

American Ninja Warrior

The Ferris Foundation

January 2025

# Introduction and Goals

See [Introduction and Goals](https://docs.arc42.org/section-1/) in the arc42 documentation.

This document describes The American Ninja Warrior (ANW) Platform. The goal of the platform is to consolidate the competition schedules from various ANW leagues and produce an easy-to-use Microsoft Excel spreadsheet from the data. This Excel spreadsheet is used to plan competition participation (travel) throughout the season.

The ANW Platform has the underlying business goals:

Table .1 – Underlying Business Goals

| ID | Goal |
| --- | --- |
| G1 | Import competition details from various ANW leagues. |
| G2 | Associate competition with a gym. |
| G3 | Associate driving time in hours and minutes to each gym. |
| G4 | Create season planner (Excel spreadsheet) based on configurable driving time. |
| G5 | Manage attendance, registration, and notes (attendance data) for each competition. |
| G6 | Sync attendance data back into the ANW Platform. |
| G7 | Associate hotel with a gym. |

## Requirements Overview

Requirements are defined for business features. Business features are logically grouped into business domains along with other features that share similar activities.

“A domain is a high-level, one- or two-word description that can be used to logically group together the stand-alone, self-contained products that carry out the important business features of the application platform.” (Remijan, 2022). Table .1 lists the ANW Platform business domains.

Table .1 – Business Domains

| Domain | Purpose |
| --- | --- |
| Collect | Import data from external sources. |
| Report | Create and export reports from the ANW data. |
| Admin | General administration of ANW data. |

A business feature (aka “feature”) implements one and only one business process which supports the platform’s underlying business goals. A feature defines an operational boundary within a particular domain. A business feature is equivalent to a DDD bounded context and helps manage complexity by clearly separating different business processes of the platform. Table .2 lists the ANW Platform business features, links them to the underlying business goals, and describes the high-level requirements (TOGAF ADM Phase A).

Table .2 – Business Features

| ID | Goal | Feature | Requirements |
| --- | --- | --- | --- |
| F1 | G1 | Load UNAA Competition data from website | Parse and import UNAA Competition data. All dates (the complete list of competitions) are processed every time. UNAA has its normal qualifiers but also has different types of competitions throughout the season. Competitions are associated with a gym. Competition data can be inserted, updated, or deleted if a competition is no longer being held. |
| F2 | G1 | Load FINA Competition data from website | Parse and import FINA Competition data. All dates (the complete list of competitions) are processed every time. FINA has its normal qualifiers but also has different types of competitions throughout the season. Competitions are associated with a gym. Competition data can be inserted, updated, or deleted if a competition is no longer being held. |
| F3 | G1 | Load CSNA Competition data from website | Parse and import CSNA Competition data. All dates (the complete list of competitions) are processed every time. Competitions are associated with a gym. Competition data can be inserted, updated, or deleted if a competition is no longer being held. |
| F4 | G6 | Load competition attendance data from season planner. | The Excel season planner worksheet manages attendance data outside of the system. This attendance data needs to be synched back to the system, so it’s included in the next season planner report. Update competition with attendance data. |
| F5 | G4, G5 | Export season planner report as Excel Worksheet | Generate an Excel season planner report. This report is used to manage attendance at competitions and plan travel. This attendance data is outside of the system and eventually needs to be synced in. |
| F6 | G2, G3 | Manage gyms | Add and update gym data. |
| F7 | G2 | Manage gym aliases | Add and update aliases associated to a gym. |
| F8 | G7 | Manage hotels | Add and update hotel data. Associate to a gym. |

## Quality Goals

The ISO 25010 standard (*ISO 25010*, n.d.) defines a product quality model consisting of nine characteristics with various sub-characteristics for each. The quality goals for the ANW Platform are determined by the top three (max five) characteristics whose fulfillment is of highest importance to the key stakeholders. The quality goals influence the fundamental architectural decisions on how the system and software are developed (TOGAF ADM Phase A).

Table .1 – Platform Quality Goals

| ID | Characteristic | Scenario |
| --- | --- | --- |
| Q1 | Interaction Capability   * Operability | It must be easy to use. Getting the competition data from the various leagues will be a manual process. So, having the platform do as much work as possible to minimize the manual work is of highest priority. This is because the manual work will need to be done many times throughout the competition season. |
| Q2 | Maintainability   * Modifiability | To minimize manual work, the code must be easy to modify so that the platform can do as much of the work as possible. |
| Q3 | Functional Suitability   * Functional Correctness | There must be confidence in the accuracy of all the data in the season planner. Time cannot be wasted manually checking against the platform against the source data. |

## Stakeholders

There are many people involved with the ANW Platform. Each person involved may have one or more distinct roles. Each role carries certain expectations of that person’s responsibilities to the platform and its documentation. Table .1 describes the general roles and expectations of the platform (TOGAF ADM Phase A).

Table .1 – Platform Roles and Expectations

| Role | Expectations |
| --- | --- |
| Owner | The owner is expected to provide the ANW Platform team decisions about (a) underlying business goals, (b) business features and functional (business) requirements, and (c) architecture and non-functional (quality) requirements. |
| Analyst | The analyst is expected to provide the ANW Platform team with fully documented functional (business) requirements of the platform features in fulfillment of the underlying business goals. The analyst owns the “what” of the platform’s operation and maintains it in the architecture documentation. |
| Architect | The architect is expected to provide the ANW Platform team with fully documented non-functional (quality) requirements of the platform features in fulfillment of the underlying quality goals. The architect owns the “how” of the platform’s operation and maintains it in the architecture documentation. |
| Developer | The developer is expected to provide the ANW Platform team with an implementation of the platform in fulfillment of both the functional (business) and non-functional (quality) requirements specified by the analyst and architect respectively. The developer owns the “do” of the platform’s operation and references the architecture documentation. |
| Tester | The tester is expected to provide the ANW Platform team with quality assurance of the platform in fulfillment of both the functional (business) and non-functional (quality) requirements specified by the analyst and architect respectively. The tester owns the “validate” of the platform’s operation and references the architecture documentation. |
| User | The user is expected to provide the ANW Platform team with feedback on the platform. The user owns the “experience” of the platform’s operation with no direct use of the architecture documentation. |

Table .2 lists the stakeholders and their roles (TOGAF ADM Phase A).

Table .2 – Platform Stakeholders

| Role | Name | Contact |
| --- | --- | --- |
| Owner, Analyst, Architect, Developer, Tester, User | Michael Remijan | [mjremijan@yahoo.com](mailto:mjremijan@yahoo.com), 314-288-4662 |
| User | Kelly Remijan | [wasmser.remijan@yahoo.com](mailto:wasmser.remijan@yahoo.com) |

# Architecture Constraints

See [Architecture Constraints](https://docs.arc42.org/section-2/) in the arc42 documentation.

There are two kinds of architecture constraints:

1. **Enterprise constraints** are identified and defined by the EAO and apply to all development. Enterprise constraints are approved by the AGB.
2. **Platform constraints** are identified and defined by various stakeholders based on the functional (business) and non-functional (quality) requirements. Platform constraints must be consistent with enterprise constraints, which always take precedence.

The ANW Platform is required to meet the following constraints. Compliance with these constraints must be verified during implementation governance (TOGAF ADM Phase G). The AGB can deny Authorization to Release if the ANW Platform is not in compliance with any of the Constraints.

## Technical Restraints

Table .1 – Technical Constraints

| Constraint | Explanation | Consequences |
| --- | --- | --- |
| Software Development Standards | All software development standards will be followed. This includes but is not limited to standards for: languages, databases, documentation, source code, source control, artifact management, versioning, branching, testing, and encryption. For details, reference [Software Development Standards.docx](https://1drv.ms/w/c/0c5144d8101c068d/EcQpuhMW6N1Pi5PUS7LXswgBA9LRRfjVAHcdyVDKK4U3Cw?e=5ct9uS). | Not following the software development standards may lead to poor quality code, bugs, low maintainability, security risks, and legal compliance issues. |
| Java | Java will be used to develop the ANW Platform. All Java programming language standards will be followed. This includes but is not limited to standards for: project naming, layout, dependency management, building, releasing, and testing. For details, reference [Java Language Standards.docx](https://1drv.ms/w/c/0c5144d8101c068d/EZOnHeuf785Nq21xSgJlU8cBiAUh5Kgr-kLj2og4rSd0ZQ?e=NVeL7V). | Let’s be honest, all other languages are crap! 😊 |
| SQL | A relational database will be used to store the data. All SQL query language standards will be followed. For details, reference [Software Development Standards.docx](https://1drv.ms/w/c/0c5144d8101c068d/EcQpuhMW6N1Pi5PUS7LXswgBA9LRRfjVAHcdyVDKK4U3Cw?e=5ct9uS). | What else is there to query a relational database? 😊 |
| Apache Derby | Apache Derby will be the relational database. For details, reference [Software Development Standards.docx](https://1drv.ms/w/c/0c5144d8101c068d/EcQpuhMW6N1Pi5PUS7LXswgBA9LRRfjVAHcdyVDKK4U3Cw?e=5ct9uS). | This database is supported by the organization. |
| Project name Maven artifactId | **ferris-anw**  For details, reference [Java Language Standards.docx](https://1drv.ms/w/c/0c5144d8101c068d/EZOnHeuf785Nq21xSgJlU8cBiAUh5Kgr-kLj2og4rSd0ZQ?e=3JjAwi). | Unable to identify the project, naming conflicts or inconsistencies with other projects, poor team collaboration, legal compliance issues. |
| External systems | See heading “ Business Context” for the list of ANW Platform external system integrations. | The ANW Platform relies on competition data from various ANW leagues. Without the data, the ANW Platform cannot operate. |
| Hexagonal Architecture | **Source Code Architecture**  Source code is organized following the Hexagonal Architecture pattern. For more details, reference [Hexagonal Architecture.docx](https://1drv.ms/w/c/0c5144d8101c068d/EdWg_v6PoKNFnRdZXud2lsUBtuQm7krihzPAwZsF5TowDg?e=QTRmNP). | Not following the pattern may lead to highly coupled spaghetti code which is costly to maintain and difficult to rearchitect in the future. |
| Feature-Oriented (modular) Architecture | **Platform Architecture**  The system platform is structured following Feature-Oriented Architecture (FOA). This focuses on structuring the system itself around independent, self-contained features. For more details, reference [Feature-Oriented (modular) Architecture.docx](https://1drv.ms/w/c/0c5144d8101c068d/EVT5zbNfhydIlsiUbR4vQCwBNEbts1aHteOSPDwvhc4kKA?e=4z4D9x). | Not following the pattern may lead to a “big ball of mud” monolith or a “death star” microservice solution, both of which are costly to maintain and difficult to rearchitect in the future. |
| Event-Driven Architecture | **Communication Architecture**  The system platform communication follows an Event-Driven Architecture (EDA). This focuses on structuring the system itself around independent, self-contained features. For more details, reference [Event-Driven Architecture.docx](https://1drv.ms/w/c/0c5144d8101c068d/EYfH1j3vJWFNq_KPsssLbcsBQ3miTwWq1Ez2-RYzHU9Zqg?e=9PH6iS). | Not following the pattern may lead to a highly coupled solution which is costly to maintain and difficult to rearchitect in the future. |

## Legal Constraints

No legal constraints.

## Organizational Constraints

Table .1 – Organizational Constraints

| Constraint | Explanation | Consequences |
| --- | --- | --- |
| Time | August 2025 | If not completed by this date, the platform will not be ready for the 2025-26 ANW season. |
| Budget | None  No direct money is to be spent. | Time is of course money, and it will take a considerable amount of time to complete the platform. However, no additional direct costs, like buying AWS services, are to be spent. |
| Operations | Refer to section Stakeholders for operations details. | n/a |
| Support | Refer to section Stakeholders for operations details. | n/a |

## Political Constraints

No political constraints

# Context and Scope

See [Context and Scope](https://docs.arc42.org/section-3/) in the arc42 documentation.

**Context**. The context determines what users, internal partner systems, and external partner systems interface with the ANW Platform.

**Scope**. The scope determines the responsibilities of each in establishing the interfaces and defining the data exchange.

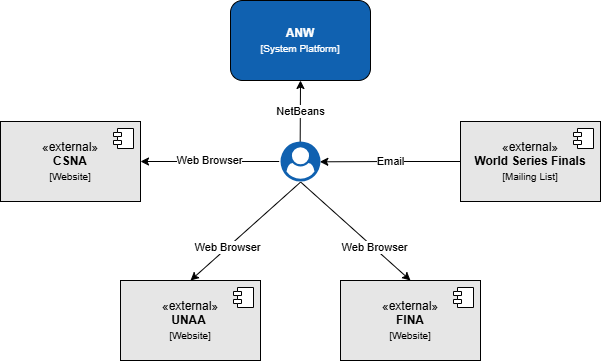
## 

## Business Context

See [System context diagram](https://c4model.com/diagrams/system-context) in the C4 model documentation.

The purpose of the Business Context Diagram in Figure 3.1.1 is to provide a high-level view of **all** systems and actors that surround and have impact on the platform. The diagram’s focus is on the interactions of the platform, showing how **communication channels** are established and what **data** is exchanged. For more details on creating this diagram, reference [Business Context Diagram.docx](https://1drv.ms/w/c/0c5144d8101c068d/EW2DTzAiU8lFodNVonl86ygBy1QKL74e5MsGpCje3EhqPA?e=KvGIjn).

Figure .1 – Business Context Diagram



The Business Context Details in Table 3.1.1 provides more detailed information about each of the communication channels and interactions with the ANW Platform. For more details on creating this table, reference [Business Context Diagram.docx](https://1drv.ms/w/c/0c5144d8101c068d/EW2DTzAiU8lFodNVonl86ygBy1QKL74e5MsGpCje3EhqPA?e=KvGIjn).

Table .1 – Business Context Details

| Partner Organization | Partner System | Data | Data Format | Data Direction | Interface Direction | Interface Mode | Interface  Implementation |
| --- | --- | --- | --- | --- | --- | --- | --- |
| User | n/a | - Competition dates and locations  - Attendance data  - Hotel data  - Gym/Alias data | [tab] | Send | Inbound | Manual | NetBeans IDE |
| CSNA | Website | Competition dates and locations | [tab] | Pull | Outbound | Manual | Web browser |
| UNAA | Website | Competition dates and locations | [tab] | Pull | Outbound | Manual | Web browser |
| FINA | Website | Competition dates and locations | [tab] | Pull | Outbound | Manual | Web browser |
| World Series Finals | Mailing List | Competition dates and locations | Email message | Receive | Inbound | Manual | Email client |

## Technical Context

See [System context diagram](https://c4model.com/diagrams/system-context) in the C4 model documentation.

The Technical Context Diagram in Figure .1 shows channels and transmission media linking the ANW Platform to its environment. It maps the I/O from all users, all internal partner systems, and all external partner systems, specifying port and protocol for all communication.

Figure .1 – ANW: Technical Context Diagram

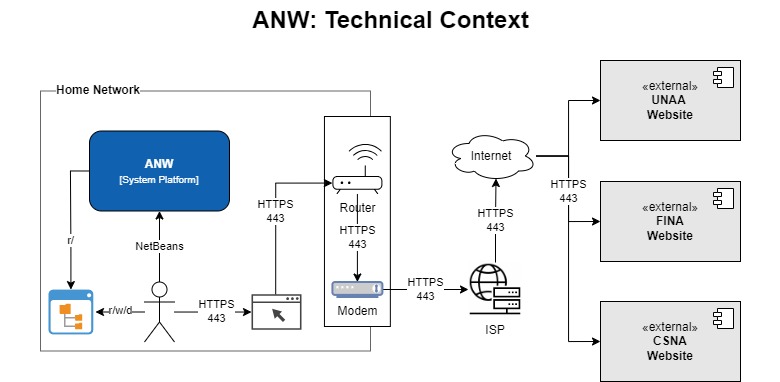


Table .1 – ANW: Technical Context Details

See Appendix B for a detailed description of each table column

| Source | | Target | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| Component | IP Address | Component | IP Address | Protocol | Port | Domain Name |
| ANW Platform | 192.168.1.5 | Wireless Router | 192.168.1.1 | HTTPS | 443 | n/a |
| Wireless Router | 192.168.1.1 | Cable Box | 71.10.191.124 | HTTPS | 443 | n/a |
| Cable Box | 71.10.191.124 | ISP | <unknown> | HTTPS | 443 | n/a |
| ISP | <unknown> | FINA | 138.197.46.19 | HTTPS | 443 | fina.ninja |
| ISP | <unknown> | UNAA | 75.2.0.180 | HTTPS | 443 | www.ultimateninja.net |
| ISP | <unknown> | CSNA | 34.149.87.45 | HTTPS | 443 | www.centralstatesninja.org |

# Solution Strategy

Contents

A short summary and explanation of the fundamental decisions and solution strategies, that shape system architecture. It includes

* technology decisions
* decisions about the top-level decomposition of the system, e.g. usage of an architectural pattern or design pattern
* decisions on how to achieve key quality goals
* relevant organizational decisions, e.g. selecting a development process or delegating certain tasks to third parties.

Motivation

These decisions form the cornerstones for your architecture. They are the foundation for many other detailed decisions or implementation rules.

**Form**

Keep the explanations of such key decisions short.

Motivate what was decided and why it was decided that way, based upon problem statement, quality goals and key constraints. Refer to details in the following sections.

See [Solution Strategy](https://docs.arc42.org/section-4/) in the arc42 documentation.

# 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. [**C4 Container Diagram**](https://c4model.com/?ref=workingsoftware.dev#ContainerDiagram)

**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. [**C4 Component Diagram**](https://c4model.com/?ref=workingsoftware.dev#ComponentDiagram)

**Level 3** zooms into selected building blocks of level 2, and so on. [**Code Diagram**](https://c4model.com/?ref=workingsoftware.dev#ComponentDiagram)

See [Building Block View](https://docs.arc42.org/section-5/) in the arc42 documentation.

## 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.

### <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>*

### <Name black box 2>

*<black box template>*

### <Name black box n>

*<black box template>*

### <Name interface 1>

…

### <Name interface m>

## 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

### White Box *<building block 1>*

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

*<white box template>*

### White Box *<building block 2>*

*<white box template>*

…

### White Box *<building block m>*

*<white box template>*

## 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.

### White Box <\_building block x.1\_>

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

*<white box template>*

### White Box <\_building block x.2\_>

*<white box template>*

### White Box <\_building block y.1\_>

*<white box template>*

# 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
* …

See [Runtime View](https://docs.arc42.org/section-6/) in the arc42 documentation.

## <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.>*

## <Runtime Scenario 2>

## …

## <Runtime Scenario n>

# 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.

**[C4 Deployment Diagram](https://c4model.com/diagrams/deployment)**

**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.

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.

See [Deployment View](https://docs.arc42.org/section-7/) in the arc42 documentation.

## 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>*

## 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.

### *<Infrastructure Element 1>*

*<diagram + explanation>*

### *<Infrastructure Element 2>*

*<diagram + explanation>*

…

### *<Infrastructure Element n>*

*<diagram + explanation>*

# 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.



See [Concepts](https://docs.arc42.org/section-8/) in the arc42 documentation.

## *<Concept 1>*

*<explanation>*

## *<Concept 2>*

*<explanation>*

…

## *<Concept n>*

*<explanation>*

# 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](https://cognitect.com/blog/2011/11/15/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

See [Architecture Decisions](https://docs.arc42.org/section-9/) in the arc42 documentation. There you will find links and examples about ADR.

# 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.

See [Quality Requirements](https://docs.arc42.org/section-10/) in the arc42 documentation.

## 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.

## 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.

# 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.

See [Risks and Technical Debt](https://docs.arc42.org/section-11/) in the arc42 documentation.

# Glossary

Table 12.1 – Glossary

| Term | Definition |
| --- | --- |
| ADM | Architecture Development Method – TOGAF |
| AGB | Architecture Governance Board – TOGAF |
| ANW | American Ninja Warrior |
| ATR | Authorization To Release |
| CSNA | Central State Ninja Association |
| EAO | Enterprise Architecture Office |
| FINA | Federation of International Ninja Athletics |
| ISP | Internet Service Provider |
| TOGAF | The Open Group Architecture Framework |
| UNAA | Ultimate Ninja Athlete Association |

# References

Remijan, M. (2022, November 17). *Pragmatic Strategy for Deconstructing a Monolith into Microservices*. <https://mjremijan.blogspot.com/2022/11/pragmatic-strategy-for-deconstructing.html>

*ISO 25010*. (n.d.). <https://iso25000.com/index.php/en/iso-25000-standards/iso-25010>

*Arc42*. (n.d.). Arc42 Documentation. <https://docs.arc42.org/home>

*C4 Model*. (n.d.). C4 Model. <https://c4model.com>

*The TOGAF® Standard Architecture Development Method*. (n.d.). <https://pubs.opengroup.org/togaf-standard/adm/>

# Appendices

## Appendix B

Additional details for Technical Context.

Table 14.2.1 – Technical context source and target

| Column | Description |
| --- | --- |
| Source | The source or starting point of the network communication |
| Target | The target or ending point of the network communication |

Table 14.2.2 – Detailed description of each table column for the technical context.

| Column | Description |
| --- | --- |
| Component | The name of the component participating in the network communication |
| IP Address | The IP Address of the component participating in the network communication |
| Protocol | The name of the network communication protocol. Be as specific as possible (HTTPS) without going into implementation details (REST, SOAP). The implementation details are part of the business context information. |
| Port | The port number of the network communication. If using a commonly known default port, still enter the number. |
| Domain Name | The domain name of the target component participating in the network communication |

# Gym Subsystem Notes

**GYM**

| Colum | Data Type | Constraints | Comment |
| --- | --- | --- | --- |
| ID | String | Primary key | UUID |
| NAME | String | Unique | Gym name. |
| GYM\_URL | String |  | Gym website. |
| FACEBOOK\_URL | String |  | Gym facebook page. |
| CITY | String | Required | City location of gym. |
| STATE | String | Required | State location of gym. |
| DIRECTION\_URL | String | Required | Google maps direction. |
| DRIVE\_HOURS | Number | Required | Hours to drive to gym. |
| DRIVE\_MINUTES | Number | Required | Minutes to drive to gym. |

**GYM\_ALIAS**

| Colum | Data Type | Constraints | Comment |
| --- | --- | --- | --- |
| ID | String | Primary key | UUID |
| GYM\_ID | String | Foreign key | GYM.ID |
| NAME | String | UNIQUE | Gym alias name |

**Import**

| Directory | Comment |
| --- | --- |
| ${import}/gym/in | Find files here to import. |
| ${import}/gym/archive | Files moved here from “in” after import. |
| ${import}/gym/done | Files containing lines of data imported without error. |
| ${import}/gym/error | Files containing lines of data with errors. |

Process Workflow

* Access data
* Read line of data
  + Ignore starts with #
  + Ignore empty
* Tokenize line of data
  + Required missing?
    - Error file
* Lookup gym name
  + GYM + GYM\_ALIAS
* Gym name exists
  + Update by GYM.ID
* Gym name map to existing
  + GYM + GYM\_ALIAS
  + Update by GYM.ID
  + Insert new alias
* Gym name new
  + Insert GYM
* Done file

# Excel Season Planner Worksheet

Crazy idea…Excel script using ODBC to update Derby directly?

A screenshot of a computer

Description automatically generated