mit einem Ziel vor Augen erstellt (z.B. zur Beschreibung eines Aktivitätsflusses) und drücken die damit verbunden Konzepte eindeutig aus. Allerdings haben Modelle auch einige Nachteile. Erstens ist das Erlernen des Lesens und Erstellen von Modellen mit einer steilen Lernkuve verbunden, die Stakeholder ohne Modellierungskenntnisse aus Diskussionen ausschließt. Zweitens beruhen Modelle auf Modellierungssprachen und deren Konzepten, die auf bestimmte Ziele beschränkt sind. Keine Modellierungssprache kann alle erfoderlichen Konzepte abecken. Daher ist ihre Anwendung kontextabhängig und erfordert eine individuelle Analyse der Bedürfnisse jedes Projekts.



Szenario erfolgreichen Geldabhebens am Bankautomat als UML Aktivitätsdiagram

[GERMAN] KEY TAKEAWAY

- Die drei häufigsten Formen von Anforderungen sind natürliche Sprache, strukturierte Sprache und Modelle.
- Anforderungen in natürlicher Sprache können leicht von allen Beteiligten verwendet werden, allerdings auf Kosten der Eindeutigkeit.
- Anforderungen in strukturierter Sprache schreiben den Inhalt und die Form einer Anforderung vor, was die Flexibilität bei der Anforderungserhebung beeinträchtigt.
- Modellbasierte Anforderungen sind präzise, erfordern aber Kenntnisse über die Modellierungssprache, was viele Stakeholder ausschließt.
- Jede Anforderungsform hat ihre eigenen Vor- und Nachteile und ihre Verwendung hängt von der jeweiligen Situation ab.

ID 1.1.1.10 - Wrapup

Meta data

	EN	DE
Status	DRAFT	
Title	Summarise Requirements Engineering Introduction	Zusammenfassung der Einführung in das Requirements Engineering
Tags		
Estimated Duration	5 min	
Level	☑ BEGINNER ☐ INTERMEDIATE ☐ EXPERT	
External ressources		
Dependencies		
Next Item		
Goals (Teaching perspective)	To summarise all knowledge described in Learning Unit 1.1.1.	
Content description (Teaching perspective)		

ENGLISCH

[ENGLISCH] DESCRIPTION

In this learning nugget we will summarise the knowledge that we have learned in this chapter.

[ENGLISCH] GOALS AND VALUE

In this Learning Nugget you will...

- Understand the importance of requirements engineering in software development projects.
- Explore the importance of prioritising problem solving over solution identification in requirements engineering.
- Recognise the different types and forms of requirements.

[ENGLISCH] CONTENT

In the previous chapters, we discussed the fundamental concepts of Requirements Engineering and provided an overview of this field. This chapter aims to summarize the knowledge we have gathered thus far.

What are Requirements Engineering and Requirements?

Requirements Engineering is the structured approach to developing clear and comprehensive requirements for software development projects. A requirement can be a stakeholder's need or a constraint imposed upon the software development project. It can also define what a system should be capable of doing or what properties it should possess.

Requirements Engineering involves selecting, analysing, documenting, and verifying stakeholders' needs and system properties. Requirements serve as the foundation for communication, facilitate understanding, and ensure that the system under development functions as intended. A successful Requirements Engineering process can significantly impact a project's success, affecting system functionality, quality, cost, benefits, and complexity.

How does the Requirements Engineering process look in practice?

It is commonly believed that smaller projects or those using agile methodologies such as Scrum do not engage in Requirements Engineering. However, this is clearly a misconception. Even organisations that follow agile principles still prioritise and track requirements as user stories or product backlog items, even if this process is not explicitly defined as requirements engineering activities. In cases where projects consist of one or two people, system requirements may be written on paper or a post-it note, and decisions may be made in conversation; however, they are still part of requirements engineering.

To improve the quality of the requirements engineering process and the resulting product quality, organizations must recognize that each project requires a unique approach that involves all stakeholders in discussions and continues throughout the project, being able to adapt to upcoming changes and track them from the requirement to implementation and verification. Requirements may change during the project and it is important to be able to adapt to these changes.

During the Requirements Engineering journey, it is important to prioritize the question of "what needs to be done" over "how it should be done." By focusing on the problem first and identifying solutions second, organizations can ensure that they meet the requirements of stakeholders, consider all project limitations and scope, and choose the best possible solution. This approach helps to streamline the process and achieve the desired outcomes more efficiently.

Types and forms of requirements

During product development, it is essential to document various types of requirements to guide the project efficiently. These include

- **Goals and contstraints:** These define the high-level goals and expected benefits from a business perspective, providing context for the project objectives.
- **Functional requirements:** These detail specific actions or behaviours expected of the system during development, and act as a blueprint for functionality.
- **Non-functional requirements:** These emphasise quality attributes and process rules beyond functionality, addressing critical issues such as performance and security.

Requirements can take various forms, each of which serves a specific purpose and has its advantages and disadvantages:

- **Natural language:** These requirements are expressed in everyday language, ensuring accessibility to stakeholders with different backgrounds. While they are easy to understand, they are susceptible to ambiguity due to potential differences in interpretation.
- **Structured language:** Requirements are organised using predefined templates to minimise ambiguity and provide clear guidance. This method dictates specific formats and content, which improves clarity but may limit flexibility.
- **Model-based:** Requirements are captured using graphical models such as UML (Unified Modelling Language) or BPMN (Business Process Model and Notation). This approach offers precision and clarity through the use of graphical or formal modelling languages, but requires expertise in interpreting and creating such models.

Product development projects depend on how clear and effective their requirements are. Each requirement must follow specific rules to ensure its relevance, measurability and feasibility. In addition, all requirements must form a coherent and comprehensive set to facilitate the development of the project. Teams can ensure transparency, alignment and successful project outcomes by emphasising compliance with both individual requirement standards and the integrity of the overall requirement set.

[ENGLISCH] KEY TAKEAWAY

- Requirements engineering is essential to successful software development projects.
- Prioritising 'what to do' over 'how to do it' streamlines the process and ensures effective problem solving.
- Different types and forms of requirements, including business, functional and non-functional, need to be carefully documented and managed.
- Requirements can take different forms, such as natural language, structured formats or model-based approaches, each with its own advantages and disadvantages.
- Adherence to requirements standards and maintaining the integrity of the overall set of requirements is critical to project transparency, alignment and success.

GERMAN

[GERMAN] DESCRIPTION

In diesem Lernabschnitt werden wir das Wissen zusammenfassen, das wir in diesem Kapitel gelernt haben.

[GERMAN] GOALS AND VALUE

In diesem Lernmodul erfahren Sie ...

- Welche Relevanz Requirements Engineering in Softwareentwicklungsprojekten hat.
- Was der Vorrang der Problemlösung vor der Lösungsfindung bei der Anforderungserhebung bedeutet.
- Wie Sie verschiedenen Arten und Formen von Anforderungen erkennen.

[GERMAN] CONTENT

In den vorangegangenen Lernmodulen haben wir die grundlegenden Konzepte des Requirements Engineering erörtert und einen Überblick über dieses Gebiet gegeben. Diese Lerneinheit soll das bisher gesammelte Wissen zusammenfassen.

Was ist Requirement Engineering und was sind Anforderungen?

Requirements Engineering ist ein methodischer Ansatz zur Entwicklung klarer und umfassender Anforderungen für Softwareentwicklungsprojekte. Eine Anforderung kann ein Bedürfnis eines Interessenvertreters sein oder eine Einschränkung, die dem Softwareentwicklungsprojekt auferlegt wird. Sie kann auch definieren, was ein System können soll oder welche Eigenschaften es haben soll.

Requirements Engineering umfasst die Erhebung, Auswahl, Analyse, Dokumentation und Verifizierung der Bedarfe der Beteiligten und der Systemeigenschaften. Anforderungen dienen als Grundlage für die Kommunikation, erleichtern das Verständnis und stellen sicher, dass das zu entwickelnde System wie vorgesehen funktioniert. Ein erfolgreicher Requirements-Engineering-Prozess kann den Erfolg eines Projekts erheblich beeinflussen, da er sich auf die Funktionalität, die Qualität, die Kosten, den Nutzen und die Komplexität des Systems auswirkt.

Anforderungstypen und Anforderungsformen

Während der Produktentwicklung ist es wichtig, verschiedene Arten von Anforderungen zu dokumentieren, um das Projekt effizient zu steuern. Dazu gehören

- **Geschäftsziele und Einschränkungen:** Sie definieren die übergeordneten Ziele und den erwarteten Nutzen aus geschäftlicher Sicht und liefern den Kontext für die Projektziele.
- **Funktionale Anforderungen:** Sie beschreiben spezifische Aktionen oder Verhaltensweisen, die vom System während der Entwicklung erwartet werden, und dienen als Entwurf für die Funktionalität.
- **Nicht-funktionale Anforderungen:** Diese betonen Qualitätsattribute und Prozessregeln, die über die Funktionalität hinausgehen, und befassen sich mit kritischen Themen wie Leistung und Sicherheit.

Anforderungen können verschiedene Formen annehmen, von denen jede einen bestimmten Zweck erfüllt und ihre Vor- und Nachteile hat:

- **Natürliche Sprache:** Diese Anforderungen werden in der Alltagssprache formuliert, so dass sie für Beteiligte mit unterschiedlichem Hintergrund zugänglich sind. Sie sind zwar leicht verständlich, aber aufgrund möglicher unterschiedlicher Auslegungen anfällig für Mehrdeutigkeiten.
- **Strukturierte Sprache:** Die Anforderungen werden anhand von vordefinierten Vorlagen organisiert, um Mehrdeutigkeiten zu minimieren und eine klare Orientierung zu bieten. Diese Methode schreibt bestimmte Formate und Inhalte vor, was die Klarheit verbessert, aber die Flexibilität einschränken kann
- **Modellbasiert Anforderungen:** Die Anforderungen werden mithilfe grafischer Modelle wie UML (Unified Modelling Language) oder BPMN (Business Process Model and Notation) erfasst. Dieser Ansatz bietet Präzision und Klarheit durch die Verwendung von grafischen oder formalen Modellierungssprachen, erfordert jedoch Fachkenntnisse in der Interaktion mit Modellen.

Produktentwicklungsprojekte hängen davon ab, wie klar und effektiv ihre Anforderungen sind. Jede Anforderung muss bestimmten Regeln folgen, um ihre Relevanz, Messbarkeit und Durchführbarkeit zu gewährleisten. Darüber hinaus müssen alle Anforderungen eine kohärente und umfassende Gruppe bilden, um die Entwicklung des Projekts zu erleichtern. Teams können für Transparenz, Abstimmung und erfolgreiche Projektergebnisse sorgen, indem sie auf die Einhaltung sowohl einzelner Anforderungsstandards als auch auf die Integrität des gesamten Anforderungspakets achten.

💡 [GERMAN] KEY TAKEAWAY

- Requirements Engineering ist essentiell für erfolgreiche Softwareentwicklungsprojekte.
- Das Priorisieren des “was“ über das “wie“ optimiert den Prozess und gewährleistet effektive Problemlösung.
- Verschiedenen Typen und Formen von Anforderungen wie Geschäftsziele, funktionale oder nicht-funktionale Anforderungen müssen sorgfältig dokumentiert und verwaltet werden.
- Anforderungen können in verschiedensten Formen wie natürlicher Sprache, strukturierter Sprache oder Modellbasierten Ansätzen vorkommen, und jede hat ihre Vor- und Nachteile.
- Die Einhaltung von Anforderungsstandards und die Wahrung der Integrität des gesamten Anforderungspakets sind entscheidend für die Transparenz, die Kommunikation und den Erfolg des Projekts.

ID 1.1.2.1 Overview of phases and core activites

Meta data

	EN	DE
Status	READY FOR REVIEW	
Title	Overview of phases	Titel
Tags	• Tag 1 • Tag 2	• Tag 1 • Tag 2
Estimated Duration	5 min	
Level	☑ BEGINNER ☐ INTERMEDIATE ☐ EXPERT	
External ressources	• Ressource 1 • Ressource 2	• Ressource 1 • Ressource 2
Dependencies		
Next Item	ID 1.1.2.2 - Requirements Elicitation	
Goals (Teaching perspective)	• Get to know the five phases in requirements engineering • Understand the relation between the phases	

Content description (Teaching perspective)	• Describes the five phases in requirements engineering
	  ◦ Requirements Elicitation
	  ◦ Requirements Analysis
	  ◦ Requirements Specification
	  ◦ Requirements Verification and Validation
	  ◦ Requirements Management
	• Gives first insights into the core activities in each phases
	• Highlights the relation between the phases	

Content summary (Teaching perspective)		

Content data

EN DE

DESCRIPTION

In this learning nugget we will explore the five phases in requirements engineering, what they are composed of, and how they relate to each other to bridge the gap between problem space and solution space.

Die Lerneinheit werden wir …

🎯 GOALS AND VALUE (User perspective)

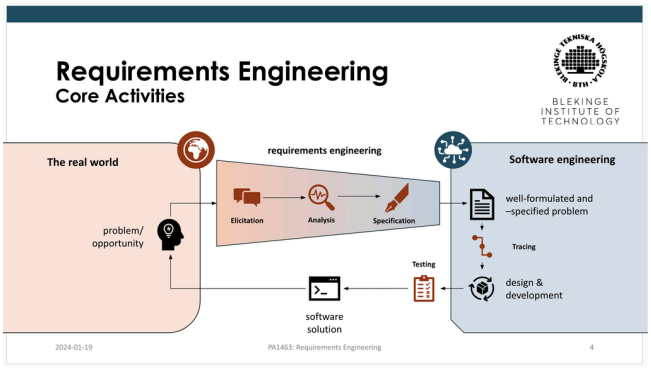
In this learning nugget you will …

- get to know the five typical phases to Requirements Engineering
- understand the goals and core activities of each phase
- identify the relations between the phases

In dieser Lerneinheit werden Sie …

- Ziel 1
- Ziel 2
- Ziel 3

CONTENT



Just for illustration purposes, to be changed against something fitting better the narrative

Die Kakerlake

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein schreckliches Ungeziefer verwandelt. ...

Die Bedeutung

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein furchtbares Ungeziefer verwandelt. Er lag auf seinem panzerartigen Rücken, ...

As we have seen in ID 1.1.1.2 - What is requirements engineering? requirements engineering is the systematic, iterative, and disciplined approach to develop explicit requirements and system specification that all stakeholders agree upon.

As such, requirements engineering can have different forms and interpretations. However, independent of the form or interpretation, requirements engineering typically spans five phases, regardless of whether these are made explicit or interwoven: requirements elicitation, requirements analysis, requirements specification, requirements verification and validation, and requirements management.

Phases

The four phases guide through the engineering of requirement to bridging the gap between the problem and the solution space.

- Requirements elicitation – identifying all relevant stakeholders and their requirements.

- Requirements analysis – understanding requirements and achieving consensus among stakeholders.
- Requirements specification – structuring, modelling, documenting requirements.
- Requirements validation and verification – ensuring validity and quality of created requirements.
- Requirements management – documenting, tracking, prioritizing, and controlling changes to requirements.

Each of those phases processes requirements in a different form and outputs requirements in a more detailed or enriched form for the next phase. For that, we perform different core activities in each phase (following J. Doerr's Reqman Process Framework):

Requirements Elicitation

During requirements elicitation, we **identify sources and stakeholders** that are to be involved in the process of gathering needs and constraints. We **define the scope** of the the problem and the solution, and consequently **define the goals** of the solution. If necessary, we **analyze the business processes** and based on all those steps, we finally **elicit the requirements.**

Further information on Requirements Elicitation

Requirements Analysis

In requirements analysis, we **refine** the elicited requirements and **structure** them, so we can properly **model the requirements**, which serves as a basis to **explain, decide, and evaluate the requirements**, among others we **evaluate the requirements for conflicts and feasibility.**

Further information on Requirements Analysis

Requirements Specification

Requirements specification is all about bringing the analyzed requirements into a common form by **documenting and structuring** them further. Here, we also ensure the **traceability and the decidability of the requirements.**

Further information on Requirements Specification

Requirements Verification and Validation

Verification and validation of requirements has the primary goals of ensuring that the requirements correctly represent the needs of stakeholders. With validation, we refer to whether we are building the right system (i.e. are the requirements correct?) and with verification, we refer to whether we are building the system right (i.e. does the system satisfy the requirements?). In that sense, requirements verification is typically captured by (acceptance) testing. With verification and validation, overall, we evaluate

requirements against stakeholder expectations. In that sense, we sometimes also **prototype** and showcase to the stakholders, perform **formal verification** where applicable, **review the specified requirements**, or already **specify test cases** for verification.

Further information on Requirements Verification and Validation

Requirements Management

Requirements management acts as the overarching activity stretching over the other four activities. In requirements management we mainly **track and trace** and **control** changes to requirements. This can be supported by integrating **reuse** of artefacts (e.g. requirements, test cases), proper **scheduling** of changes, accompanied with **management of variability** within requirements.

Further information on Requirements Management

💡 Key takeaway

Requirements Engineering, independent of how it is performed spans five phases:

- **Requirements Elicitation:** to gather requirements
- **Requirements Analysis:** to generate a common understanding among stakeholders
- **Requirements Specification:** to structure and document requirements in an aggreed on format
- **Requirements Verification and Validation:** to ensure the solution matches the problem
- **Requirements Management:** to track and trace changes to requirements

Die europäischen Sprachen gehören zu ein und derselben Familie. Ihre getrennte Existenz ist ein Mythos. In Wissenschaft, Musik, Sport usw. verwendet Europa den gleichen Wortschatz. Die Sprachen unterscheiden sich nur in ihrer Grammatik, ihrer Aussprache und ihren gebräuchlichsten Wörtern. Jedem ist klar, warum eine neue gemeinsame Sprache wünschenswert wäre: Man könnte auf die Bezahlung teurer Übersetzer verzichten. Um dies zu erreichen, wären eine einheitliche Grammatik, eine einheitliche Aussprache und mehr gemeinsame Wörter notwendig.

- Auflistung
- Auflistung
- Auflistung

ID 1.1.2.2 - Requirements Elicitation

Meta data

	EN	DE
Status	READY FOR REVIEW	
Title	Requirements Elicitation	Anforderungserhebung
Tags	• Tag 1 • Tag 2	• Tag 1 • Tag 2
Estimated Duration	5 min	
Level	☐ BEGINNER ☑ INTERMEDIATE ☐ EXPERT	
External ressources	• Ressource 1 • Ressource 2	• Ressource 1 • Ressource 2
Dependencies	ID 1.1.2.1 Overview of core activities	
Next Item	ID 1.1.2.3 - Requirements Analysis	
Goals (Teaching perspective)	• Obtain a clear understanding of the elicitation phase • Obtain a clear understanding of the key activities in elicitation • Obtain a clear understanding on the techniques, parties involved, and recommended practices in requirements elicitation	
Content description (Teaching perspective)		

ENGLISCH

[ENGLISCH] DESCRIPTION

In this learning nugget, we will look at the first phase of requirements engineering: requirements elicitation and it's key activities.

[ENGLISCH] GOALS AND VALUE

In this learning nugget you will learn …

- the key activities of requirements elicitation.
- what are stakeholders.
- requirements elicitation sources.
- the techniques of requirements elicitation.

[ENGLISCH] CONTENT

Motivation/Importance

- Requirements elicitation is the starting point of all requirements and software development activities.
- Often, we don't know all requirements from the beginning and first need to uncover them.
- A thorough requirements elicitation reduces the likelihood of issues later in the software development process due to missing or incorrect requirements.

Goals of Requirements Elicitation

It is one common misconception that requirements are somehow omnipresent and just need to be collected. Truth is, that requirements elicitation is among the most cumbersome activities in software development. In principle, during requirement elicitation, we need to identify relevant sources and elicit existing (known) but also new (unknown/hidden) requirements as a basis for:

- Understanding the domain of interest
- Further requirements analysis and refinement
- Documentation
- Considering requirements elicitation is an iterative process, thus one of the goals of requirements elicitation is to improve the elicitation process by:
  - Adapting the techniques and methods for requirements elicitation accordingly to the stakeholder characteristics
  - Negotiation, consolidation, prioritisation

Note: The requirements sources provide *candidate* requirements, often in the form of assumptions. Only the decision to accept / approve a candidate makes it a "real" (accepted) requirement. The transition from candidate requirements to accepted requirements is typically done in the verification and validation phase.

Core Activities

Requirements elicitation activities can summarized by the following steps:

1. Identify requirements sources (more on elicitation sources below) and existing requirements
2. Identify (and group) stakeholders in order to ensure effective communication and prioritization
3. Conduct structured elicitation by applying elicitation (and creativity) techniques such as interviews, questionnaires… (more on techniques below)
4. Document requirements, for example with the help of checklists and templates (e.g. Volere or self-made)
5. Structure and transfer to models in order to provide a more structured representation: What belongs where?
6. Further refine and classification in order to achieve more clarity, completeness, and unambiguity
7. Conduct further analysis, refinement and completion as basis for further (stakeholder) feedback
8. Unless not approved by stakeholders, requirements remain candidates (or assumptions)
9. Since requirements elicitation is an iterative process, go back to 2.

Requirements (elicitation) sources

- Stakeholders (is a person or a group of persons, an interest group, or an organisation that has to a certain extent an interest in the system to be developed, or that takes/should take influence on the system's development) and their goals

Note: "interest in the system" does not necessarily imply "interest in the project success". For example, unions, authorities, certification and otherwise regulatory entities may well be only interested in that a system is conforming to regulations, regardless of how successful the system is. Stakeholders can therefore be interested in regulatory compliances, public safety, or adherence to laws, no matter the consequences (e.g. effects on usability).

Exemplary stakeholders: Different stakeholders have different interests and, thus, different requirements:

- User and customers
- Legislators, certification bodies, unions, or compliance managers
- Marketing, product managers
- Domain experts
- Developers, architects, testers
- Maintenance and operation
- In case of…
  - … embedded systems: hardware, mechanics, electronic developers,
  - … information systems: process responsible, domain experts, …

- Documents

Examples:

- Regulatory documents (laws, standards, norms)
- Legacy documents (requirements specs, user manual)
- Strategic papers, marketing documents
- Documentation on business processes
- Literature

- Other Systems

Examples:

- Legacy or predecessor systems
- Competitor systems
- Systems with similar functions
- Future adjacent systems

- Many, many more

Examples:

- Usage data
- App stores
- Discussion threats and social media
- ….

Techniques

Depending on the requirements sources, we may need to use different elicitation techniques. Exemplary techniques are shown below.

- Questioning techniques:

Examples:

- Interview
- Questionnaire

- Observational techniques:

Examples:

- Field observation
- Apprenticing
- Contextual inquiry

- Creativity techniques:

Examples:

- Brainstorming
- Analogy technique

- Document-based techniques:

Examples:

- System archeology
- Perspective-based reading
- Requirements reuse

- Supporting techniques:

Examples:

- Prototyping
- User feedback mining
- Modelling

- **Important:** The choice of technique depends on the source
- The table below presents a more detailed overview of elicitation techniques, their requirements sources, and other supplementary information.

Elicitation Technique	Requirements Source	Direction of Activity	Activity Partner	Nature of Activity	Frame of Reference
Interview	Stakeholders	Questioning	Neutral	Analytical	Neutral
Questionnaire	Stakeholders	Questioning	Individual	Analytical	Neutral
Field Observation	Stakeholders, legacy system	Neutral	Neutral	Involvement	Reality
Apprenticing	Stakeholders, legacy system	Neutral	Individual	Involvement	Reality
Contextual Inquiry	Stakeholders, legacy system	Neutral	Individual	Involvement	Reality
Brainstorming	Stakeholders	Questioning	Group	Analytical	Neutral
Analogy Technique	Stakeholders	Questioning	Group	Analytical	Projecting
System Archeology	Documents, legacy system	Neutral	Neutral	Neutral	Reality
Perspective-based Reading	Documents	Neutral	Neutral	Neutral	Reality
Requirements Reuse	Documents	Neutral	Neutral	Neutral	Reality
Prototyping	Stakeholders	Demonstrating	Neutral	Involvement	Projecting
User Walkthrough	Users	Demonstrating	Individual	Involvement	Projecting
Feedback Mining	Users	Questioning	Individual	Analytical	Neutral

Example 1:

- Types of interviews
  - Open: Only topic or rough guideline is given
  - Structured: Order of the questions, exact wording, and possible answers are given
  - Semi-structured: Parts are given, parts are open
- Types of questions
  - Closed-ended: Selection from a set of predefined answers
  - Open-ended: Free response style
- Qualitative interviews: Open interviews with open-ended questions
  - Goal: Exploring topics in detail, discovering background and relationships
- Quantitative interviews: (Semi-)Structured interviews with closed-ended questions
  - Goal: Measuring quantities
- Sources of errors
  - Suggestive questions
  - Hypothetical questions

- Inhibition and distortion (e.g., by personal questions)

In reality, the most frequently used elicitation techniques are stakeholder workshops, but also the study of previously used (similar) systems and the reuse of old specifications. More creativity-centric techniques are barely used, although they have a well-deserved place in Requirements Engineering.

Example 2: Questionnaire-based survey

- Similar to interviews, but
- no possibility to interact
- no possibility for inquiries
- usually conducted individually
- scales better than interviews
- Typically done as online survey
- There are also qualitative and quantitative questionnaires

Example 3: Prototying

- Make requirements “come alive” by visualising or physically instantiating them
- Not exclusive to agile approaches
- Main goals
  - Find new or missing requirements (exploring)
  - Get feedback on innovative solutions (experimenting)
  - Get stakeholders involved
  - Elicit contextual constraints by using the prototype in its real context
- Different types of prototyping
  - Physical prototype (plastic, wood, 3D-printed)
  - Wizard-of-Oz: Prospective functionality is conducted by a human
  - Paper-and-pencil: Hand-drawn sketches of screen designs
  - GUI mockups: Linked static screens, interactive wireframes
  - Programmed vertical prototypes

Example 4: Prototyping cont. Working with mock-ups

- Specifically created model / abstraction (e.g. of GUI). Can be paper-based or implemented.
- Goal: Build something to…
- … present and discuss with stakeholders, understand their needs and preferences
- … be criticised (i.e. keep it low-fidelity, as known from Design Thinking)
- … be thrown away (not functional prototype)

[ENGLISCH] KEY TAKEAWAY

- Requirements elicitation is the activity of identifying requirements and requirements sources.
- Requirements can have many sources such as stakeholders, documents, and other systems.
- Requirements elicitation can be done in multiple techniques which are adapted according to the stakeholder characteristics.

GERMAN

[GERMAN] DESCRIPTION

Die Lerneinheit werden wir …

[GERMAN] GOALS AND VALUE

In dieser Lerneinheit werden Sie …

- Ziel 1
- Ziel 2
- Ziel 3

[GERMAN] CONTENT

Die Kakerlake

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein schreckliches Ungeziefer verwandelt. ...

Die Bedeutung

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein furchtbares Ungeziefer verwandelt. Er lag auf seinem panzerartigen Rücken, ...

[GERMAN] KEY TAKEAWAY

Die europäischen Sprachen gehören zu ein und derselben Familie. Ihre getrennte Existenz ist ein Mythos. In Wissenschaft, Musik, Sport usw. verwendet Europa den gleichen Wortschatz. Die Sprachen unterscheiden sich nur in ihrer Grammatik, ihrer Aussprache und ihren gebräuchlichsten Wörtern. Jedem ist klar, warum eine neue gemeinsame Sprache wünschenswert wäre: Man könnte auf die Bezahlung teurer Übersetzer verzichten. Um dies zu erreichen, wären eine einheitliche Grammatik, eine einheitliche Aussprache und mehr gemeinsame Wörter notwendig.

- Auflistung
- Auflistung
- Auflistung

ID 1.1.2.3 - Requirements Analysis

Meta data

	EN	DE
Status	DRAFT	
Title	Requirements Analysis	Titel
Tags	• Tag 1 • Tag 2	• Tag 1 • Tag 2
Estimated Duration	20 min	
Level	☐ BEGINNER ☑ INTERMEDIATE ☐ EXPERT	
External ressources	• Ressource 1 • Ressource 2	• Ressource 1 • Ressource 2
Dependencies	ID 1.1.2.1 Overview of core activites	
Next Item	ID 1.1.2.2 - Requirements Elicitation	
Goals (Teaching perspective)	• Understand the requirements analysis phases • Understand the core activities inside requirements analysis • See examples of output from the core activities	
Content description (Teaching perspective)		

ENGLISCH

[ENGLISCH] DESCRIPTION

In this learning nugget, we will look at the phase requirements analysis and its involved core activities. We investigate techniques for those core activities and look at examples of their outcome.

[ENGLISCH] GOALS AND VALUE

In this learning nugget you will learn …

- what requirements analysis is and why it is paramount for requirements engineering
- which core activities requirements analysis entails and how they work together
- what the outcome of those core activities potentially looks like

[ENGLISCH] CONTENT

Motivation/Importance

When eliciting requirements, many needs and constraints of the various stakeholders remain in various forms, depending on the used source and employed elicitation techniques. Often, the requirements are documented in protocols or meeting minutes from workshops and interviews. Solely taking these needs and constraints as is would not serve any use, as they vary in granularity, may have duplicates or overlap with other requirements and they may have no common format.

Goals of Requirements Analysis

The main goal of requirements analysis is the refinement and classification of the previously gathered requirement candidates to ensure they:

- Reflect the needs/constraints of all stakeholders
- Resolve potential conflicts
- Enable traceability between needs/constraints and the requirement candidates
- Facilitate communication about requirement candidates among stakeholders

Core Activities

The core activities in this phase include:

1. structurization of requirements candidates building a hierarchy of needs and constraints
2. refinement of the requirements candidates, so that they are of similar granularity in the hierarchy and so that they allow to systematically classify them according to different types (e.g. functional requiremetns)
3. modeling of the requirements candidates and surrounding information to support decisions
4. explanation, evaluation and decision on requirements candidates through involved stakeholders to ensure a common understanding

5. if required, evaluation of the feasibility of the requirements candidate by different stakeholder perspectives

Example: Structurization of Requirement Candidates

The structuring of requirement candidates helps handling overlaping requirements and duplicates between requirements effectively, and, on the other hand, it helps building a hierarchy of goals to visualize their interdependencies facilitating conflict resolution, constraint detection and resource sharing. A more detailed explanation of goals is given in later learning nuggets.

For now, we only need to understand that it is important to reflect upon goals. For example, a high-level goal could be to develop a cheap system. In order to achieve a cheap system, the system must be cheap to build, cheap to run, and cheap to maintain. In order for the system to be cheap to maintain, it needs to be robust and reliable, however, this conflicts already with the goal of being cheap to build.

While there are many ways to structure goals in complex goal models (e.g. CGM, piStar), these approaches remain often at the more theorical level with little practical application. In practice, we may aggregate goals in simple lists, to not overcomplicate matters. For the abve example, such a list could for instance look like the following one:

Identifier	Goal	Priority	Level	Relation	Notes
G1	Cheap system	high	high		
G2	System cheap to build	medium	medium	supports G1	
G3	System cheap to maintain	low	medium	supports G1	
G4	Robust and reliable system	low	low	supports G3	Conflicts G2

Example: Modeling of Requirements Candidates and Metadata

To facilitate further analyses, discussions and decisions, the requirements candidates and their surrounding information, such as related stakeholders, can be modelled. Examples of such can be stakeholder models entailing the individuals, groups, or institutions with responsibility for requirements and major interests in the project (e.g. UML actor hierarchy), or a domain model describing the external systems, that interact with the system under development (e.g. UML activity diagram or BPMN). In later learning nuggets, we provide more concrete examples while taking a more practical perspective.

Involved Stakeholders

At this phase in the requirement engineering process, various stakeholders should be involved to support the analysis. The leading stakeholders in requirement analysis are roles that have knowledge about the operational context and domain.

However, depending on the core activity, other roles well be necessary to contribute to an effective elicitation. Note that these roles may well include also members of the engineering team. For example, for the evaluating requirements candidates towards their understandability or feasibility, test engineers might provide very valuable input. Especially when if comes to creating and ensuring a common understanding of the requirements, it makes sense to involve stakeholders from diverse backgrounds, covering both, the problem and the solution space.

[ENGLISCH] KEY TAKEAWAY

- Requirements Analysis is about the refinement and classification of candidate requirements
- Requirements Analysis typically entails five core activities
- The main responsible stakeholders are business analysts and requirements engineers

GERMAN

[GERMAN] DESCRIPTION

Die Lerneinheit werden wir …

[GERMAN] GOALS AND VALUE

In dieser Lerneinheit werden Sie …

- Ziel 1
- Ziel 2
- Ziel 3

[GERMAN] CONTENT

Die Kakerlake

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein schreckliches Ungeziefer verwandelt. ...

Die Bedeutung

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein furchtbares Ungeziefer verwandelt. Er lag auf seinem panzerartigen Rücken, ...

[GERMAN] KEY TAKEAWAY

Die europäischen Sprachen gehören zu ein und derselben Familie. Ihre getrennte Existenz ist ein Mythos. In Wissenschaft, Musik, Sport usw. verwendet Europa den gleichen Wortschatz. Die Sprachen unterscheiden sich nur in ihrer Grammatik, ihrer Aussprache und ihren gebräuchlichsten Wörtern. Jedem ist klar, warum eine neue gemeinsame Sprache wünschenswert wäre: Man könnte auf die Bezahlung teurer Übersetzer verzichten. Um dies zu erreichen, wären eine einheitliche Grammatik, eine einheitliche Aussprache und mehr gemeinsame Wörter notwendig.

- Auflistung
- Auflistung
- Auflistung

ID 1.1.2.4 - Requirements Specification

Meta data

	EN	DE
Status	DRAFT	
Title	Title	Titel
Tags	• Tag 1 • Tag 2	• Tag 1 • Tag 2
Estimated Duration	5 min	
Level	☐ BEGINNER ☑ INTERMEDIATE ☐ EXPERT	
External ressources	• Ressource 1 • Ressource 2	• Ressource 1 • Ressource 2
Dependencies	ID 1.1.2.1 Overview of core activities	
Next Item	ID 1.1.2.5 Verification and Validation	
Goals (Teaching perspective)	• Obtain a clear understanding of the specification phase • Obtain a clear understanding of the key activities in specification	
Content description (Teaching perspective)		

ENGLISCH

[ENGLISCH] DESCRIPTION

In this learning nugget, we will look at the phase of requirements specification and it's key activities.

[ENGLISCH] GOALS AND VALUE

In this learning nugget you will learn …

- the importance and goals of requirements specification
- what is an SRS document
- stakeholder involvement in requirements specification

[ENGLISCH] CONTENT

Motivation/Importance

A well documented requirements specification forms the the basis for all other activities in the development lifecycle and the quality of the specification is a foundation for the project success. Note, however, that there is no one-size-fits-all blueprint for how exactly a requirements specification should look like in terms of contents and form. In any case, it still sets the foundation (1) traceability of the requirements gathered in previous stages, and (2) providing an early agreement between all parties involved and thus avoids making cost-intensive assumption about things that nobody really needs.

Generally speaking, a well documented requirements specification serves several purposes by:

- reducing ambiguity and the risks of misunderstandings by clearly presenting requirements
- setting timelines and milestones
- allowing for estimating costs in advance
- guiding developers in understanding how the product will be designed
- enhancing client satisfaction when the what is agreed on in the SRS is fully implemented

Depending on the project situation, we may need to document the requirements in more detail. For example, a tendering process in which we have suppliers developing a system will require far more documented details then an internal development process where we can ensure a continuous communication line.

In later learning nuggets, we will introduce a clear structure that guides the requirements specification.

💡 [ENGLISCH] KEY TAKEAWAY

- Requirements specification phase leads to the creation of a structured document called “Software requirements specification“. This phase also ensures:
  - traceability
  - decidability
  - formal mutual agreement with stakeholders

GERMAN

ℹ️ [GERMAN] DESCRIPTION

Die Lerneinheit werden wir …

🎯 [GERMAN] GOALS AND VALUE

In dieser Lerneinheit werden Sie …

- Ziel 1
- Ziel 2
- Ziel 3

📄 [GERMAN] CONTENT

Die Kakerlake

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein schreckliches Ungeziefer verwandelt. ...

Die Bedeutung

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein furchtbares Ungeziefer verwandelt. Er lag auf seinem panzerartigen Rücken, ...

[GERMAN] KEY TAKEAWAY

Die europäischen Sprachen gehören zu ein und derselben Familie. Ihre getrennte Existenz ist ein Mythos. In Wissenschaft, Musik, Sport usw. verwendet Europa den gleichen Wortschatz. Die Sprachen unterscheiden sich nur in ihrer Grammatik, ihrer Aussprache und ihren gebräuchlichsten Wörtern. Jedem ist klar, warum eine neue gemeinsame Sprache wünschenswert wäre: Man könnte auf die Bezahlung teurer Übersetzer verzichten. Um dies zu erreichen, wären eine einheitliche Grammatik, eine einheitliche Aussprache und mehr gemeinsame Wörter notwendig.

- Auflistung
- Auflistung
- Auflistung

ID 1.1.2.5 - Verification and Validation

Meta data

	EN	DE
Status	DRAFT	
Title	Title	Titel
Tags	• Tag 1 • Tag 2	• Tag 1 • Tag 2
Estimated Duration	5 min	
Level	☐ BEGINNER ☐ INTERMEDIATE ☐ EXPERT	
External ressources	• Ressource 1 • Ressource 2	• Ressource 1 • Ressource 2
Dependencies		
Next Item		
Goals (Teaching perspective)		
Content description (Teaching perspective)		

ENGLISCH

[ENGLISCH] DESCRIPTION

In this learning nugget, we will look at the requirements verification and validation phase and its involved core activities. We investigate techniques for those core activities and look at examples of their outcome.

[ENGLISCH] GOALS AND VALUE

In this learning nugget you will learn …

- what requirements verification and validation is and why it is important for successful requirements engineering
- which core activities requirements verification and validation entails and how they work together
- what the outcome of those core activities potentially looks like

[ENGLISCH] CONTENT

Motivation/Importance

Until this point, all needs and constraints we elicited need to be considered “requirements candidates“, which means that they are assumptions of what the stakeholders may or may not really need. Requirements validation and verification is about transforming these requirements candidates into accepted requirements through vadalidation by the customers following the scope set out initially. This may be done in an manner as simple as asking them to formally accept the requirements in a written form and extend to employing a structured workshop with a walkthrough through all requirements and asking “Why” questions (e.g. “why do you really need that requirement?”, “What is the purpose?” “What happens if that requirement is not implemented?”).

Goals of Requirements Verification and Validation

The main goal is to ensuring all stakeholders agree on requirements, building the basis for liability (e.g. Lastenheft). This ensures,

- that the customer gets what they expect
- the developing stakeholders can be sure they address the customers expectations correctly

Requirements Validation

> *Ensure the right system is built. Validate, that the requirements correctly reflect the customers needs.*

Example

Requirement: The user must be able to log in using their username and password.

Checks to perform with customer:

- Completeness: Does this requirements cover all necessary information? Or should users by default login with their username, password and a second factor?
- Consistency: Does this requirements conflict with other requirements? Is there a different requirement prescribing users have to use a passkey to log in?
- Clarity: Is the requirements clear and unambiguous to the customer? Can there be missunderstandings towards which functionality requires a login?

Requirements Verification

> *Ensure the system is built the right way. Verify, that the requirements have been implemented correctly.*

Example

Requirement: The application must respond to user requests within 2 seconds.

Verification Activities:

- Define acceptance criteria: Application responds within 2 seconds
- Identify edge cases: Is the system allowed to respond slower under load?
- Implement verification mechanisms: Write test cases measuring response time in different scenarios

Note that requirements verification is typically in scope of (acceptance) testing which, in many cases, comes with own teams and phases in the respective software process model.

Core Activities

1. Evaluate Requirements against customer expectations to validate, that the requirements correctly reflect the customers needs.
2. Prototyping to support customers in getting a clear vision for the system
3. Formal verification for systems with respective requirements such as avionic systems
4. Reviews of requirements and the supporting information (e.g. goals, stakeholders, implementation) to validate and verify them

5. Specification of test cases to ensure requirements can be verified in an automated fashion

Techniques

Common techniques to perform validation and verification are:

- Presentations and workshop to evaluate the requirements against the customer's expectations
- Roleplay to show-case the systems usage and respective requirements
- Prototyping of parts of systems, especially if the overall project is very large

Involved Stakeholders

At this phase of the requirement engineering lifecycle, the main responsibles are requirements engineers. They coordinate validation with the respective stakeholders, such as customers, and enable verification with modeling experts in case formal verification is required, or test engineers for the specification of test cases.

[ENGLISCH] KEY TAKEAWAY

- Requirements Verification and Validation is about transitioning requirement candidates into agreed requirement, that act as basis for the implementation
- Validation is about ensuring the customers requirements are reflected correctly
- Verification is about ensuring the requirements are implemented correctly
- Requirements Verification and Validation entails one mandatory and four voluntary core activities

GERMAN

[GERMAN] DESCRIPTION

Die Lerneinheit werden wir …

[GERMAN] GOALS AND VALUE

In dieser Lerneinheit werden Sie …

- Ziel 1
- Ziel 2
- Ziel 3

[GERMAN] CONTENT

Die Kakerlake

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein schreckliches Ungeziefer verwandelt. ...

Die Bedeutung

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein furchtbares Ungeziefer verwandelt. Er lag auf seinem panzerartigen Rücken, ...

💡 [GERMAN] KEY TAKEAWAY

Die europäischen Sprachen gehören zu ein und derselben Familie. Ihre getrennte Existenz ist ein Mythos. In Wissenschaft, Musik, Sport usw. verwendet Europa den gleichen Wortschatz. Die Sprachen unterscheiden sich nur in ihrer Grammatik, ihrer Aussprache und ihren gebräuchlichsten Wörtern. Jedem ist klar, warum eine neue gemeinsame Sprache wünschenswert wäre: Man könnte auf die Bezahlung teurer Übersetzer verzichten. Um dies zu erreichen, wären eine einheitliche Grammatik, eine einheitliche Aussprache und mehr gemeinsame Wörter notwendig.

- Auflistung
- Auflistung
- Auflistung

ID 1.1.2.6 - Requirements Management

Meta data

	EN	DE
Status	READY FOR REVIEW	
Title	Title	Titel
Tags	• Tag 1 • Tag 2	• Tag 1 • Tag 2
Estimated Duration	5 min	
Level	☐ BEGINNER ☑ INTERMEDIATE ☐ EXPERT	
External ressources	• Ressource 1 • Ressource 2	• Ressource 1 • Ressource 2
Dependencies	ID 1.1.2.1 Overview of core activities	
Next Item	ID 1.2 - AMDiRE4KMU - TOPIC AREA - S4KMU	
Goals (Teaching perspective)	• Obtain a clear understanding of the requirements management phase • Obtain a clear understanding of the key activities in requirements management	
Content description (Teaching perspective)		

ENGLISCH

[ENGLISCH] DESCRIPTION

In this learning nugget, we will look at the phase of requirements management and it's key activities.

[ENGLISCH] GOALS AND VALUE

In this learning nugget you will learn …

- the motivation and goals of requirements management
- the key activities of requirements management.

[ENGLISCH] CONTENT

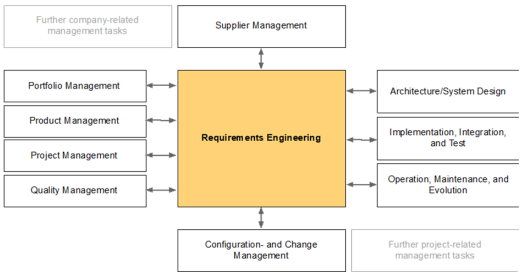
Motivation/Importance

Effectively and efficiently managing requirements throughout their entire lifecycle is important for various reasons, such as:

- Requirements specifications can become very large as the project evolves and they need to be managed in order to avoid redundancy and repetition.
- Requirements may change (even after when a system is deployed into production) due to internal or external causes. Managing these changes and tracking down the versions is important.
- Requirements influence other artefacts (also beyond RE) since they are the starting point for all later activities.
- Requirements are "used" by several persons, thus it is important that all these persons are on the same page.
- Requirements have different release states (assumption, reviewed, accepted / rejected,…). Thus, management ensures tracking of progress.
- Meta information of requirements are important for controlling the RE process (e.g., author, responsible, current assignee)

- Requirements may be reused in other projects. Thus, well managed and documented requirements in one project will allow you to save time and effort on other projects.
- Requirements need to be kept up to date (for an effective project control) in order to ensure that decisions are being made according to accurate and updated information.

This shows already how requirements are used along various activities in the software development lifecycle, as also shown in the following figure:



Goals of Requirements Management

Requirements Management aims at efficiently and effectively managing and using requirements along the whole system lifecycle.

- Elementary tasks include, but are not limited to:
  - Archiving (storing current and older versions of the requirements) and baselining requirements (taking how requirements look at a specific point as a reference)
  - Modifying requirements due to new knowledge/insights
  - Tracing and verifying requirements (impact analysis, support of change processes)

Core Activities

Requirements management includes various activities, such as:

- Attribution: Adding meta information to requirements for an effective control
- Assessment and prioritisation
- Tracing: Describe and follow the lifecycle of a requirement
- Versioning: Describe and follow changes within the requirements
- Reporting: Collecting, analysing, and visualising information about RE
- Controlling the RE process

In essence, a proper requirements management provides answers to the following questions:

- Which requirements come from which source? (attribution, tracing)
- Which requirements are how important? (attribution, assessment)
- Who changed a requirement when? (attribution, versioning)
- How efficient is the RE process? (controlling)

Change Management

One important synergy to consider between requirements management and other activities in the software development lifecycle is an overarching change management. Here, it is important to note that the goals of change management dictate, to some extent, also how we engage in requiremetns management as it should effectively allow for:

- Systematic control of changes and their impacts
- Assessment of planned changes (incl. their effort and risks)
- Structured decision of whether to implement change (or not)
- Reproducible implementation of change

Related tasks may well include:

- Impact analysis: What are the effects of the change?
- Versioning: Which version of requirements and artefacts exist?
- Configuration management: Which requirements and artefacts build a consistent baseline?
- Claim management: What was agreed on and how does the shipped solution satisfy the (formally accepted) requirements?

[ENGLISCH] KEY TAKEAWAY

- Requirements management is an important phase of requirements engineering since it aims to effectively manage and use requirements along the whole system lifecycle. The main tasks of requirements management are:
  - Attribution (Adding meta information to requirements for an effective control)
  - Assessment and prioritisation
  - Tracing (Describe and follow the lifecycle of a requirement)
  - Versioning (Describe and follow changes within the requirements)
  - Reporting (Collecting, analysing, and visualising information about RE)
  - Controlling the RE process

GERMAN

[GERMAN] DESCRIPTION

Die Lerneinheit werden wir …

[GERMAN] GOALS AND VALUE

In dieser Lerneinheit werden Sie …

- Ziel 1
- Ziel 2
- Ziel 3

[GERMAN] CONTENT

Die Kakerlake

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein schreckliches Ungeziefer verwandelt. ...

Die Bedeutung

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein furchtbares Ungeziefer verwandelt. Er lag auf seinem panzerartigen Rücken, ...

[GERMAN] KEY TAKEAWAY

Die europäischen Sprachen gehören zu ein und derselben Familie. Ihre getrennte Existenz ist ein Mythos. In Wissenschaft, Musik, Sport usw. verwendet Europa den gleichen Wortschatz. Die Sprachen unterscheiden sich nur in ihrer Grammatik, ihrer Aussprache und ihren gebräuchlichsten Wörtern. Jedem ist klar, warum eine neue gemeinsame Sprache wünschenswert wäre: Man könnte auf die Bezahlung teurer Übersetzer verzichten. Um dies zu erreichen, wären eine einheitliche Grammatik, eine einheitliche Aussprache und mehr gemeinsame Wörter notwendig.

- Auflistung
- Auflistung
- Auflistung

ID 1.2.1.1 - AMDiRE Core Components

Meta data

	EN	DE
Status	READY FOR REVIEW	
Title	AMDiRE Core Components	AMDiRE Kernkomponenten
Tags	• Tag 1 • Tag 2	• Tag 1 • Tag 2
Estimated Duration	5 min	
Level	☐ BEGINNER ☑ INTERMEDIATE ☐ EXPERT	
External ressources	• Ressource 1 • Ressource 2	• Ressource 1 • Ressource 2
Dependencies	ID 1.2.1.1 - AMDiRE Core Components ID 1.2.1.2 - Role Model	
Next Item	ID 1.2.1.2 - Role Model	
Goals (Teaching perspective)	• Obtain a clear understanding on the basic components of the artefact-based RE model	
Content description (Teaching perspective)		
Content summary (Teaching perspective)		

Content data

EN DE

DESCRIPTION

In this learning nugget we will …

- get a basic introduction into the AMDiRE approach and its underlying philosophy

Die Lerneinheit werden wir …

- look into one of the basic components of artefact-based RE

🎯 GOALS AND VALUE (User perspective)

In this learning nugget you will …

- understand the relevance of an artefact-based philosophy
- get a basic introduction into the AMDiRE approach and its underlying philosophy of artefact orientation
- get a first look at the basic components of the AMDiRE model and the interactions of these components

In dieser Lerneinheit werden Sie …

- Ziel 1
- Ziel 2
- Ziel 3

📄 CONTENT

Artefact-based Requirements Engineering (RE) is an approach to specifying requirements with the goal to support companies to focus on the content and structure of their requirements artifacts (What needs to be specified?), rather than solely on the RE processes they use for such specification (How should we proceed step by step?).

The simple reason for focussing on the what over the how is that the detailed way we proceed in our processes can by no means be standardised, because it depends on many surrounding factors, such as

- the chosen process model (e.g. agile versus plan-driven)
- the project situation (e.g. public tending versus in-house development, or developing a new system versus replacing a system)
- the application domain (e.g. software for the medical domain versus software for eCommerce)
- the capabilities of the various stakeholders

In contrast, what can be standardised are the results created during Requirements Engineering. We call these results (requirements) **artefacts**. In very simple terms, we define an artefact the following way.

> An artefact is a deliverable of major interest that abstracts from contents of a specification document. It is used as input, output, or as an intermediate result of a process step.

In an artefact-based RE approach, we essentially do not define a complex series of process steps which would then provide little to no real guidance, but we define the major expected output (artefacts), how these different outputs related to each other (artefact model) as well as roles and responsibilities (role model) and how to navigate through the artefacts in terms of a coarse process (miles stones).

Die Kakerlake

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein schreckliches Ungeziefer verwandelt. ...

Die Bedeutung

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein furchtbares Ungeziefer verwandelt. Er lag auf seinem panzerartigen Rücken, ...

Artefact-based RE offers several benefits, making it particularly useful for small and medium-sized enterprises (SMEs). It emphasises the expected quality of requirements, while process-based approaches mainly focus on the correctness of process execution. We experienced that in the end: what matters are the results and not so much whether we follow a standard process guide to create those results. Given the closeness of RE to so many influencing factors - among them the skills, expectations and experiences (and even preferences) of the various stakeholders with respect to how to do RE as well as surrounding process models - these processes are as discussed not standardisable via a blueprint anyway. What can be captured in such a blueprint, however, are the main results (artefacts). Artefact-based RE approaches provide such a blueprint based on the best practices of RE.

In the series of learning paths at hands, we introduce you to our very specific and well established and practically proven artefact-based RE approach called the *Artefact Model for Domain-Independent Requirements Engineering* (**AMDiRE**). This learning nugget covers the very basic components of AMDiRE and an overview of their relationships. A far more detailed introduction into AMDiRE can be also taken from our publicly available scientific article that also introduces into the history of AMDiRE (emerging in industrial context along decades of applied research).

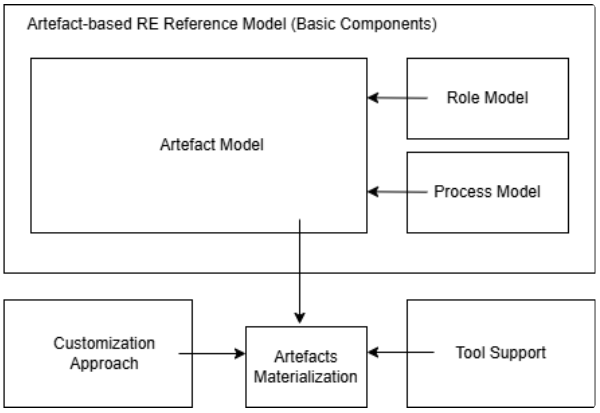
Artefact-based RE, in essence, involves specifying requirements (artefacts) following the contents and relationships outlined in the artefact model. This model provides a flexible blueprint for creating the requirements artefacts - regardless of the form in which we want to document the results. The artefact model is the backbone of the AMDiRE approach.

Two other models support the application of requirements in pracgice:

- Role Model: Describes the roles involved in the project and their responsibility for specific artefacts.
- Process Model: Outlines the milestones achieved during the specification of the artefacts.

The three core components of AMDiRE—artefact model, role model, and process model—interact to form the basis for executing RE. They provide a blueprint for requirements specification, linking instances of requirements (artefacts) to the roles responsible for them and milestones in the process execution.

General overview



Following the artefact-based philosophy, AMDiRE has several components that are all rooted in the artefact model. Beyond the basic components, introduced above and sufficient to define a project specific approach to RE, AMDiRE uses a what we characterise as customisation approach and tool support. It is the latter that makes AMDiRE practicable. The customisation approach supports selecting only those artefacts and their contents that are really relevant - otherwise it would remain a purely academic exercise - and tool support is what helps creating the artefacts. On this platform, we address both with our toolbox.

💡 Key takeaway

Artefact-based RE Reference model consists of the following components:

- Artefact Model
- Role Model
- Process Model

Die europäischen Sprachen gehören zu ein und derselben Familie. Ihre getrennte Existenz ist ein Mythos. In Wissenschaft, Musik, Sport usw. verwendet Europa den gleichen Wortschatz. Die Sprachen unterscheiden sich nur in ihrer Grammatik, ihrer Aussprache und ihren gebräuchlichsten Wörtern. Jedem ist klar, warum eine neue gemeinsame Sprache wünschenswert wäre: Man könnte auf die Bezahlung teurer Übersetzer verzichten. Um dies zu erreichen, wären eine einheitliche Grammatik, eine einheitliche Aussprache und mehr gemeinsame Wörter notwendig.

- Auflistung
- Auflistung
- Auflistung

ID 1.2.1.2 - Role Model

Meta data

	EN	DE
Status	READY FOR REVIEW	
Title	AMDiRE Role model	Rollenmodell in AMDiRE
Tags	• Tag 1 • Tag 2	• Tag 1 • Tag 2
Estimated Duration	5 min	
Level	☐ BEGINNER ☑ INTERMEDIATE ☐ EXPERT	
External ressources	• Ressource 1 • Ressource 2	• Ressource 1 • Ressource 2
Dependencies	ID 1.2.1.1 - AMDiRE Core Components ID 1.2.1.3 - Process Model	
Next Item	ID 1.2.1.3 - Process Model	
Goals (Teaching perspective)		
Content description (Teaching perspective)		
Content summary (Teaching perspective)		

Content data

EN DE

DESCRIPTION

In this learning nugget we will look into one of the core components of the AMDiRE model: role model and it's importance.

Die Lerneinheit werden wir …

🎯 GOALS AND VALUE (User perspective)

In this learning nugget you will learn…

- what roles generally are and why they are important
- how the role model fits in the AMDiRE model

In dieser Lerneinheit werden Sie …

- Ziel 1
- Ziel 2
- Ziel 3

CONTENT

A role model generally addresses the question "Who is involved in requirements engineering and what are the responsibilities of the roles?" It essentially lays out responsibilities for each member in a project through their relationships with the Process model and Artifact model, as well as their necessary skills and abilities.

Roles

Roles have explicitly defined skills and responsibilities for an artefact. Usually and when following an artefact-based approach, each artefact has one associated role that has the responsibility for creating the artefact and for maintaining its quality in a level sufficiently high so that other roles that use that artefact as input can do their work.

Components of Role Model

A role model consists of three very basic elements: The name of the role, its basic description, and its responsibilities. The latter consists of a list of duties/tasks that the role must perform to help achieve a set of functional goals and/or quality goals, and constraints that represents a list of conditions that the role must take into consideration when performing its responsibilities. In an artefact-based approach, as we postulate here, the responsibility is the creation and maintenance of the associated artefact and it comes with the skills required to do so (e.g. domain expertise).

Role Model in AMDiRE

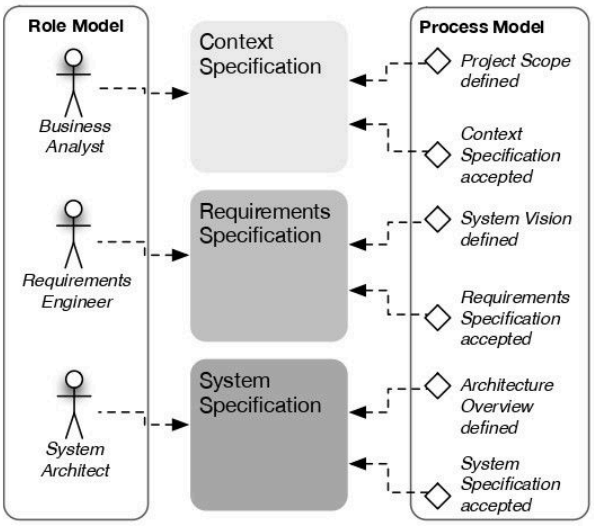
For each artefact, we define one particular role, which has the responsibility for that artefact, independent of other potentially supporting roles provided by the software process model (e.g. quality manager), and independent of whether same persons are assigned to different roles in a project. Needless to say that one person can also take several roles in a project.

Die Kakerlake

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein schreckliches Ungeziefer verwandelt. ...

Die Bedeutung

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein furchtbares Ungeziefer verwandelt. Er lag auf seinem panzerartigen Rücken, ...



Main roles in Role Model

1. **Domain Expert**: Responsible for Software Context Specification artefacts and needs to show relevant domain knowledge.
2. **Requirements Engineer**: Responsible for Requirements Specification artefacts and usually a mediator between domain expert and the software development team (e.g. the system architect)
3. **System Architect**: Responsible for System Specification artefacts.

These roles are generic, meaning they outline the fundamental responsibilities in the requirements engineering process and can be customized to meet the specific needs of individual projects.

Characteristics for roles in Role Model

Different roles have different responsibilities (see above) and, in consequence, they require different skill sets outlined below

Domain expert:

- Profound domain knowledge of the operational context of the system (e.g. workflows in a company)
- Familiar with the relevant regulatory domain (can at least identify standards and regulations relevant to the domain)

Requirements engineer:

- Good communication skills
- Critical thinking
- Empathy and conflict resolution skills
- (Not necessarily a domain expert)
- (Not necessarily a technical expert)

System Architect:

- Good technical skills
- Good problem solving skills

- Good communication skills

💡 Key takeaway

AMDiRE roles are:

- Domain experts
- Requirements Engineer as mediator between context and development
- Architect as bridge into the solution design

The role model helps identify the participants responsible for the specification and maintenance of particular artifacts. It also clarifies the basic abilities that these roles possess and apply within the requirements engineering process.

Die europäischen Sprachen gehören zu ein und derselben Familie. Ihre getrennte Existenz ist ein Mythos. In Wissenschaft, Musik, Sport usw. verwendet Europa den gleichen Wortschatz. Die Sprachen unterscheiden sich nur in ihrer Grammatik, ihrer Aussprache und ihren gebräuchlichsten Wörtern. Jedem ist klar, warum eine neue gemeinsame Sprache wünschenswert wäre: Man könnte auf die Bezahlung teurer Übersetzer verzichten. Um dies zu erreichen, wären eine einheitliche Grammatik, eine einheitliche Aussprache und mehr gemeinsame Wörter notwendig.

- Auflistung
- Auflistung
- Auflistung

ID 1.2.1.3 - Process Model

Meta data

	EN	DE
Status	READY FOR REVIEW	
Title	AMDiRE Process Modell	AMDiRE Prozessmodell
Tags	• Tag 1 • Tag 2	• Tag 1 • Tag 2
Estimated Duration	5 min	
Level	☐ BEGINNER ☑ INTERMEDIATE ☐ EXPERT	
External ressources	• Ressource 1 • Ressource 2	• Ressource 1 • Ressource 2
Dependencies	ID 1.2.1.1 - AMDiRE Core Components ID 1.2.1.3 - Process Model	
Next Item	ID 1.2.1.4 - AMDiRE Core Artefacts - LEARNING NUGGET	
Goals (Teaching perspective)		
Content description (Teaching perspective)		
Content summary (Teaching perspective)		

Content data

EN DE

DESCRIPTION

In this learning nugget, we will look at one of the core components of the AMDiRE model: the process model.

Die Lerneinheit werden wir …

🎯 GOALS AND VALUE (User perspective)

In this learning nugget you will …

- learn what a process model is and why it is important
- learn about the milestones that constitute the AMDiRE model

In dieser Lerneinheit werden Sie …

- Ziel 1
- Ziel 2
- Ziel 3

📄 CONTENT

Following the artefact-based philosophy, the process model is defined by the minimal set of milestones that need to be reached in requirements engineering, answering the question "by when should which artefacts be specified (or accepted)?" Reaching the milestones, formally, is usually a point in time that gives a confidence in that the artefact has reached a sufficient level of detail or that it is formally accepted. The latter is important from a legal perspective as it comes also with liability, but in essence, it is mostly important as it strengthens our confidence in that the content is reliable and that we may proceed by using the information captured in the artefact for the creation of the next. The milestones are therefore also indirectly linked to specific artefacts that signify the completion of each milestone.

How these milestones are reached is intentionally left open to allow for a maximum degree of flexibility (following, for instance, an agile approach or a more plan-driven one). Both ways of working have their place in practice.

Process Model in AMDiRE

For each artefact type, we define essentially two milestones.

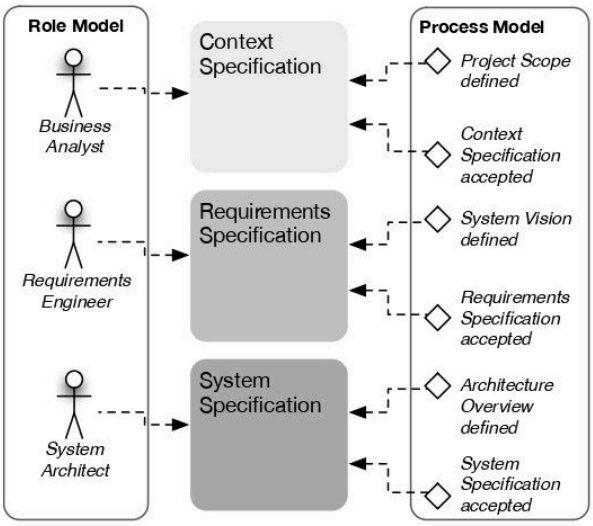
- The **first milestone** defines the point in time in which the first content item of the artefact is defined, thus, reflecting a certain maturity of the content in the artefact as the first content items serve the purpose of a summary for subsequent contents. While we will introduce the content later, a take-away now is that, for instance, the system vision in the requirements specification comprises an overview of the major use cases; its definition and agreement indicate therefore that the use cases are sufficiently defined to be further elaborated in detail and, thus, allowing, for example, for first cost estimations based on function points or for inclusion in a contract.
- The **second milestone** of each artefact indicates the point in time when an artefact is finalised, respectively formally accepted.

Die Kakerlake

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein schreckliches Ungeziefer verwandelt. ...

Die Bedeutung

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein furchtbares Ungeziefer verwandelt. Er lag auf seinem panzerartigen Rücken, ...



Milestones (Overview):

- **Artefact:** Context Specification:
  - **First:** Project Scope defined
  - **Second:** Context Specification accepted
- **Artefact:** Requirements Specification
  - **First**: System vision defined
  - **Second**: Requirements Specification
- **Artefact:** System Specification
  - **First:** Architecture overview defined
  - **Second**: System Specification accepted

Goal of milestones

The process model and milestones serve the purpose of a process integration as they give us the opportunity to formally embed the artefacts into project-specific decisions. These decisions are to be taken at a specific point in time, such as when to conduct first cost estimations, when to make make-or-buy decisions, or when changes in the requirements should be formally defined via change requests.

Key takeaway

The process model, one of the key components of the AMDiRE model. It constitutes of a set of milestones. Every artefact type has two sets of milestones.

Process model serves to integrate and instantiate the process, providing a framework to formally embed artifacts into project-specific decisions.

Die europäischen Sprachen gehören zu ein und derselben Familie. Ihre getrennte Existenz ist ein Mythos. In Wissenschaft, Musik, Sport usw. verwendet Europa den gleichen Wortschatz. Die Sprachen unterscheiden sich nur in ihrer Grammatik, ihrer Aussprache und ihren gebräuchlichsten Wörtern. Jedem ist klar, warum eine neue gemeinsame Sprache wünschenswert wäre: Man könnte auf die Bezahlung teurer Übersetzer verzichten. Um dies zu

erreichen, wären eine einheitliche Grammatik, eine einheitliche Aussprache und mehr gemeinsame Wörter notwendig.

- Auflistung
- Auflistung
- Auflistung

ID 1.2.1.4 - Artefact Model

Meta data

	EN	DE
Status	READY FOR REVIEW	
Title	AMDiRE Core Artefacts	AMDiRE Kernartefakte
Tags	• Tag 1 • Tag 2	• Tag 1 • Tag 2
Estimated Duration	5 min	
Level	☐ BEGINNER ☑ INTERMEDIATE ☐ EXPERT	
External ressources	• Ressource 1 • Ressource 2	• Ressource 1 • Ressource 2
Dependencies	ID 1.2.2.1 - AMDiRE Core Artefacts	
Next Item		
Goals (Teaching perspective)		
Content description (Teaching perspective)		
Content summary (Teaching perspective)		

Content data

EN DE

DESCRIPTION

In this learning nugget we will …

- introduce the three layers of the AMDiRE model
- list the artefacts that capture the relevant information at each layer

Die Lerneinheit werden wir …

🎯 GOALS AND VALUE (User perspective)

In this learning nugget you will …

- learn about the three levels of abstraction introduced before and how the AMDiRE artefact model relates to those levels
- understand the relation of the more abstract concept of “levels of abstraction” and how it manifests in practice via concrete hands-on artefacts

In dieser Lerneinheit werden Sie …

- Ziel 1
- Ziel 2
- Ziel 3

CONTENT

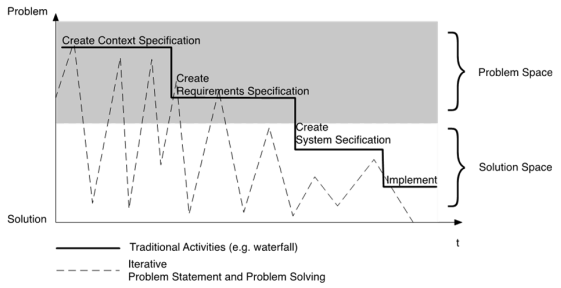
Artefact Model

The artefact model in AMDiRE defines a family of requirements artefacts and their dependencies. This model is central to the artefact-based approach and guides all relevant roles on what needs to be specified and how the contents should relate to other contents. The basic component of the artefact model is the artefact, a deliverable used as input, output, or intermediary result.

The artifact model captures three basic types of artefacts:

- **Context Specification**: Define the operational context, i.e. the broader problem space, including the project scope, stakeholders, rules, goals, constraints, and the domain model specification. Answers the “WHY” question introduced before.
- **Requirements Specification**: Captures the problem space in connection to the concrete software system, defining user requirements without addressing how it should be implemented. Answers the “WHAT” question introduced before.
- **System Specification**: Captures the solution space and how software system shall be implemented in order to satisfy the previously specified requirements. Often, these artefacts are not the primary focus of requirements engineering activities but are captured in AMDiRE to show how the contents relate to subsequent (downstream) activities to facilitate the integration into the software development lifecycle . Answers the “HOW” question introduced before.

We have seen this distinct into abstract layers before in for ID 1.1.1.6 - Functional Requirements and for ID 1.1.1.7 - Non-functional Requirements.



These three types of artifacts capture each two viewpoints:

- **Content Model**: Focuses on concrete concepts and information specified in artifacts, excluding modeling concepts such as specific specification methods.
- **Structure Model**: Provides a logical structuring of the concepts and information into **content items**. It is used for integrating the artifact model with the role and process models.

To navigate the artifact model, it is best to start by focusing on the content items reflected in the artifact structure model and then use the artifact content model to identify the specific information and concepts related to the content item of interest. For reasons of simplicity, on this platform, we do not distinguish further between those two viewpoints. Rather, we focus on the structure by showing which artefacts to consider and which content items they should have (e.g. what is part of the requirements specification). In reality and when documenting requirements, each content item may then materialise as a section in a document. The content model in turn defines a set of rules and how to document each of the content items. This is implicitly reflected by how we elaborate the learning nuggets and with the contents you will find in the toolbox.

Next we provide an overview of content items constituting structure of the three types of artefacts. Note that details will be given in the subsequent learning paths. Further, note that we will not introduce for each of the content items one individual learning nugget, but will focus on what we consider to be the most relevant content items, taking a more practical approach.

Context Specification

The following artefacts are part of the context specification:

- Project Scope
- Constraints and rules
- Business Cases
- Stakeholder Model
- Objectives and Goals
- Domain Model
- Glossary

Requirements Specification

The following artefacts are part of the requirements specification:

- System Vision
- Usage Model
- Service Model
- Process Requirements
- Functional Hierarchy
- Data Model
- Deployment Requirements
- Risk List
- System Constraints
- Quality Requirements
- Glossary

System Specification

The following artefacts are part of the system specification:

- Architecture Overview
- Function Model
- Data Model
- Component Model
- Behaviour Model
- Glossary

Dependencies

- The artefacts belonging to the same specification have dependencies among each other and to other artefacts belonging to a different specification.

💡 Key takeaway

The AMDiRE model constitutes of two parts: content and structure. Our main focus, the structure model, constitutes of three artefacts: context specification, requirements specification, and system specification.

Die europäischen Sprachen gehören zu ein und derselben Familie. Ihre getrennte Existenz ist ein Mythos. In Wissenschaft, Musik, Sport usw. verwendet Europa den gleichen Wortschatz. Die Sprachen unterscheiden sich nur in ihrer Grammatik, ihrer Aussprache und ihren gebräuchlichsten Wörtern. Jedem ist klar, warum eine neue gemeinsame Sprache wünschenswert wäre: Man könnte auf die Bezahlung teurer Übersetzer verzichten. Um dies zu erreichen, wären eine einheitliche Grammatik, eine einheitliche Aussprache und mehr gemeinsame Wörter notwendig.

- Auflistung
- Auflistung
- Auflistung

ID 1.2.1.5 - Walktrough of AMDiRE

Meta data

	EN	DE
Status	DRAFT	
Title	Walkthrough of AMDiRE	Durchlauf von AMDiRE
Tags	• Tag 1 • Tag 2	• Tag 1 • Tag 2
Estimated Duration	5 min	
Level	☐ BEGINNER ☑ INTERMEDIATE ☐ EXPERT	
External ressources	• Ressource 1 • Ressource 2	• Ressource 1 • Ressource 2
Dependencies	ID 1.2.1.4 - Artefact Model ID 1.2.2.1 - (delete?) AMDiRE Core Artefacts - LEARNING NUGGET ID 1.2.2.2 - AMDiRE Context Specification (Overview) ID 1.2.2.3 - AMDiRE Requirements Specification (Overview) - LEARNING PATH ID 1.2.2.4 - AMDiRE System Specification (Overview) - LEARNING PATH	
Next Item	ID 1.2.1.6 - Operationalisation	
Goals (Teaching perspective)		
Content description (Teaching perspective)		
Content summary (Teaching perspective)		

Content data

EN DE

DESCRIPTION

In this learning nugget we will …

- make a simple walk-through through the steps following the AMDiRE model.

Die Lerneinheit werden wir …

🎯 GOALS AND VALUE (User perspective)

In this learning nugget you will …

- familiar with the AMDiRE artefacts in a very simplified step by step process
- understand further the benefits of artefact orientation as it allows for a maximum of flexibility

In dieser Lerneinheit werden Sie …

- Ziel 1
- Ziel 2
- Ziel 3

CONTENT

Walk through

In essence, there is no universal way of doing Requirements Engineering as it very much depends on a plethora of project characteristics, such as:

- Do we develop a new system or do we replace an already existing system?
- What is the information already documented an known?
- We we develop internally (in-house) or do we elaborate requirements for external suppliers or do we develop requirements as external suppliers for our customer?

All this lays essentially for the foundation for how to start. In essence, however, the first step consists of customising the AMDiRE model and making it fit for purpose. This includes assigning roles and milestones and agreeing on what content will be necessary for my particular project. For example and as you will discover, AMDiRE offers several ways of specifying functional requirements in a very complementary manner while we will never need to specify all of them (e.g. user stories versus use cases, more interaction-driven scenarios versus static feature models). Once this is set, we approach the AMDiRE model and specify the information in a top-down manner. This does not reflect so much a concrete process but more a step-wise approach towards the content specification following the previously introduced levels of abstraction. While a more hands-on introduction will be given along the following learning path and the toolbox, we visualise and show what the elementary steps are at an example:

1. Capture the domain (terminology, rules, “specialties”, background) - laying the foundation with the relevant information, for example operational workflows in a company capturing processes that shall be supported (partially) with the system under consideration. This context includes also

the external systems our system under consideration shall interact with.

2. Identify stakeholders - laying the foundation with the relevant stakeholders as sources for the requirements elicitation, including regulators giving useful information on which non-negotiable constraints to keep in mind but also user groups that shall interact with the system in the future.
3. Identify goals, conditions, and constraints - laying the foundation for better understanding why certain requirements are formulated the way they are formulated and how important they are (allowing also for a precise prioritisation of requirements)
4. Analysis of the “as-is” state - laying the foundation to understand strengths, weaknesses, and needs that results from the current situation (for example, reasons why an existing system shall be replaced and things that work well and that should be kept also in the future system)
5. Define the scope of the system - laying the foundation to later identify the borders for the problem and the system, and its environment; essentially capturing what is part of the system under consideration and what is part of neighbouring system (later captured in the AMDiRE system vision)
6. Analyse functional properties to satisfy goals - laying the foundation for functional requirements, for example by understanding how future user groups intend to interact with the system.
7. Analyse quality requirements - laying the foundation for non-functional, system-specific properties that need to be preserved (e.g. usability-, performance-, or security-related requirements)
8. Iteratively develop the system specification. In analogy to the constant back and forth between the problem space and the solution space introduced here, the elaboration and analysis of requirements is an iterative process (regardless of the chosen process model). In consequence, we may often go back to step 1 and continue our work until reaching the milestones (based on common agreements between the stakeholders and the formal acceptance).

Figure 1 visualizes the steps of the walk-through, mapping it to the corresponding layer (Context, Requirements, System) they occur. This figure can be mapped 1:1 to the AMDiRE artefacts and you will see later how those exemplary contents related to the detailed artefacts introduced in the subsequent learning path,

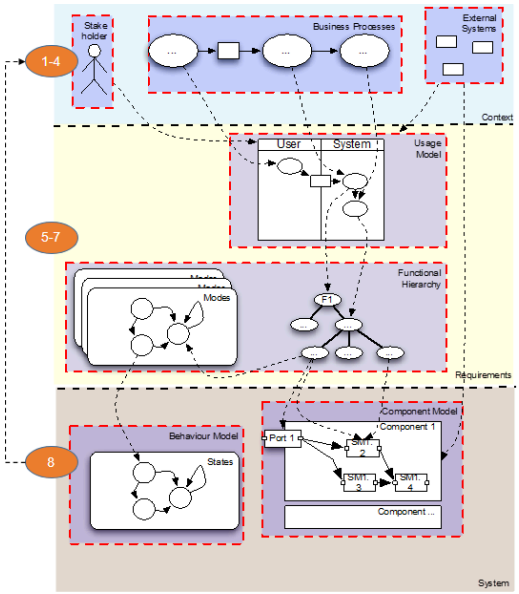


Figure 1

Key takeaway

- There is not a universal way of working in Requirements Engineering that would fit all project circumstances. Nevertheless, when using an artefact-centric approach as a blueprint, the artefact model helps identifying relevant contents in Requirements Engineering to elaborate.
- Creating a requirements specification and the operational context can and should be considered a top-down approach, regardless of surround process models.

Die europäischen Sprachen gehören zu ein und derselben Familie. Ihre getrennte Existenz ist ein Mythos. In Wissenschaft, Musik, Sport usw. verwendet Europa den gleichen Wortschatz. Die Sprachen unterscheiden sich nur in ihrer Grammatik, ihrer Aussprache und ihren gebräuchlichsten Wörtern. Jedem ist klar, warum eine neue gemeinsame Sprache wünschenswert wäre: Man könnte auf die Bezahlung teurer Übersetzer verzichten. Um dies zu erreichen, wären eine einheitliche Grammatik, eine einheitliche Aussprache und mehr gemeinsame Wörter notwendig.

- Auflistung
- Auflistung
- Auflistung

ID 1.2.1.6 - Operationalisation

Meta data

	EN	DE
Status	READY FOR REVIEW	
Title	Operationalisation	Operationalisierung
Tags	• Tag 1 • Tag 2	• Tag 1 • Tag 2
Estimated Duration	5 min	
Level	☐ BEGINNER ☐ INTERMEDIATE ☑ EXPERT	
External ressources	• Ressource 1 • Ressource 2	• Ressource 1 • Ressource 2
Dependencies		
Next Item	ID 1.2.2.1 - (delete?) AMDiRE Core Artefacts - LEARNING NUGGET	
Goals (Teaching perspective)	ID 1.2.1.1 - AMDiRE Core Components	
Content description (Teaching perspective)		
Content summary (Teaching perspective)		

Content data

EN DE

DESCRIPTION

In this learning nugget we will …

- give some insights into how AMDiRE is applied and customised in order to fit your project's need

Die Lerneinheit werden wir …

🎯 GOALS AND VALUE (User perspective)

In this learning nugget you will about …

- AMDiRE customisation principles
- AMDiRE tool support

In dieser Lerneinheit werden Sie …

- Ziel 1
- Ziel 2
- Ziel 3

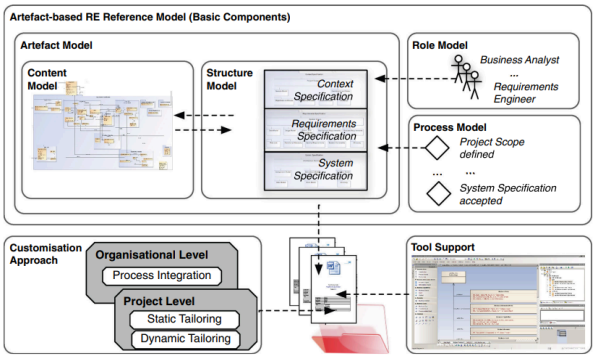
CONTENT

Motivation

Considering that not all projects have the same needs, flexibility, dynamics, and workflow, AMDiRE will always require some customisation in order to be usable in practice. Once AMDiRE is customised, it develops its full potential as it captures then both the necessary accuracy to overcome the major problems in Requirements Engineering (e.g. inconsistent requirements or requirements that are not testable) while offering the necessary pragmatism that we need to apply it in reality. Note that we document requirements not as a means in itself but rather as a means to an end and the same philosophy applies to AMDiRE.

AMDIRE Operationalisation

As seen previously in AMDiRE core components, the basic components of AMDiRE are supported by a customisation approach and a tool support to form the artefact materialisation.



The operationalisation of AMDiRE in a particular project context is done by essentially two means:

- Customisation approach where we agree in what parts to define how
- Tool Support where we agree on which tools to use

In the following, we explain the principles of both while referring also to the toolbox where we will give more practical advice.

Die Kakerlake

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein schreckliches Ungeziefer verwandelt. ...

Die Bedeutung

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein furchtbares Ungeziefer verwandelt. Er lag auf seinem panzerartigen Rücken, ...

Customisation Approach

AMDiRE is customised over two principle levels:

- **Organisational level:** At the organisational level, we consider the company specific customisation of AMDiRE, i.e. the process integration into an individual organisational reference software process model. Here, we map all elements to the elements of surrounding process models. For instance, if we follow a more agile approach, we associate the role in AMDiRE and the milestones to those we may have in our agile approach (e.g. sprint-related milestones that may then lead to one milestone in AMDiRE at some point). This step is usually done once and holds for many projects.
- **Project level:** The customisation at project level shall adapt AMDiRE to the particularise of one project following all the individual project characteristic. This should be done before entering a project, for instance, by removing contents from AMDiRE that shall not be used (e.g. elements that are more important to embedded systems when specifying requirements for enterprise resource planning, to name one example). Note, however, that sometimes new information is discovered during a project that requires further adoption during the project. For instance, we may discover that end users are not available for discussions and then may decide to not document use cases models but rather work with more abstract feature lists.

Tool support

Tool support for Requirements Engineering can be a never-ending topic. In essence, it is possible to work with various tools available on the market; for example:

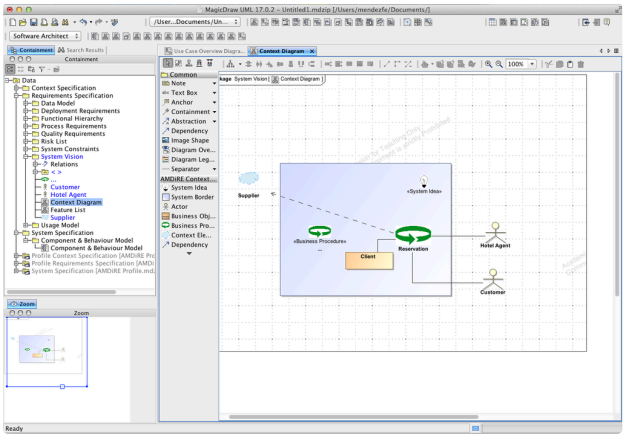
- Modelling-centric tools, such as MagicDraw where we can define an UML profile based not eh content model provided by AMDiRE (See the example below)
- More pragmatic tools that may as well serve the purpose of documenting requirements individually in natural language (and enriching them with additional information such s models), for example with ticket systems (e.g. Jira)

In the end, the tools do not matter as much as the questions of

1. what we can use considering the project circumstances (what is available and what can be used also in the eyes of the various stakeholders who need access to the elaborated requirements along the discussions of the requirements), and
2. to what extent can we make sure that the content outlined by AMDiRE can be captured following the rules defined in AMDiRE

Both are essentially also possible with document management systems and office software and using such documents makes very often sense from a pragmatic point of view. In the toolbox, we offer

templates that can be used with such software as we, too, follow a pragmatic perspective.



Key takeaway

Operationalization of the AMDiRE model according to the project needs occurs by:

1. Tailoring by customization over the organizational and project level.
2. AMDiRE can be operationalized through the application of templates and checklists or with Tool Support such as MagicDraw 4, Jira, spreadsheets.

Die europäischen Sprachen gehören zu ein und derselben Familie. Ihre getrennte Existenz ist ein Mythos. In Wissenschaft, Musik, Sport usw. verwendet Europa den gleichen Wortschatz. Die Sprachen unterscheiden sich nur in ihrer Grammatik, ihrer Aussprache und ihren gebräuchlichsten Wörtern. Jedem ist klar, warum eine neue gemeinsame Sprache wünschenswert wäre: Man könnte auf die Bezahlung teurer Übersetzer verzichten. Um dies zu erreichen, wären eine einheitliche Grammatik, eine einheitliche Aussprache und mehr gemeinsame Wörter notwendig.

- Auflistung
- Auflistung
- Auflistung

1.2.2.2.1 - Project Scope

Meta data

	EN	DE
Status	DRAFT	
Title	AMDiRE Core Artefacts	AMDiRE Kernartefakte
Tags	• Tag 1 • Tag 2	• Tag 1 • Tag 2
Estimated Duration	5 min	
Level	☐ BEGINNER ☐ INTERMEDIATE ☐ EXPERT	
External ressources	• Ressource 1 • Ressource 2	• Ressource 1 • Ressource 2
Dependencies		
Next Item	1.2.2.2.2 - Stakeholder Model	
Goals (Teaching perspective)		
Content description (Teaching perspective)		
Content summary (Teaching perspective)		

Content data

EN DE

DESCRIPTION

In this learning nugget we will elaborate on the project scope. Die Lerneinheit werden wir …

GOALS AND VALUE (User perspective)

In this learning nugget you will …

- what the project scope is, and
- why it is important to capture it.

In dieser Lerneinheit werden Sie …

- Ziel 1
- Ziel 2
- Ziel 3

CONTENT

Definition

The project scope in the Context Specification lays the foundation for all subsequent activities as it sets out the problem to be solved with a future system and the corresponding statement of intent, i.e. a conclusion of the objectives of a potentially resulting project. Here, we can also define measures of success for when the overall goals are achieved.

Purpose

The project scope is the basic common agreement of what shall be achieved in the project and serves several purposes, among them:

- Basis for contracts where the problem shall be elaborated (e.g. deficiencies of current systems to be replaced) and where the overall scope of the project shall be captured.
- Detection of moving targets or scope creeps.

Here, we may also set out to define the key responsibilities in a project; for instance, whose responsibility it is to steer the Requirements Engineering activities (customer? contractor?).

Since the project scope provides essential information on the overall direction of the project, ideally in the form of short statements, formally accepting it is also associated with the very first milestone of AMDiRE.

Concepts

- Problem description
- Statement of intent

Notation(s)

- Natural language text

Die Kakerlake

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein schreckliches Ungeziefer verwandelt. ...

Die Bedeutung

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein furchtbares Ungeziefer verwandelt. Er lag auf seinem panzerartigen Rücken, ...

Key takeaway

The project scope, although in tendency consisting of a short paragraph, is essential to lay the foundation for contractual agreements, assignments or responsibilities, and the detection of moving targets.

Die europäischen Sprachen gehören zu ein und derselben Familie. Ihre getrennte Existenz ist ein Mythos. In Wissenschaft, Musik, Sport usw. verwendet Europa den gleichen Wortschatz. Die Sprachen unterscheiden sich nur in ihrer Grammatik, ihrer

Aussprache und ihren gebräuchlichsten Wörtern. Jedem ist klar, warum eine neue gemeinsame Sprache wünschenswert wäre: Man könnte auf die Bezahlung teurer Übersetzer verzichten. Um dies zu erreichen, wären eine einheitliche Grammatik, eine einheitliche Aussprache und mehr gemeinsame Wörter notwendig.

- Auflistung
- Auflistung
- Auflistung

1.2.2.2.2 - Stakeholder Model

Meta data

	EN	DE
Status	DRAFT	
Title	Stakeholder Model	AMDiRE Kernartefakte
Tags	• Tag 1 • Tag 2	• Tag 1 • Tag 2
Estimated Duration	5 min	
Level	☐ BEGINNER ☐ INTERMEDIATE ☐ EXPERT	
External ressources	• Ressource 1 • Ressource 2	• Ressource 1 • Ressource 2
Dependencies		
Next Item	1.2.2.2.3 - Domain Model	
Goals (Teaching perspective)		
Content description (Teaching perspective)		
Content summary (Teaching perspective)		

Content data

EN DE

DESCRIPTION

In this learning nugget we will explore the specification of a fundamental aspect of Software Context: the Stakeholder Model.

Die Lerneinheit werden wir …

🎯 GOALS AND VALUE (User perspective)

In this learning nugget you will learn …

- what stakeholders are and what their relevance are
- what the concept is in the context specification

In dieser Lerneinheit werden Sie …

- Ziel 1
- Ziel 2
- Ziel 3

CONTENT

Definition

In essence, a stakeholder is an individual, a group, or an organization that has an interest in the software system being developed or that can influence its development.

It is important to note that "Interest in the system" should be understood broadly and that it does not necessarily imply an interest in the project's success. For example, many actors, such as governmental or regulatory entities, are stakeholders without a direct interest in the project's success. All they care about is that the system complies wiht the respective regulations.

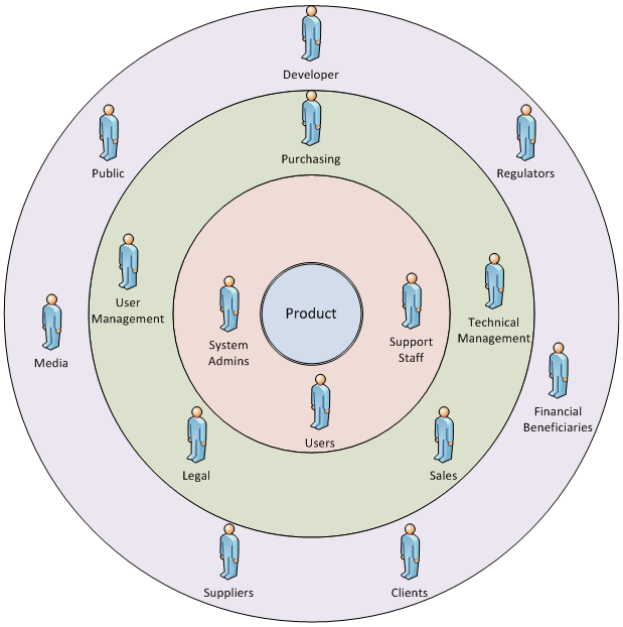
Stakeholder interests can be shaped by their relationship to the software system under development. These relationships can be effectively illustrated using a stakeholder onion diagram which displays, for illustrative purposes, stakeholders according to their (future) interaction with the developed software system. As that visualisation shows, there are stakeholders that will be directly interacting with the system, such as end users, and others that will maybe not interacting with the system. While all stakeholders constitute a potential source for requirements, the those interacting with the system will be used also in other content items to explore, for example, how this interaction shall take place, this, serving as a source for elaborating for functional and non-functional requirements.

Die Kakerlake

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein schreckliches Ungeziefer verwandelt. ...

Die Bedeutung

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein furchtbares Ungeziefer verwandelt. Er lag auf seinem panzerartigen Rücken, ...



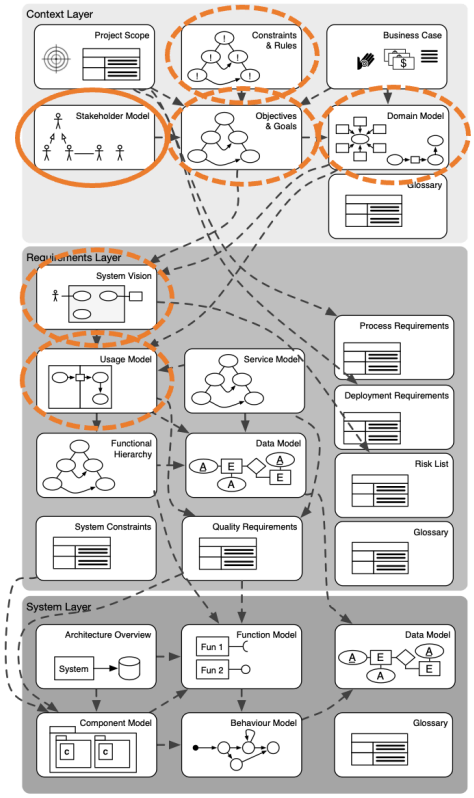
Purpose

Every project has different stakeholders, for example:

- Users and customers
- Legislators, certification bodies
- Compliance managers
- Marketing and product managers
- Domain experts
- Developers, architects, and testers
- Maintenance and operations teams

The names and types of stakeholders and what their relevance is can vary greatly depending on the type of software system (e.g., corporate information systems, embedded systems).

Different stakeholders have different interests and, thus, different requirements. Herewith, such interests and requirements need to be reconciled on the basis of the stakeholders' roles and their importance to the software project. For example, users are primarily interested in the quality of the software system, while product managers also care about sustainability of the business model and balancing costs and quality of the product.



Relationship between the stakeholder model and other content items

In any case, capturing the stakeholders explicitly is important as they are the source for various information with respect to the operational context of a system and requirements (e.g. end users documented later in the usage model as part of, for example, functionla requirements).

Views and view points of stakeholders

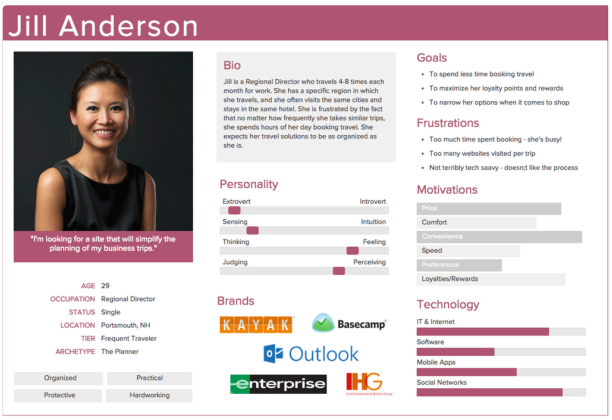
Different stakeholders might have different views on a system. A view in that sense represents the intention and the interest of the stakeholders interaction with the system. For instance, a user of a system has different intentions and interest in a system interaction, than its operator. For the software development, views facilitate communication and requirements elicitation.

In particular views are applied to:

- group and analyze different stakeholders' perspectives;
- elicit requirements from different stakeholder groups;
- tailor requirements elicitation techniques for the perspective stakeholder groups;
- identify and resolve conflicts and contradtions between stakehodler groups.

While views describe the explicit interest and intention of stakeholders, personas are used to represent each stakeholder group in detail (as a fictional individual) and are typically employed to describe and analyze user groups. They help to concretize

stakeholder profiles and structure empirically collected data about stakeholders.



Example of a Persona

The role of Stakeholder model

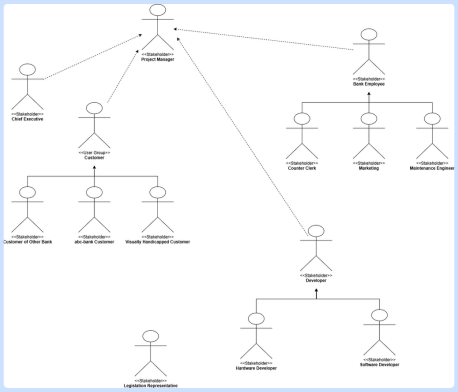
A stakeholder model forms the foundation for:

- Sources of requirements
- Goals justify (rationale) underlying requirements
- Potential end users (Actors of the system) in:
  - Domain Modeling (e.g. business processes)
  - Usage Model (use cases, scenarios)

Notation(s)

- UML actor hierarchy

Example



Stakeholder Model

- Tables to capture an overview of the relevant information

💡 Key takeaway

- Stakeholder models represent all stakeholder with an interest in or influence on the system under development
- A primary stakeholders interest is not necessarily the project success (e.g. regulators)
- Stakeholder models form the foundation for effective requirement elicitation
- The stakeholders idenfied in the stakeholder model can be grouped and analyzed via views and empirically concretized through personas

Die europäischen Sprachen gehören zu ein und derselben Familie. Ihre getrennte Existenz ist ein Mythos. In Wissenschaft, Musik, Sport usw. verwendet Europa den gleichen Wortschatz. Die Sprachen unterscheiden sich nur in ihrer Grammatik, ihrer Aussprache und ihren gebräuchlichsten Wörtern. Jedem ist klar, warum eine neue gemeinsame Sprache wünschenswert wäre: Man könnte auf die Bezahlung teurer Übersetzer verzichten. Um dies zu erreichen, wären eine einheitliche Grammatik, eine einheitliche Aussprache und mehr gemeinsame Wörter notwendig.

- Auflistung
- Auflistung
- Auflistung

1.2.2.2.3 - Domain Model

Meta data

	EN	DE
Status	DRAFT	
Title	Domain Model	AMDiRE Kernartefakte
Tags	• domain model	• Tag 1 • Tag 2
Estimated Duration	5 min	
Level	☐ BEGINNER ☑ INTERMEDIATE ☐ EXPERT	
External ressources		• Ressource 1 • Ressource 2
Dependencies		
Next Item	1.2.2.2.4 - Objectives and Goals	
Goals (Teaching perspective)	To understand what a domain model is, and what it captures.	
Content description (Teaching perspective)	• Domain model definition & purpose • Dynamic view • Static view	
Content summary (Teaching perspective)		

Content data

EN DE

DESCRIPTION

In this learning nugget, we will explore what purpose the domain model serves and which information it captures.

Die Lerneinheit werden wir …

🎯 GOALS AND VALUE (User perspective)

In this learning nugget you will …

- understand what a domain model is
- see the purpose of the domain model in requirements engineering
- explore the two perspectives of information captured in the domain model

In dieser Lerneinheit werden Sie …

- Ziel 1
- Ziel 2
- Ziel 3

📄 CONTENT

Definition

The domain model describes the understanding of the operational context (i.e. application domain) of a system under consideration. For that, it captures the most important workflows and types of objects in the context of the system.

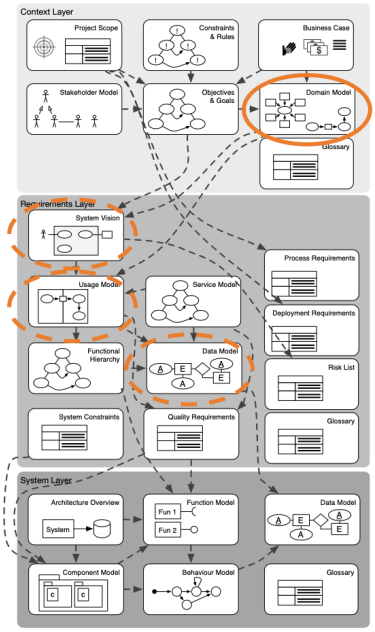
Purpose

The purpose of the domain model is to gain a sufficient understanding of the application domain and/or the operational environment (elementary concepts, terms) of the system under consideration. Capturing this understanding is among the first steps into a more profound analysis and when elaborating on the domain, we do not necessarily have the actual system in mind but, rather, aim at understanding the basics of the future environment of the system. This understanding may capture both typical processes/workflows and/or elementary concepts/objects. This serves then as a basis for the later definition of the system vision, functional requirements (in the form of a usage model), and the data model.

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein schreckliches Ungeziefer verwandelt. ...

Die Bedeutung

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein furchtbares Ungeziefer verwandelt. Er lag auf seinem panzerartigen Rücken, ...



Domain model and relationships to other AMDiRE content items.

The contents and the focus of the domain model may differ depending on type of system and the application domain. For example, when elaborating requirements for ERP or business information systems, the operational context is likely characterised heavily by workflows in the form of business processes; in contrast, when elaborating requirements for embedded systems, the operational domain is likely characterised more by information about hardware/software products, where those interacting entities might be more relevant.

Contents

To serve the different purposes, the domain model captures relevant information from two complementary perspectives and it remains the responsibility of the domain expert to capture the information deemed relevant for a sufficient understanding on the domain: a dynamic view and a static view. In case we aim at replacing a system with a new one, it is even more important to focus on an ideal view on the future to-be state in the domain rather than recapitulating exactly how it is at the moment (with all potential areas for improvement).

Dynamic View (Workflows / Process Models)

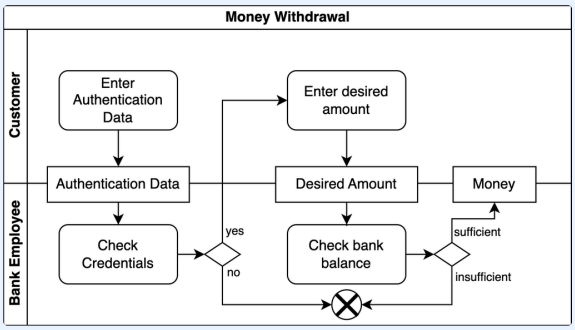
Through a dynamic view, we capture workflows and processes. A typical application of that view is a business process model for an information system. A business process model serves to understand the real-life workflows (business activities) which may be partially support by the system under consideration. In tune with the domain model being part of the context layer, it is important to

note that at this level, we do not analyse and model the specific workflows of the software system itself (e.g. software functionalities), but rather the real-life conceptual business activities of the domain with the goal of identifying tasks, dependencies, and involved roles/actors and included processes. To understand how the system under consideration has to support those business processes, we elaborate on the system vision and the usage models later in the requirement specification.

The dynamic view is most often conveyed via Business Process Model and Notation (BPMN) or via UML activity diagrams.

Example

In the following example, we depict the relevant workflows of two roles in money withdrawal process at a bank. The two relevant roles in this context are the customer and the bank employee. The identified business activities in the money withdrawal process range from entering the authentication data to checking the bank balance. Each such business activity also processes static domain objects like the authentication data, or the money.



UML Activity Diagram of the Money Withdrawal Process

Static View (Domain Concepts / Object Models)

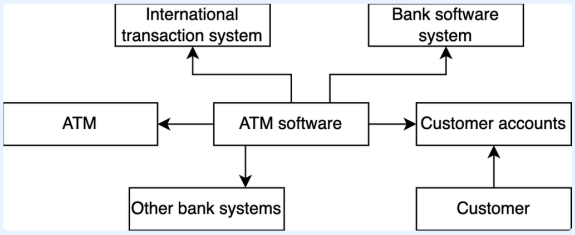
Through a static view, we focus more on the domain objects that interact or get processed by the workflows. As such, object models serve as conceptual models of elements in the problem space. Important to note as well here is that domain objects are not software objects, but a representation of real-life conceptual objects and important terminology used in the respective domain. Those conceptual objects can later be used to infer a data model for the system under consideration (if required).

Most often, the static view is represented as a UML class diagram.

Example

Following the example of an ATM in the banking area, the static view would capture the objects involved in the withdrawal of money at an ATM of a different bank. Here, we solely describe the real-life conceptual objects, such as

the ATM software, the international transaction software or the customer accounts and their relations, or what information a customer consists of.



Static View of Domain Objects in Money Withdrawal at ATM

Key takeaway

The domain model

- is a way to capture the the application domain and operational context of a system
- improves the understanding of the process and objects in the domain
- entails a dynamic view focused on workflows and processes
- captures relevant domain objects in the static view

Die europäischen Sprachen gehören zu ein und derselben Familie. Ihre getrennte Existenz ist ein Mythos. In Wissenschaft, Musik, Sport usw. verwendet Europa den gleichen Wortschatz. Die Sprachen unterscheiden sich nur in ihrer Grammatik, ihrer Aussprache und ihren gebräuchlichsten Wörtern. Jedem ist klar, warum eine neue gemeinsame Sprache wünschenswert wäre: Man könnte auf die Bezahlung teurer Übersetzer verzichten. Um dies zu erreichen, wären eine einheitliche Grammatik, eine einheitliche Aussprache und mehr gemeinsame Wörter notwendig.

- Auflistung
- Auflistung
- Auflistung

1.2.2.2.4 - Objectives and Goals

Meta data

	EN	DE
Status	DRAFT	
Title	Constraints and Goals	AMDiRE Kernartefakte
Tags	• Tag 1 • Tag 2	• Tag 1 • Tag 2
Estimated Duration	5 min	
Level	☐ BEGINNER ☐ INTERMEDIATE ☐ EXPERT	
External ressources	• Ressource 1 • Ressource 2	• Ressource 1 • Ressource 2
Dependencies		
Next Item	1.2.2.3.1 - System Vision	
Goals (Teaching perspective)		
Content description (Teaching perspective)		
Content summary (Teaching perspective)		

Content data

EN DE

DESCRIPTION

In this learning nugget, we will learn about Goals as defined in a context specification.

Die Lerneinheit werden wir …

🎯 GOALS AND VALUE (User perspective)

In this learning nugget, you will learn …

- what goals are and which types we typically distinguish
- how they relate to requirements
- the purpose of goal models

In dieser Lerneinheit werden Sie …

- Ziel 1
- Ziel 2
- Ziel 3

CONTENT

Definition

Generally speaking, a goal is a prescriptive statement of (stakeholder's) intent. It describes an abstract, future comprehensive state that a system must support or which should be achieved with the help of the system. Goals can be stated in relation to the system's operational context, and/or the process of system development.

It is important to distinguish goals from requirements. In particular goals are: abstract, high-level and often related to the system context, and they do not require a clear rationale, explanation i.e. are not measurable. Goals themselves are the rationale for future requirements as they help to understand the intended system and its requirements and they serve as basis for concrete requirements as we will see in later learning nuggets. Knowing the goals of the various stakeholders helps better understanding what the requirements are, making more informed assumptions, and prioritising requirements according to their relevance to support underlying goals.

Concepts

While literature offers a plethora of different approaches to specify and classify goals, we take a more simplified view and distinguish between three major types of goals (each relating to different specification contents later):

- **business goals** representing organization-level strategic goals related to the project;
- **usage goals** are goals of end users and usually serving as basis for functional requirements later on;
- **system goals** are goals related to qualitative properties of systems and usually directed at non-functional (quality) requirements later on.

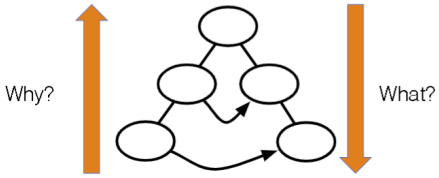
A goal model captures all relevant goals of stakeholders and their (hierarchical) relationships. The hierarchical structure of goals allows the refinement from the most abstract, high-level goals to more concrete goals, and goal abstraction from more concrete to abstract, high-level goals. Whether to capture goals explicitly in a goal model depends on whether goals need to be broken down to more detailed goals and whether we want to analyse them in more detail (e.g. to identify and resolve conflicts).

Natural text, graphsDie Kakerlake

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein schreckliches Ungeziefer verwandelt. ...

Die Bedeutung

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein furchtbares Ungeziefer verwandelt. Er lag auf seinem panzerartigen Rücken, ...



Goal refinement, that is breaking goals down to subgoals and so on, answers the question *What? or How?* (a goal might be achieved) and supports problem scoping and system scoping.

Goal abstraction, in turn, answers the question *Why?* and supports explanation, justification and prioritization of requirements as we identify underlying reasons for requirements in the forms of goals that justify the requirements.
Goal models and Stakeholder models are strongly related as they both serve as a basis for requiremnts prioritisation and early conflict resolution.

Goal Models

Purpose

- capture the goals of various stakeholders, their hierarchies and dependencies
- facilitate problem and system scoping
- support the reasoning and prioritization of requirements
- contributes to identifying and resolving conflicts

Notation(s)

- Natural text
- Graphs
- Goal-oriented requirements engineering (GORE) provide mathematical approaches and formalisations for conflicts resolutions and operational goals via logic expressions. GORE are often seen as burdensome for application in practice

---

Key takeaway

- Goals are abstract high-level statements describing a property of a system or that has to be fulfilled through the system
- We distinguish three goal types: business goals, usage goals, and system goals
- Goal models highlight the relations between goals and as such support in the analysis of conflicts and reasoning and prioritization of requirements

Die europäischen Sprachen gehören zu ein und derselben Familie. Ihre getrennte Existenz ist ein Mythos. In Wissenschaft, Musik, Sport usw. verwendet Europa den gleichen Wortschatz. Die Sprachen unterscheiden sich nur in ihrer Grammatik, ihrer Aussprache und ihren gebräuchlichsten Wörtern. Jedem ist klar, warum eine neue gemeinsame Sprache wünschenswert wäre: Man könnte auf die Bezahlung teurer Übersetzer verzichten. Um dies zu erreichen, wären eine einheitliche Grammatik, eine einheitliche Aussprache und mehr gemeinsame Wörter notwendig.

- Auflistung

- Auflistung
- Auflistung

1.2.2.3.1 - System Vision

Meta data

	EN	DE
Status	READY FOR REVIEW	
Title	AMDiRE Core Artefacts	AMDiRE Kernartefakte
Tags	• Tag 1 • Tag 2	• Tag 1 • Tag 2
Estimated Duration	5 min	
Level	☐ BEGINNER ☑ INTERMEDIATE ☐ EXPERT	
External ressources	• Ressource 1 • Ressource 2	• Ressource 1 • Ressource 2
Dependencies		
Next Item	1.2.2.3.2 - Functional Requirements	
Goals (Teaching perspective)		
Content description (Teaching perspective)		
Content summary (Teaching perspective)		

Content data

EN

DE

DESCRIPTION

In this learning nugget, we will focus on the system vision content item which is the first content item developed as a part of Requirements specification.

Die Lerneinheit werden wir …

🎯 GOALS AND VALUE (User perspective)

The goal of this learning nugget is to convey the

- definition and idea of system vision,
- the content and purposes of system vision, and
- notations used for system vision specification.

In dieser Lerneinheit werden Sie …

- Ziel 1
- Ziel 2
- Ziel 3

CONTENT

Definition

A **system vision** is a high-level description of the system under consideration in relation to its operational context.

A system vision provides a big picture of:

- connections between Context specification and Requirements specification by describing the scope of the software development project;
- the immediate context of the system relevant to the project;
- boundaries of the system;
- the most important use cases;
- key features of the system in interaction with the context.

In essence, the system vision describes what is part of the system under consideration and what isn't as well as the major (most relevant) high-level features.

Purpose

The main purposes of system vision is to:

- facilitate communication, agreement with stakeholders, and scoping of the project;
- specify and approve the main use cases to be implemented in the project.

We do so by capturing the system borders and identifying the most pressing high-level features in relation to external actors. Those

features can be, for example, high-level use cases. The content of those identified use cases is then described in the usage model when documenting the functional requirements.

Notations

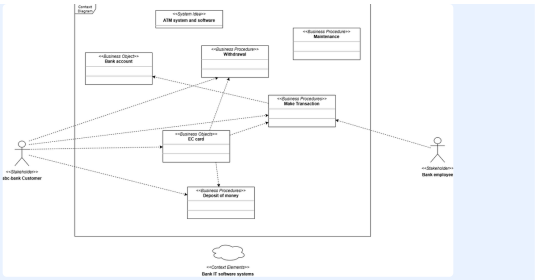
- System context diagram

Die Kakerlake

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein schreckliches Ungeziefer verwandelt. ...

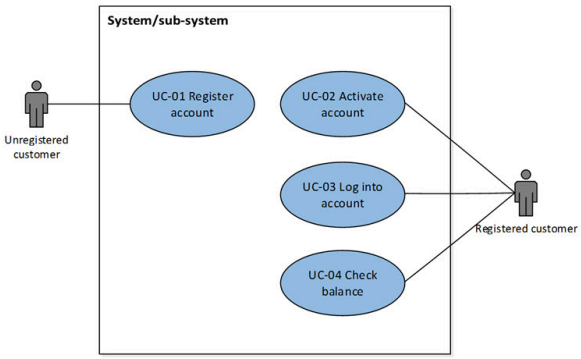
Die Bedeutung

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein furchtbares Ungeziefer verwandelt. Er lag auf seinem panzerartigen Rücken, ...



System Vision - Context Diagram

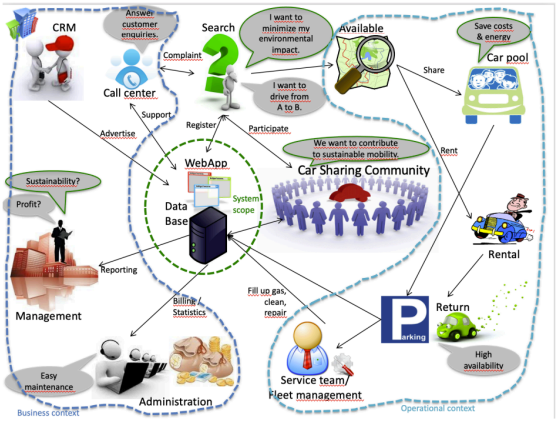
- UML Use case diagram (capturing relationships between different use cases without details of each use case)
- (Use case maps?)
- Rich pictures



use cave overview example

One aspect often underrated is that the system vision is one of the primary ways of communicating what is part of the system and what not (scoping) to all relevant stakeholders. As such, it needs to be understood by all stakeholders. A notation that is valuable in this context is given by rich pictures, explained next.

Rich pictures are a way to diagrammatically represent complex or even vague problems and/or interaction of the intended system with its complex operational environment. Rich pictures play a crucial role in learning about the problem and collaborating with stakeholders possessing domain knowledge and showing a “big picture”. The loose syntax of rich pictures is also its strength as it provides an opportunity for more flexible interaction with stakeholders and capturing complexities of the interaction of the system with its context. The example below shows a system in scope of car sharing with its complex operational background.



Rich picture example

Key takeaway

The system vision captures the high-level description of the system with its context to facilitate communication with stakeholders and scoping projects. Rich pictures can be effectively applied to specify system vision.

Die europäischen Sprachen gehören zu ein und derselben Familie. Ihre getrennte Existenz ist ein Mythos. In Wissenschaft, Musik, Sport usw. verwendet Europa den gleichen Wortschatz. Die Sprachen unterscheiden sich nur in ihrer Grammatik, ihrer Aussprache und ihren gebräuchlichsten Wörtern. Jedem ist klar, warum eine neue gemeinsame Sprache wünschenswert wäre: Man könnte auf die Bezahlung teurer Übersetzer verzichten. Um dies zu erreichen, wären eine einheitliche Grammatik, eine einheitliche Aussprache und mehr gemeinsame Wörter notwendig.

- Auflistung
- Auflistung
- Auflistung

1.2.2.3.2 - Functional Requirements

Meta data

	EN	DE
Status	DRAFT	
Title	AMDiRE Core Artefacts	AMDiRE Kernartefakte
Tags	• Tag 1 • Tag 2	• Tag 1 • Tag 2
Estimated Duration	5 min	
Level	☐ BEGINNER ☐ INTERMEDIATE ☐ EXPERT	
External ressources	• Ressource 1 • Ressource 2	• Ressource 1 • Ressource 2
Dependencies		
Next Item	ID 1.1.1.7 - Non-functional Requirements	
Goals (Teaching perspective)		
Content description (Teaching perspective)		
Content summary (Teaching perspective)		

Content data

EN DE

DESCRIPTION

In this learning nugget we will learn about the purpose and specification of functional requirements.

Die Lerneinheit werden wir …

🎯 GOALS AND VALUE (User perspective)

In this learning nugget you will …

- understand the functional requirements;
- learn about the purpose of functional requirements;
- learn about the main sources of information used for the specification of functional requirements;
- learn two main ways to specify functional requirements.

In dieser Lerneinheit werden Sie …

- Ziel 1
- Ziel 2
- Ziel 3

📄 CONTENT

Functional requirements

Functional requirements define the user-visible behavior of a system usually expressed in relation to the inputs a system receives and outputs it returns.
Functional requirements constitute the basic form of requirements and are supported by non-functional requirements expressing information about the qualities of a system that need to be implemented.

Purpose

Specification of functional requirements describes functions to be implemented and plays an essential role in assuring the verifiability of a system.

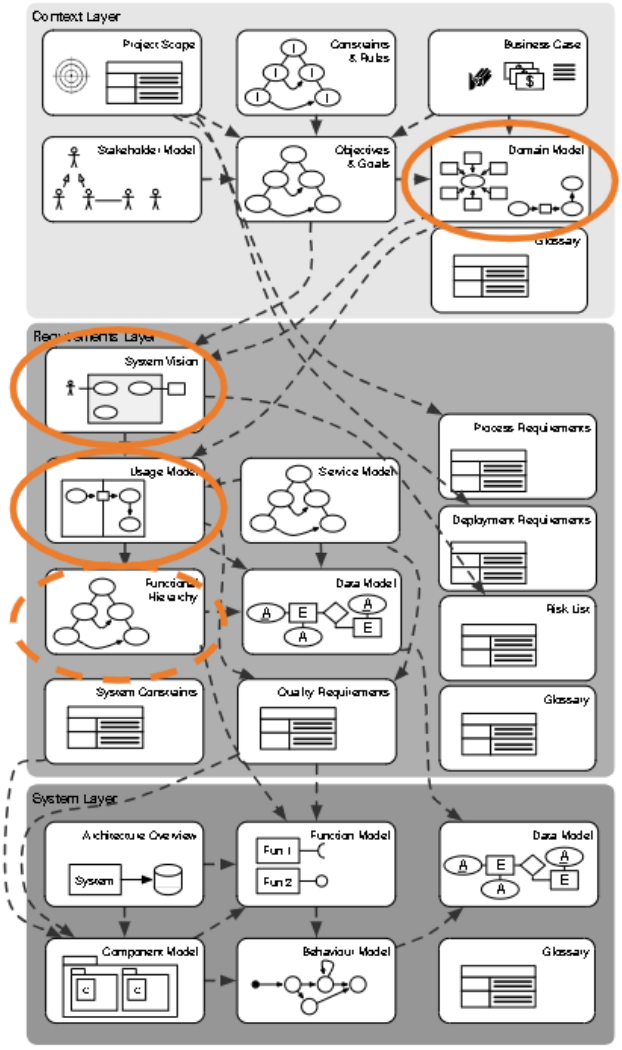
Functional requirements are related to different levels of abstraction at which behavior is specified in AMDiRE. In particular, at Context Layer Domain model and System vision content items are important sources of information about the required system behavior in the system operational context. At the Requirements layer Usage model content item provides a high-level overview of expected user-visible interactions.

Die Kakerlake

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein schreckliches Ungeziefer verwandelt. ...

Die Bedeutung

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein furchtbares Ungeziefer verwandelt. Er lag auf seinem panzerartigen Rücken, ...

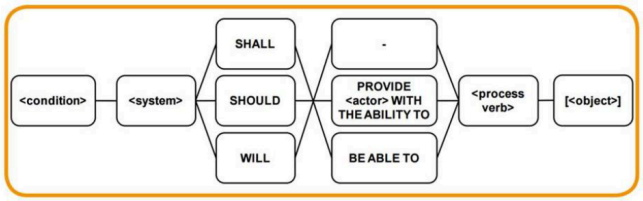


Basic specification of functional requirements

Typical natural language form for functional requirements is "When <input>/<condition>, a system must do <function> and return <output>". Functional requirements are usually specified at different levels of abstraction.

Functional requirements can be specified as atomic natural language statements describing isolated functions. In such form, functional requirements are consolidated or linked to describe the expected behavior of the system in its entirety.

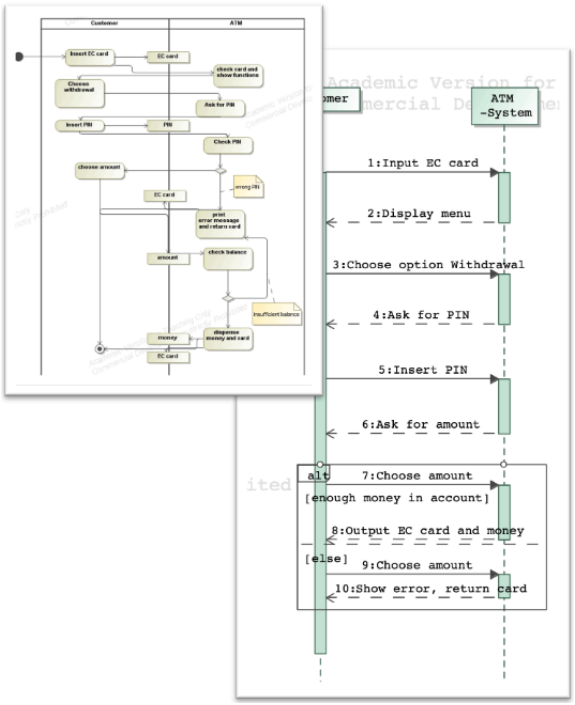
Specification of functional requirements in such form does not require any knowledge of notation and hence is suitable for non-technical addresses.



Specification of functional requirements as usage models

A more systematic alternative to the specification of functional requirements is the application of usage models that describe the system behavior from the point of view of the user ("black box"). Next, we consider two types of usage models used in AMDiRE which are:

- use scenarios describes an ordered set of interactions between the system and users or other systems. Notations used for the specification of use scenarios are structured text, interaction diagrams (activity diagrams, message sequence charts, etc.).

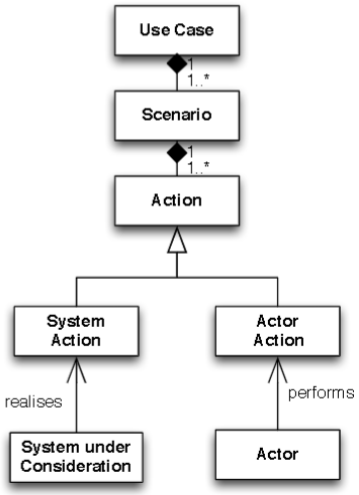


An example of use scenarios.

- use cases integrate all possible scenarios, which might occur when an actor tries to achieve a specific usage goal. Notations used for the specification of use cases are structured text, use case diagram.

USE CASE #	<the name is the goal as a short active verb phrase>		
Context of Use	<a longer statement of the context of use if needed>		
Scope	<what system is being considered black box under design>		
Level	<one of summary, primary task, subfunction>		
Primary Actor	<a role name for the primary actor, or a description>		
Stakeholder and Interests	Stakeholder	Interest	
	<stakeholder name>	<put here the interest of the stakeholder>	
	<stakeholder name>	<put here the interest of the stakeholder>	
Preconditions	<what we expect is already the state of the world>		
Minimal Guarantees	<the interests as protected on any exit>		
Success Guarantees	<the interests as satisfied on a successful ending>		
Trigger	<the action upon the system that starts the use case>		
Description	Step	Action	
	1	<put here the steps of the scenario from trigger to goal delivery and any cleanup after>	
	2	<...>	
	3		
Extensions	Step	Branching Action	
	1a	<condition causing branching> : <action or name of sub use case>	
Technology and Data Variations			
	1	<list of variations>	

An example of a use case.



Relationship between use cases, use scenarios, and actions (covered in functional requirements).

5 - PA1463 - Requirements Engineering - Functional Requirements.pptx (sharepoint.com)

Link to:

ID 1.1.1.6 - Functional Requirements

One morning, when Gregor Samsa woke from troubled dreams, he found himself transformed in his bed into a horrible vermin. He lay on his armour-like back, …

💡 Key takeaway

Functional requirements play an essential role in software projects. They are specified in the basis of the information specified on multiple levels of abstraction. Functional requirements can be specified as simple atomic natural language expression or in a form of usage models.

Die europäischen Sprachen gehören zu ein und derselben Familie. Ihre getrennte Existenz ist ein Mythos. In Wissenschaft, Musik, Sport usw. verwendet Europa den gleichen Wortschatz. Die Sprachen unterscheiden sich nur in ihrer Grammatik, ihrer Aussprache und ihren gebräuchlichsten Wörtern. Jedem ist klar, warum eine neue gemeinsame Sprache wünschenswert wäre: Man könnte auf die Bezahlung teurer Übersetzer verzichten. Um dies zu erreichen, wären eine einheitliche Grammatik, eine einheitliche Aussprache und mehr gemeinsame Wörter notwendig.

- Auflistung
- Auflistung
- Auflistung

1.2.2.3.3 - Non-Functional Requirements

Meta data

	EN	DE
Status	DRAFT	
Title	AMDiRE Core Artefacts	AMDiRE Kernartefakte
Tags	• Tag 1 • Tag 2	• Tag 1 • Tag 2
Estimated Duration	5 min	
Level	☐ BEGINNER ☐ INTERMEDIATE ☐ EXPERT	
External ressources	• Ressource 1 • Ressource 2	• Ressource 1 • Ressource 2
Dependencies		
Next Item	ID 1.2.2.4 - AMDiRE System Specification (Overview) - LEARNING PATH	
Goals (Teaching perspective)		
Content description (Teaching perspective)		
Content summary (Teaching perspective)		

Content data

EN DE

DESCRIPTION

In this learning nugget we will …

Die Lerneinheit werden wir …

GOALS AND VALUE (User perspective)

In this learning nugget you will …

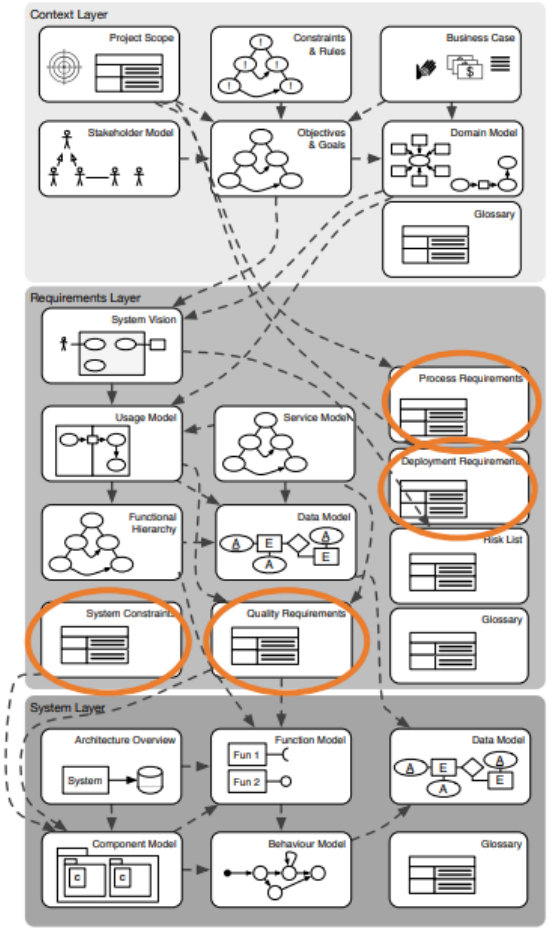
- Goal 1
- Goal 2
- Goal 3

In dieser Lerneinheit werden Sie …

- Ziel 1
- Ziel 2
- Ziel 3

CONTENT

The cockroach



6 - PA1463 - Requirements Engineering - Non-Functional Requirements.pptx (sharepoint.com)

The meaning

One morning, when Gregor Samsa woke from troubled dreams, he found himself transformed in his bed into a horrible vermin. He lay on his armour-like back, …

Die Kakerlake

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein schreckliches Ungeziefer verwandelt. ...

Die Bedeutung

Eines Morgens, als Gregor Samsa aus unruhigen Träumen erwachte, fand er sich in seinem Bett in ein furchtbares Ungeziefer verwandelt. Er lag auf seinem panzerartigen Rücken, ...

💡 Key takeaway

The European languages are members of the same family. Their separate existence is a myth. For science, music, sport, etc, Europe uses the same vocabulary. The languages only differ in their grammar, their pronunciation and their most common words. Everyone realizes why a new common language would be desirable: one could refuse to pay expensive translators. To achieve this, it would be necessary to have uniform grammar, pronunciation and more common words.

- List item
- List item
- List item

Die europäischen Sprachen gehören zu ein und derselben Familie. Ihre getrennte Existenz ist ein Mythos. In Wissenschaft, Musik, Sport usw. verwendet Europa den gleichen Wortschatz. Die Sprachen unterscheiden sich nur in ihrer Grammatik, ihrer Aussprache und ihren gebräuchlichsten Wörtern. Jedem ist klar, warum eine neue gemeinsame Sprache wünschenswert wäre: Man könnte auf die Bezahlung teurer Übersetzer verzichten. Um dies zu erreichen, wären eine einheitliche Grammatik, eine einheitliche Aussprache und mehr gemeinsame Wörter notwendig.

- Auflistung
- Auflistung
- Auflistung

GLOSSARY

Word	Description
Natural text	Refers to the way humans communicate and express themselves in written words.
Team composition	Refers to the overall characteristics, demographics, and traits of the team
SCO	Self-checkout
Iteration	A single, timeboxed event that produces rapid feedback to ensure the product continuously delivers business value.
Stakeholder	Anyone having any type of relation/interest in the project is known as stakeholder.
UML actor hierarchy	Specifies a role played by a user or any other system that interacts with the subject.
Multidisciplinary team	Are teams consisting of individuals drawn from different disciplines who come together to achieve a common goal.
Benchmarking reports	A benchmark report is a type of business report that shows how one's product, performance, or company compares to similar products or companies.
User story	A user story is an informal, general explanation of a software feature written from the perspective of the end user or customer.
Natural Language	Refers to the way humans communicate and express themselves through spoken or written words.
Prototype	A prototype is an early sample, model, or release of a product built to test a concept or process.
Software-intensive systems	It can be defined as any system where software influences, to a large extent, the design, construction, deployment, and evolution of the system as a whole.
Wireframe	Wireframe is a two-dimensional skeletal outline of a webpage or app.
User satisfaction	Measures of how well a company's products and overall customer experience meet customer expectations.
Rich picture	A diagrammatic way of relating your own experiences and perceptions to a given problem

	situation through the identification and linking of a series of concepts
UML	Is a general-purpose visual modeling language that is intended to provide a standard way to visualize the design of a system
Flowchart	A diagram of the sequence of movements or actions of people or things involved in a complex system or activity.
BPMN	Business Process Model and Notation (BPMN) is a graphical representation for specifying business processes in a business process model.
Small Enterprises (according to European guidelines)	Number of employees: 0-50 Revenue: Low Target Market: Focused Market influence: Local
Medium Enterprises (according to European guidelines)	Number of employees: 50-250 Revenue: Moderate Target Market: Diverse Market: Regional
Large Enterprise (according to European guidelines)	Number of employees: >=250 Revenue: High Target Market: Diverse Market influence: Global

Natürlicher Text	Bezieht sich auf die Art und Weise, wie Menschen kommunizieren und sich in geschriebenen Worten ausdrücken.
Zusammensetzung des Teams	Bezieht sich auf die allgemeinen Merkmale, die Demografie und die Charaktereigenschaften des Teams
SCO	Selbstbedienungs-Kasse
Iteration	Ein einzelnes, zeitlich begrenztes Ereignis, das schnelles Feedback liefert, um sicherzustellen, dass das Produkt kontinuierlich einen geschäftlichen Nutzen bringt.
Interessensvertreter	Jeder, der in irgendeiner Form mit dem Projekt in Verbindung steht oder ein Interesse daran hat, wird als Stakeholder bezeichnet.
UML-Akteurshierarchie	Gibt eine Rolle an, die ein Benutzer oder ein anderes System spielt, das mit dem Thema interagiert.

Multidisziplinäres Team	Es handelt sich um Teams, die sich aus Personen unterschiedlicher Fachrichtungen zusammensetzen, um ein gemeinsames Ziel zu erreichen.
Benchmarking-Berichte	Ein Benchmark-Bericht ist eine Art Geschäftsbericht, der zeigt, wie das eigene Produkt, die eigene Leistung oder das eigene Unternehmen im Vergleich zu ähnlichen Produkten oder Unternehmen abschneidet.
Anwenderbericht	Eine User Story ist eine informelle, allgemeine Erklärung einer Softwarefunktion, die aus der Perspektive des Endbenutzers oder Kunden geschrieben wird.
Natürliche Sprache	Bezieht sich auf die Art und Weise, wie Menschen kommunizieren und sich durch gesprochene oder geschriebene Worte ausdrücken.
Prototyp	Ein Prototyp ist ein frühes Muster, Modell oder eine frühe Version eines Produkts, das zum Testen eines Konzepts oder Prozesses gebaut wird.
Software-intensive Systeme	Es kann als jedes System definiert werden, bei dem die Software in hohem Maße den Entwurf, die Konstruktion, den Einsatz und die Entwicklung des Systems als Ganzes beeinflusst.
Drahtgitter	Ein Wireframe ist ein zweidimensionales Grundgerüst einer Webseite oder Anwendung.
Zufriedenheit der Nutzer	Misst, wie gut die Produkte eines Unternehmens und das allgemeine Kundenerlebnis die Erwartungen der Kunden erfüllen.
Reiches Bild	Eine schematische Art und Weise, die eigenen Erfahrungen und Wahrnehmungen mit einer gegebenen Problemsituation in Beziehung zu setzen, indem eine Reihe von Konzepten identifiziert und miteinander verknüpft wird
UML	ist eine allgemeine visuelle Modellierungssprache, die eine Standardmethode zur Visualisierung des Entwurfs eines Systems bieten soll
Flussdiagramm	Ein Diagramm der Abfolge von Bewegungen oder Aktionen von Menschen oder Dingen, die an einem komplexen System oder einer Aktivität beteiligt sind.
BPMN	Business Process Model and Notation (BPMN) ist eine grafische Darstellung zur Spezifikation von Geschäftsprozessen in einem Geschäftsprozessmodell.
Kleine Unternehmen (gemäß den europäischen Leitlinien)	Anzahl der Mitarbeiter: 0-50 Einkünfte: Niedrig

	Zielmarkt: Fokussiert Markteinfluss: Lokal
Mittlere Unternehmen (gemäß den europäischen Leitlinien)	Anzahl der Mitarbeiter: 50-250 Einkünfte: Mäßig Zielmarkt: Vielfältig Markt: Regional
Großunternehmen (gemäß den europäischen Leitlinien)	Anzahl der Beschäftigten: >=250 Einkünfte: Hoch Zielmarkt: Vielfältig Markteinfluss: Weltweit

FAQ