# 

American Ninja Warrior

The Ferris Foundation

January 2023

About The Ferris Foundation

The Ferris Foundation is the software organization created by Michael Remijan for all of his software development projects.

About arch42

[Home | arc42 Documentation](https://docs.arc42.org/home/)

# Introduction and Goals

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

This document describes The Ferris Foundation – American Ninja Warrior (ANW) software 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 that data. This Excel spreadsheet is used to plan competition participation (travel) throughout the season.

The ANW Platform has the underlying business goals:

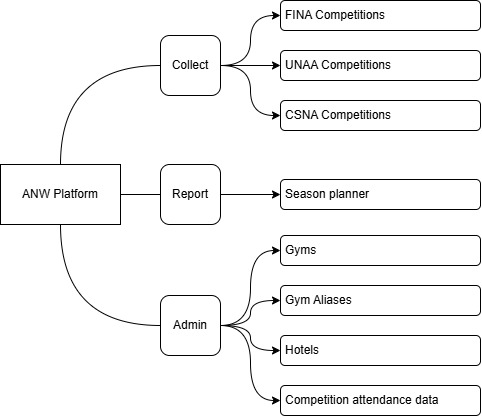
Table 1.1 – Underlying Business Goals

| 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 Excel season planner worksheet 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

The ANW Platform implements the following features to meet the underlying business goals.

Diagram 1.1.1 - Feature Map



“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 1.1.1 describes the ANW Platform domains.

Table 1.1.1 – ANW Platform domains

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

Each feature is associated with an underlying business goal and the high-level requirements needed to fulfil that business goal. Table 1.1.2 describes the ANW Platform features.

Table 1.1.2 – ANW Platform features

| ID | Goal | Feature | Requirements |
| --- | --- | --- | --- |
| F1 | G1 | UNAA Competitions | Parse and import UNAA Competition data. 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 or updated. |
| F2 | G1 | FINA Competitions | Parse and import FINA Competition data. 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 or updated. |
| F3 | G1 | CSNA Competitions | Parse and import CSNA Competition data. Competitions are associated with a gym. Competition data can be inserted or updated. |
| F4 | G6 | Competition attendance data | The Excel season planner worksheet manages attendance data outside of the database. This attendance data needs to be synched back to the database, so it’s included in the next season planner report. Updated competition with attendance data. |
| F5 | G4, G5 | Season planner | Generate the Excel season planner report. This report manages attendance data outside of the database. |
| F6 | G2, G3 | Gyms | Add and update gym data. |
| F7 | G2 | Gym Aliases | Add and update aliases associated to a gym. |
| F8 | G7 | 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 goal for the ANW Platform system and software architecture is determined by the top three (max five) characteristics whose fulfillment is of highest importance to the major stakeholders. The quality goal influences the fundamental architectural decisions on how the system and software is developed.

Table 2.1.1 – ANW Platform quality goal

| ID | Characteristic | Scenario |
| --- | --- | --- |
| Q1 | Ineration 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 data. |

## Stakeholders

There are many people involved with the ANW Platform. Each person has a particular role and expectation with respect to the ANW Platform architecture and its documentation. People may have many roles. Table 3.1.1 describes the roles and expectations. Table 3.1.2 lists the stakeholders and their roles.

Table 3.1.1 – ANW Platform roles and expectations

| Role | Expectations |
| --- | --- |
| Owner | The owner is expected to provide the ANW Platform team decisions about underlying business goals; platform features and functional (business) requirements; and platform architecture and non-functional (quality) requirements. |
| Analyst | The analyst is expected to provide the ANW Platform team |
| Architect | The architect is expected to provide the ANW Platform team |
| Developer | The developer is expected to provide the ANW Platform team |
| Tester | The tester is expected to provide the ANW Platform team |
| User | The user is expected to provide the ANW Platform team |

Table 3.1.2 – ANW Platform stakeholders

| Role | Contact | Expectations |
| --- | --- | --- |
| Owner, Architect, Developer, Tester, User | Michael Remijan  [mjremijan@yahoo.com](mailto:mjremijan@yahoo.com)  314-288-4662  @mjremijan | Owner, architect, developer, user |

# Architecture Constraints

Contents

Any requirement that constraints software architects in their freedom of design and implementation decisions or decision about the development process. These constraints sometimes go beyond individual systems and are valid for whole organizations and companies.

**Motivation**

Architects should know exactly where they are free in their design decisions and where they must adhere to constraints. Constraints must always be dealt with; they may be negotiable, though.

**Form**

Simple tables of constraints with explanations. If needed you can subdivide them into technical constraints, organizational and political constraints and conventions (e.g. programming or versioning guidelines, documentation or naming conventions)

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

# Context and Scope

Contents

Context and scope - as the name suggests - delimits your system (i.e. your scope) from all its communication partners (neighboring systems and users, i.e. the context of your system). It thereby specifies the external interfaces.

If necessary, differentiate the business context (domain specific inputs and outputs) from the technical context (channels, protocols, hardware).

**Motivation**

The domain interfaces and technical interfaces to communication partners are among your system’s most critical aspects. Make sure that you completely understand them.

**Form**

Various options:

* Context diagrams
* Lists of communication partners and their interfaces.

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

## Business Context

Contents

Specification of **all** communication partners (users, IT-systems, …) with explanations of domain specific inputs and outputs or interfaces. Optionally you can add domain specific formats or communication protocols.

[**C4 System Context diagram**](https://c4model.com/?ref=workingsoftware.dev#SystemContextDiagram)

**Motivation**

All stakeholders should understand which data are exchanged with the environment of the system.

**Form**

All kinds of diagrams that show the system as a black box and specify the domain interfaces to communication partners.

Alternatively (or additionally) you can use a table. The title of the table is the name of your system, the three columns contain the name of the communication partner, the inputs, and the outputs.

**<Diagram or Table>**

**<optionally: Explanation of external domain interfaces>**

## Technical Context

Contents

Technical interfaces (channels and transmission media) linking your system to its environment. In addition a mapping of domain specific input/output to the channels, i.e. an explanation which I/O uses which channel.

**Motivation**

Many stakeholders make architectural decision based on the technical interfaces between the system and its context. Especially infrastructure or hardware designers decide these technical interfaces.

**Form**

E.g. UML deployment diagram describing channels to neighboring systems, together with a mapping table showing the relationships between channels and input/output.

**<Diagram or Table>**

**<optionally: Explanation of technical interfaces>**

**<Mapping Input/Output to Channels>**

# 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 |
| --- | --- |
| ANW | American Ninja Warrior |
|  |  |

# 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>

# 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