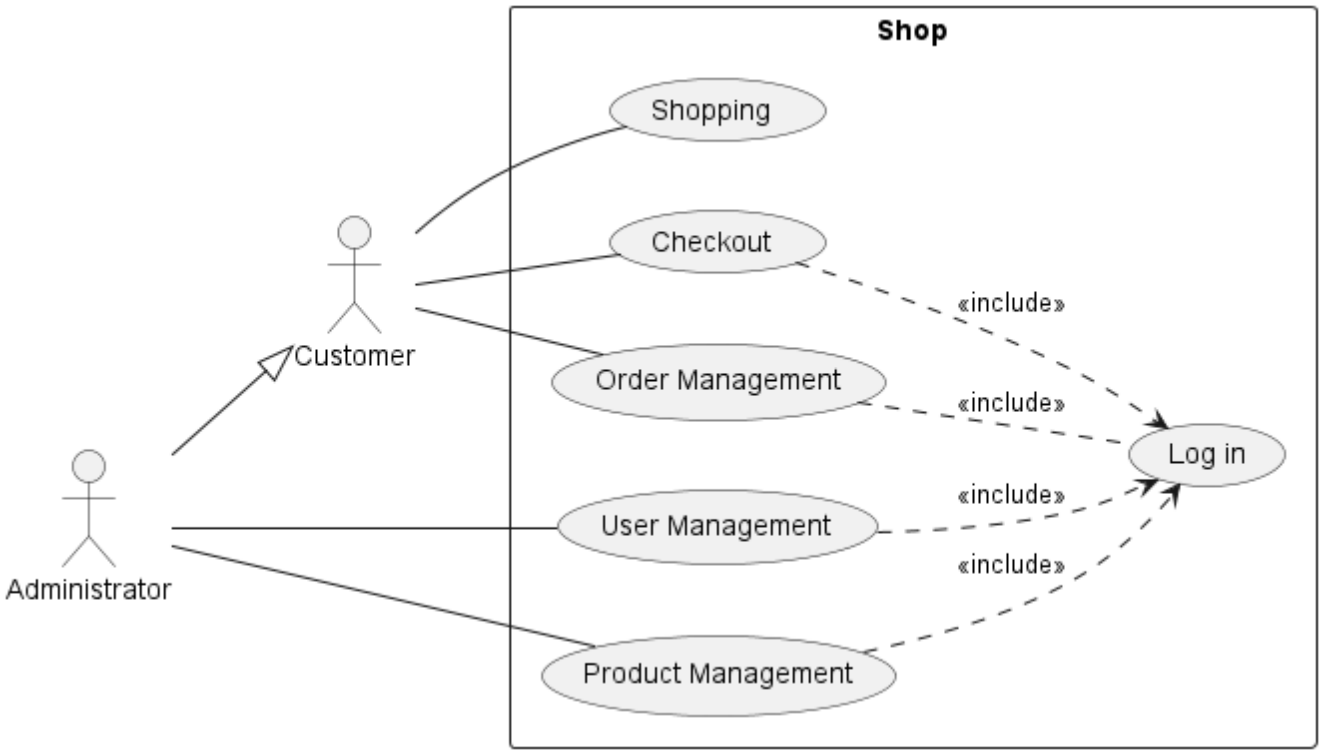
Table of Contents

1. Introduction and Goals

The eCommerce platform should aim to increase market share and attract new customers by providing a user-friendly shopping experience.

The primary business goal is to generate revenue through online sales. Therefore, the platform should be designed to encourage purchases.

1.1. Requirements Overview



Product Management

The platform must support comprehensive product management capabilities, including catalog management, inventory tracking, and product variations (e.g., sizes, colors).

User Accounts and Authentication

Users should be able to create accounts, log in securely, and manage their profiles, preferences, and orders.

Shopping Cart and Checkout

A seamless shopping cart and checkout process are essential for a positive user experience. Users should be able to add items to their cart, review their order, and complete the checkout process efficiently.

Payment Processing

Secure payment processing is critical. The platform should support multiple payment methods, such as credit/debit cards, digital wallets, and alternative payment options, while complying with PCI DSS standards.

Order Management

The platform must have robust order management capabilities, including order tracking, fulfillment, and returns processing.

1.2. Quality Goals

Priority	Quality-Goal	Scenario
1	Maintainability	Structure the codebase and architecture to be modular, well-documented, and easy to maintain, allowing for efficient updates, bug fixes, and enhancements.
1	Extensibility	Enable the platform to be easily extended with new features, functionalities, or integrations without requiring significant changes to the existing codebase.
1	Security	Implement robust security measures to protect sensitive customer data, prevent unauthorized access, and ensure compliance with industry regulations (e.g., GDPR, PCI DSS).
1	Interoperability	Ensure compatibility with third-party services, APIs, and integrations (e.g., payment gateways, shipping providers) to facilitate seamless transactions and order fulfillment.
2	Usability	Provide intuitive navigation, search functionality, and a user-friendly interface to enhance the shopping experience and encourage conversions.
2	Flexibility	Provide configurable options and customization capabilities to adapt to changing business requirements, market trends, and customer preferences.
2	Testability	Ensure that the platform can be thoroughly tested with unit tests, integration tests, and end-to-end tests to verify functionality and detect regressions.
3	Cost	Minimize operating costs associated with hosting, maintenance, and development while maximizing the value delivered to customers and stakeholders.
3	Localization	Support multiple languages.

1.3. Stakeholders

Role/Name	Contact	Expectations
Developer	[Contact information]	Should understand the architecture to effectively implement features and modules. Need to collaborate with architects and other team members to ensure adherence to architectural guidelines.
Architect	[Contact information]	Should define the system architecture, ensuring it meets business requirements and quality attributes. Need to communicate architectural decisions effectively to developers and stakeholders.

Role/Name	Contact	Expectations
Product Owner	[Contact information]	Should understand the architecture to align technical decisions with business objectives. Need to provide input on feature priorities and requirements based on architectural constraints.
Tester	[Contact information]	Should understand the architecture to design and execute appropriate tests. Need to identify areas of risk and ensure comprehensive test coverage based on architectural dependencies.
DevOps	[Contact information]	Should understand the architecture to deploy, monitor, and maintain the system effectively. Need to configure deployment pipelines and infrastructure based on architectural requirements.
Project Manager	[Contact information]	Should understand the architecture to plan and monitor project activities effectively. Need to ensure project timelines and resources align with architectural constraints.
Business Analyst	[Contact information]	Should understand the architecture to elicit and document requirements effectively. Need to ensure that business requirements are aligned with architectural decisions and constraints.

2. Architecture Constraints

This section outlines requirements that impose limitations on the freedom of design and implementation decisions within the development process.

2.1. Technical Constraints

	Constraint	Background and / or motivation
Software and Programming Constraints		
TC1	Implementation in C#	The application must be developed using the C# programming language to ensure compatibility with the .NET 8 environment.
TC2	Third-party Software Licensing	All third-party software utilized in the application must be available under an open-source license compatible with the project's requirements. Additionally, it should be installable via a package manager to facilitate seamless integration and updates.
Operating System Constraints		
TC3	OS-Independent Development	The application should be designed to be platform-independent, ensuring compatibility with major operating systems such as Mac OS X, Linux, and Windows. This flexibility allows for wider deployment options and increased accessibility for users.
TC4	Linux Server Deployment	To accommodate deployment in various hosting environments, the application must be capable of being deployed on Linux-based servers using standard deployment procedures. This ensures compatibility with common server configurations and facilitates seamless deployment processes.
Hardware Constraints		
TC5	Memory Efficiency	Given potential resource constraints in shared or cloud-based hosting environments, the application should prioritize memory efficiency. Optimizing memory usage reduces operational costs and improves overall performance, making the application more scalable and cost-effective.

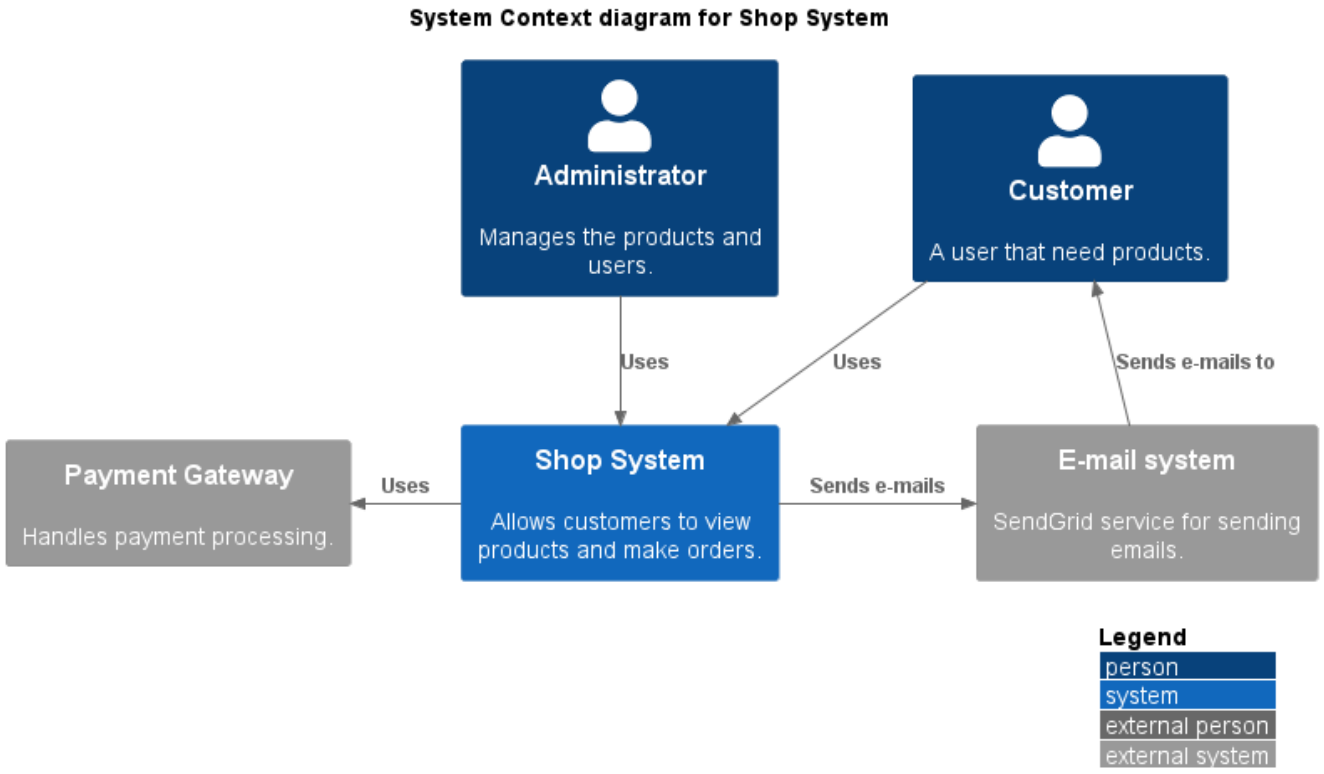
2.2. Conventions

	Conventions	Background and / or motivation
C1	Architecture documentation	Structure based on the english arc42-Template in version 8.2

	Conventions	Background and / or motivation
C2	Language	English. The project targets an international audience, so English should be used throughout the whole project.
C3	Naming conventions	The project uses the Code-style name rules provided by micorosoft. They are checked and enforced with .editorconfig.

3. System Scope and Context

This chapter provides an overview of the application's environment and context, detailing its users and dependencies on other systems.



Neighbor	Description
Customer	Represents individuals who use the e-commerce application to browse products and make purchases.
Administrator	Refers to authorized personnel responsible for managing the e-commerce application.
E-mail system	Represents the SendGrid service used for sending communication and notifications via email.
Payment Gateway	Represents the external service responsible for handling payment processing for transactions.

4. Solution Strategy

The system architecture is meticulously crafted to align with key quality goals and organizational requirements, with a focus on modularity, cleanliness, and modern frontend technology.

Modular Architecture

The system is designed using a modular architecture approach, allowing for the decomposition of the application into reusable and independent modules. Each module follows clean architecture principles, ensuring clear separation of concerns and maintainability.

Clean Architecture Per Module

Within each module, clean architecture principles are applied rigorously. This architectural style emphasizes the separation of concerns, with distinct layers for business logic, application services, and infrastructure. This promotes code maintainability, testability, and flexibility.

Blazor for Frontend

The frontend of the application leverages Blazor, a modern web framework that allows for the development of interactive web UIs using C# and .NET. Blazor enables the creation of rich, client-side web applications with minimal JavaScript, streamlining the development process and enhancing code consistency.

Technology Stack

The technology stack is carefully chosen to support the architectural decisions and meet project requirements. This includes the use of .NET Core for backend development, Blazor for frontend development, and other complementary technologies for database management, authentication, and integration.

Quality Goal Achievement

The architecture is shaped with a strong focus on achieving key quality goals such as maintainability, scalability, security, and usability. Each architectural decision is made with these goals in mind, ensuring that the final system meets the highest standards of reliability, performance, and user experience.

Organizational Alignment

The architectural decisions are aligned with organizational objectives and development processes. Agile methodologies are adopted to facilitate iterative development and collaboration, while resource allocation and team structure are optimized to support the modular architecture approach.

5. Building Block View

Content

The building block view shows the static decomposition of the system into building blocks (modules, components, subsystems, classes, interfaces, packages, libraries, frameworks, layers, partitions, tiers, functions, macros, operations, data structures, …) as well as their dependencies (relationships, associations, …)

This view is mandatory for every architecture documentation. In analogy to a house this is the *floor plan*.

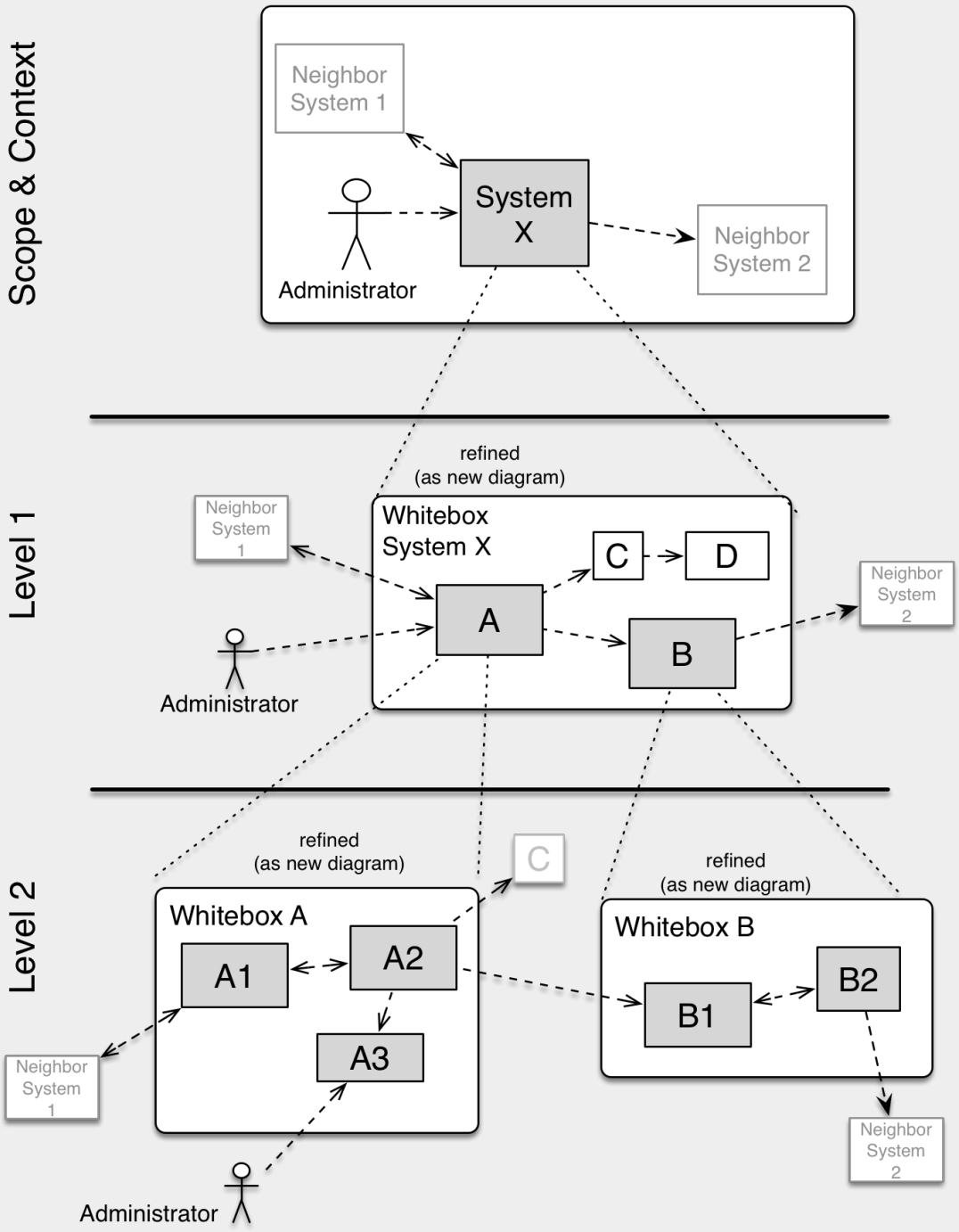
Motivation

Maintain an overview of your source code by making its structure understandable through abstraction.

This allows you to communicate with your stakeholder on an abstract level without disclosing implementation details.

Form

The building block view is a hierarchical collection of black boxes and white boxes (see figure below) and their descriptions.



Level 1 is the white box description of the overall system together with black box descriptions of all contained building blocks.

Level 2 zooms into some building blocks of level 1. Thus it contains the white box description of selected building blocks of level 1, together with black box descriptions of their internal building blocks.

Level 3 zooms into selected building blocks of level 2, and so on.

Further Information

See Building Block View in the arc42 documentation.

5.1. Whitebox Overall System

Here you describe the decomposition of the overall system using the following white box template. It contains

- an overview diagram
- a motivation for the decomposition
- black box descriptions of the contained building blocks. For these we offer you alternatives:
  - use *one* table for a short and pragmatic overview of all contained building blocks and their interfaces
  - use a list of black box descriptions of the building blocks according to the black box template (see below). Depending on your choice of tool this list could be sub-chapters (in text files), sub-pages (in a Wiki) or nested elements (in a modeling tool).
- (optional:) important interfaces, that are not explained in the black box templates of a building block, but are very important for understanding the white box. Since there are so many ways to specify interfaces why do not provide a specific template for them. In the worst case you have to specify and describe syntax, semantics, protocols, error handling, restrictions, versions, qualities, necessary compatibilities and many things more. In the best case you will get away with examples or simple signatures.

*<**Overview Diagram**>*

Motivation

<text explanation>

Contained Building Blocks

<Description of contained building block (black boxes)>

Important Interfaces

<Description of important interfaces>

Insert your explanations of black boxes from level 1:

If you use tabular form you will only describe your black boxes with name and responsibility according to the following schema:

Name	**Responsibility**
<black box 1>	*<Text>*
<black box 2>	*<Text>*

If you use a list of black box descriptions then you fill in a separate black box template for every important building block . Its headline is the name of the black box.

5.1.1. <Name black box 1>

Here you describe <black box 1> according the the following black box template:

- Purpose/Responsibility
- Interface(s), when they are not extracted as separate paragraphs. This interfaces may include qualities and performance characteristics.
- (Optional) Quality-/Performance characteristics of the black box, e.g.availability, run time behavior, ….
- (Optional) directory/file location
- (Optional) Fulfilled requirements (if you need traceability to requirements).
- (Optional) Open issues/problems/risks

<Purpose/Responsibility>

<Interface(s)>

<(Optional) Quality/Performance Characteristics>

<(Optional) Directory/File Location>

<(Optional) Fulfilled Requirements>

<(optional) Open Issues/Problems/Risks>

5.1.2. <Name black box 2>

<black box template>

5.1.3. <Name black box n>

<black box template>

5.1.4. <Name interface 1>

…

5.1.5. <Name interface m>

5.2. Level 2

Here you can specify the inner structure of (some) building blocks from level 1 as white boxes.

You have to decide which building blocks of your system are important enough to justify such

a detailed description. Please prefer relevance over completeness. Specify important, surprising, risky, complex or volatile building blocks. Leave out normal, simple, boring or standardized parts of your system

5.2.1. White Box <*building block 1*>

...describes the internal structure of *building block 1*.

<*white box template*>

5.2.2. White Box <*building block 2*>

<*white box template*>

...

5.2.3. White Box <*building block m*>

<*white box template*>

5.3. Level 3

Here you can specify the inner structure of (some) building blocks from level 2 as white boxes.

When you need more detailed levels of your architecture please copy this part of arc42 for additional levels.

5.3.1. White Box <_building block x.1_>

Specifies the internal structure of *building block x.1*.

<*white box template*>

5.3.2. White Box <_building block x.2_>

<*white box template*>

5.3.3. White Box <_building block y.1_>

<*white box template*>

6. Runtime View

Contents

The runtime view describes concrete behavior and interactions of the system's building blocks in form of scenarios from the following areas:

- important use cases or features: how do building blocks execute them?
- interactions at critical external interfaces: how do building blocks cooperate with users and neighboring systems?
- operation and administration: launch, start-up, stop
- error and exception scenarios

Remark: The main criterion for the choice of possible scenarios (sequences, workflows) is their **architectural relevance**. It is **not** important to describe a large number of scenarios. You should rather document a representative selection.

Motivation

You should understand how (instances of) building blocks of your system perform their job and communicate at runtime. You will mainly capture scenarios in your documentation to communicate your architecture to stakeholders that are less willing or able to read and understand the static models (building block view, deployment view).

Form

There are many notations for describing scenarios, e.g.

- numbered list of steps (in natural language)
- activity diagrams or flow charts
- sequence diagrams
- BPMN or EPCs (event process chains)
- state machines
- …

Further Information

See Runtime View in the arc42 documentation.

6.1. <Runtime Scenario 1>

- *<insert runtime diagram or textual description of the scenario>*
- *<insert description of the notable aspects of the interactions between the building block instances depicted in this diagram.>*

6.2. <Runtime Scenario 2>

6.3. …

6.4. <Runtime Scenario n>

7. Deployment View

Content

The deployment view describes:

1. technical infrastructure used to execute your system, with infrastructure elements like geographical locations, environments, computers, processors, channels and net topologies as well as other infrastructure elements and
2. mapping of (software) building blocks to that infrastructure elements.

Often systems are executed in different environments, e.g. development environment, test environment, production environment. In such cases you should document all relevant environments.

Especially document a deployment view if your software is executed as distributed system with more than one computer, processor, server or container or when you design and construct your own hardware processors and chips.

From a software perspective it is sufficient to capture only those elements of an infrastructure that are needed to show a deployment of your building blocks. Hardware architects can go beyond that and describe an infrastructure to any level of detail they need to capture.

Motivation

Software does not run without hardware. This underlying infrastructure can and will influence a system and/or some cross-cutting concepts. Therefore, there is a need to know the infrastructure.

Form

Maybe a highest level deployment diagram is already contained in section 3.2. as technical context with your own infrastructure as ONE black box. In this section one can zoom into this black box using additional deployment diagrams:

- UML offers deployment diagrams to express that view. Use it, probably with nested diagrams, when your infrastructure is more complex.
- When your (hardware) stakeholders prefer other kinds of diagrams rather than a deployment diagram, let them use any kind that is able to show nodes and channels of the infrastructure.

Further Information

See Deployment View in the arc42 documentation.

7.1. Infrastructure Level 1

Describe (usually in a combination of diagrams, tables, and text):

- distribution of a system to multiple locations, environments, computers, processors, .., as well as physical connections between them
- important justifications or motivations for this deployment structure
- quality and/or performance features of this infrastructure
- mapping of software artifacts to elements of this infrastructure

For multiple environments or alternative deployments please copy and adapt this section of arc42 for all relevant environments.

<*Overview Diagram*>

Motivation

<explanation in text form>

Quality and/or Performance Features

<explanation in text form>

Mapping of Building Blocks to Infrastructure

<description of the mapping>

7.2. Infrastructure Level 2

Here you can include the internal structure of (some) infrastructure elements from level 1.

Please copy the structure from level 1 for each selected element.

7.2.1. <*Infrastructure Element 1*>

<diagram + explanation>

7.2.2. <*Infrastructure Element 2*>

<diagram + explanation>

...

7.2.3. <*Infrastructure Element n*>

<diagram + explanation>

8. Cross-cutting Concepts

Content

This section describes overall, principal regulations and solution ideas that are relevant in multiple parts (= cross-cutting) of your system. Such concepts are often related to multiple building blocks. They can include many different topics, such as

- models, especially domain models
- architecture or design patterns
- rules for using specific technology
- principal, often technical decisions of an overarching (= cross-cutting) nature
- implementation rules

Motivation

Concepts form the basis for *conceptual integrity* (consistency, homogeneity) of the architecture. Thus, they are an important contribution to achieve inner qualities of your system.

Some of these concepts cannot be assigned to individual building blocks, e.g. security or safety.

Form

The form can be varied:

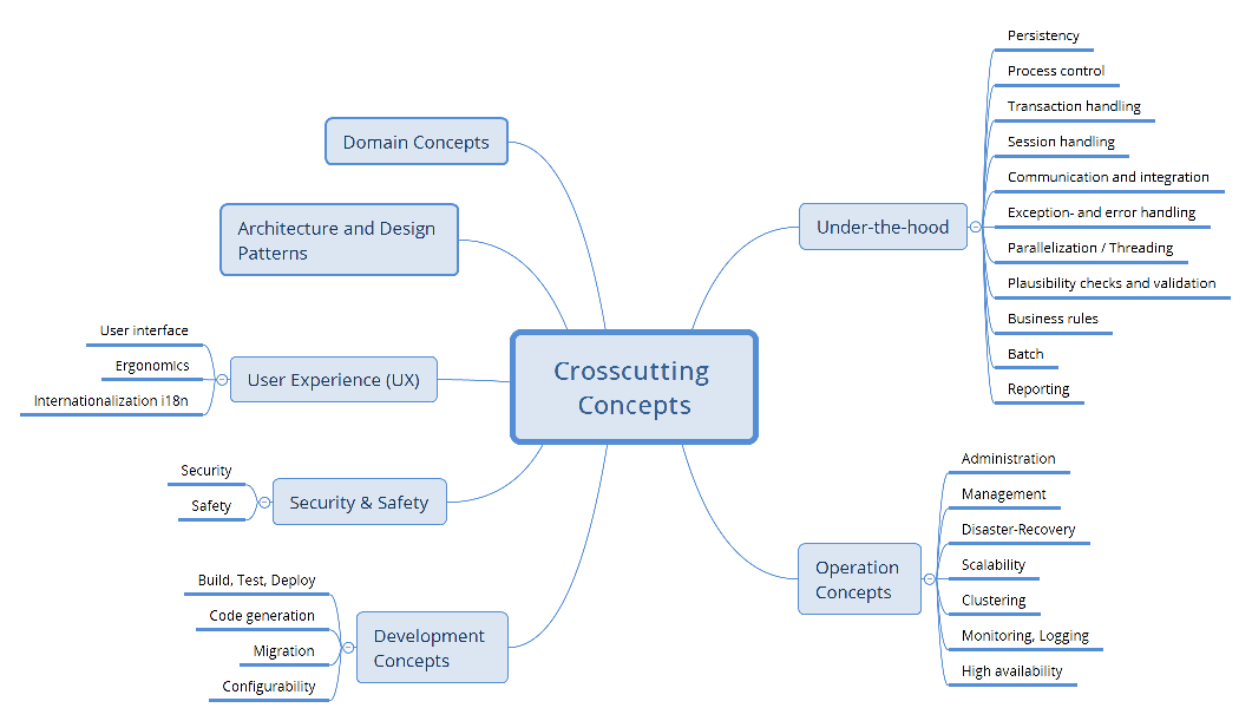
- concept papers with any kind of structure
- cross-cutting model excerpts or scenarios using notations of the architecture views
- sample implementations, especially for technical concepts
- reference to typical usage of standard frameworks (e.g. using Hibernate for object/relational mapping)

Structure

A potential (but not mandatory) structure for this section could be:

- Domain concepts
- User Experience concepts (UX)
- Safety and security concepts
- Architecture and design patterns
- "Under-the-hood"
- development concepts
- operational concepts

Note: it might be difficult to assign individual concepts to one specific topic on this list.



Further Information

See Concepts in the arc42 documentation.

8.1. <*Concept 1*>

<explanation>

8.2. <*Concept 2*>

<explanation>

...

8.3. <*Concept n*>

<explanation>

9. Architecture Decisions

Contents

Important, expensive, large scale or risky architecture decisions including rationales. With "decisions" we mean selecting one alternative based on given criteria.

Please use your judgement to decide whether an architectural decision should be documented here in this central section or whether you better document it locally (e.g. within the white box template of one building block).

Avoid redundancy. Refer to section 4, where you already captured the most important decisions of your architecture.

Motivation

Stakeholders of your system should be able to comprehend and retrace your decisions.

Form

Various options:

- ADR (Documenting Architecture Decisions) for every important decision
- List or table, ordered by importance and consequences or:
- more detailed in form of separate sections per decision

Further Information

See Architecture Decisions in the arc42 documentation. There you will find links and examples about ADR.

10. Quality Requirements

Content

This section contains all quality requirements as quality tree with scenarios. The most important ones have already been described in section 1.2. (quality goals)

Here you can also capture quality requirements with lesser priority, which will not create high risks when they are not fully achieved.

Motivation

Since quality requirements will have a lot of influence on architectural decisions you should know for every stakeholder what is really important to them, concrete and measurable.

Further Information

See Quality Requirements in the arc42 documentation.

10.1. Quality Tree

Content

The quality tree (as defined in ATAM – Architecture Tradeoff Analysis Method) with quality/evaluation scenarios as leafs.

Motivation

The tree structure with priorities provides an overview for a sometimes large number of quality requirements.

Form

The quality tree is a high-level overview of the quality goals and requirements:

- tree-like refinement of the term "quality". Use "quality" or "usefulness" as a root
- a mind map with quality categories as main branches

In any case the tree should include links to the scenarios of the following section.

10.2. Quality Scenarios

Contents

Concretization of (sometimes vague or implicit) quality requirements using (quality) scenarios.

These scenarios describe what should happen when a stimulus arrives at the system.

For architects, two kinds of scenarios are important:

- Usage scenarios (also called application scenarios or use case scenarios) describe the system's runtime reaction to a certain stimulus. This also includes scenarios that describe the system's efficiency or performance. Example: The system reacts to a user's request within one second.
- Change scenarios describe a modification of the system or of its immediate environment. Example: Additional functionality is implemented or requirements for a quality attribute change.

Motivation

Scenarios make quality requirements concrete and allow to more easily measure or decide whether they are fulfilled.

Especially when you want to assess your architecture using methods like ATAM you need to describe your quality goals (from section 1.2) more precisely down to a level of scenarios that can be discussed and evaluated.

Form

Tabular or free form text.

11. Risks and Technical Debts

Contents

A list of identified technical risks or technical debts, ordered by priority

Motivation

“Risk management is project management for grown-ups” (Tim Lister, Atlantic Systems Guild.)

This should be your motto for systematic detection and evaluation of risks and technical debts in the architecture, which will be needed by management stakeholders (e.g. project managers, product owners) as part of the overall risk analysis and measurement planning.

Form

List of risks and/or technical debts, probably including suggested measures to minimize, mitigate or avoid risks or reduce technical debts.

Further Information

See Risks and Technical Debt in the arc42 documentation.

12. Glossary

Contents

The most important domain and technical terms that your stakeholders use when discussing the system.

You can also see the glossary as source for translations if you work in multi-language teams.

Motivation

You should clearly define your terms, so that all stakeholders

- have an identical understanding of these terms
- do not use synonyms and homonyms

Form

A table with columns <Term> and <Definition>.

Potentially more columns in case you need translations.

Further Information

See Glossary in the arc42 documentation.

Term	Definition
<Term-1>	*<definition-1>*
<Term-2>	*<definition-2>*