**Netcompany – Methodology and Security**

**O0500 - Software Architecture**

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

|  |  |
| --- | --- |
| **Version:** | **2.2** |
| **Status:** | Approved |
| **Approver:** | Leni Støvring Barfred |
| **Author:** | Leni Støvring Barfred |



**Document history**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Date | Author | Status | Remarks |
| 1.0 | 23-09-2015 | Leni Barfred | Final | Taken from the intranet and revised |
| 2.0 | 04-05-2016 | Simon Shan Man Ng | Approved | 2016 New logo |
| 2.1 | 06-04-2020 | Mads Falkenberg Brenøe | Approved | Added Data view and Security view |
| 2.2 | 15-09-2020 | Leni Støvring Barfred | Approved | New colors + security input from CHR |

**References**

|  |  |  |  |
| --- | --- | --- | --- |
| Reference | Title | Author | Version |
|  |  |  |  |

Table of contents

[1 Introduction 3](#_Toc51080262)

[1.1 Purpose 3](#_Toc51080263)

[1.2 Scope 3](#_Toc51080264)

[1.3 Target audience 3](#_Toc51080265)

[1.4 Tools 3](#_Toc51080266)

[2 Software architecture – what and why 4](#_Toc51080267)

[3 Contents of O0500 5](#_Toc51080268)

[3.1 Architectural aims and framework 5](#_Toc51080269)

[3.2 Architectural model 7](#_Toc51080270)

[3.2.1 Use case view 8](#_Toc51080271)

[3.2.2 Logical view 8](#_Toc51080272)

[3.2.3 Process view 8](#_Toc51080273)

[3.2.4 Implementation view 8](#_Toc51080274)

[3.2.5 Deployment view 9](#_Toc51080275)

[3.2.6 Data view 9](#_Toc51080276)

[3.2.7 Security view 10](#_Toc51080277)

[3.2.7.1 CIA (Confidentiality, Integration, Availability) 11](#_Toc51080278)

[3.2.7.2 Important issues to consider 11](#_Toc51080279)

[3.2.7.2.1 Cross-Site Scripting (XSS) 12](#_Toc51080280)

[3.2.7.2.2 Using Components with Known Vulnerabilities (2017) 12](#_Toc51080281)

[3.2.7.2.3 Secure HTTP Headers 12](#_Toc51080282)

# Introduction

## Purpose

The purpose of this deliverable is to provide a collective overview of the architecture for the various system stakeholders, including customers, business architects, technical architects, developers, database specialist and operators.

## Scope

The software architecture interacts with the following deliverables and should create an overview of these:

* A0140 - Use Case
* O0400 - Technical Infrastructure
* D0130 - Logical Datamodel
* DD160 - Programming Guidelines

The software architecture document also provides input for detailed design:

* DD130 - Detailed Design
* DD120 - Physical Datamodel

## Target audience

All stakeholders, including developers, technical architects, business architects, operators and the customer.

* Architects and developers use the software architecture to understand their marching orders. The software architecture defines immovable constraints and degrees of freedom that can be used.
* Testers and developers of integrations use the software architecture to specify the correct "black box" behaviour of the system.
* The project manager uses the software architecture to help build up teams, define the organisational structure and distribute tasks.
* Designers of other systems use the software architecture to define the set of interfaces offered and the protocols necessary for technical compatibility (see also D0180 - External Interface Design).
* Architects and requirement definers representing the customer use the software architecture as a discussion reference for negotiation and trade-offs in the event of competing requirements.
* Maintenance uses the documentation as a starting point for maintenance.

## Tools

Use of a UML model in Sparx Enterprise Architect, where the software architecture can be maintained, is recommended.

# Software architecture – what and why

This deliverable is essential to the success of the project. The software architecture is the first document a new project participant should read in order to gain an overview of the system.

The deliverable must be started as soon as possible in the project. Introduction of a software architecture as early as the actual tendering process is recommended.

Software architecture is an essential discipline in software development and particularly important when implementing large systems.

The software architecture document is the first document – and for some, the only document – to read when you want to understand:

* what the system can do
* what the system is responsible for
* what elements make up the system
* where the system runs
* where the best place is to add new functionality
* how new functionality is implemented which is consistent with the rest of the code
* how the requirements are implemented by the design
* how the design is implemented by the structure
* how the structure is supported by the physical deployment environment.

The software architecture document:

* Is important for all kinds of project
* Is important for projects of all sizes
* The degree of detail must be just right
* Describes the things that the code itself does not describe
* Explains the structure of our software.
* Is a precise abstraction of the architecture
* Describes principles and constraints that have to be enforced
* Technologies and platforms
* Evaluates how the architecture meets the requirements

Software architecture also involves security. The code and structure we supply must be secure.

Microsoft offers a free threat modelling tool which has been created to help designers locate threats during the design phase of a development project.  
LINK to further information: <http://blogs.microsoft.com/cybertrust/2015/10/07/whats-new-with-microsoft-threat-modeling-tool-2016/>

The job of the software architect is to convey the entire system to project participants and must help the project participants to understand the scope, framework, constraints, organisation and distribution of responsibilities in the system. With this knowledge, project participants can focus more effectively on the components to be implemented and their mutual relationships, shielding project participants from non-essential details.

If the overall system is broken down competently into smaller elements (with mutual relationships), groups of project participants separated by organisational, geographical and temporal boundaries will be allowed to work together productively on resolving a greater problem than would be the case if they were to work individually.

The aim of the software architecture is to allow each individual component of the system to be implemented with a knowledge of just the general framework, not all system details.

The characteristics exhibited by the system when it is executed are one of the most important factors to consider when designing or implementing the architecture of the system. Of course, what the system calculates is one of these factors. But equally as important are characteristics (including quality attributes) such as performance, robustness, security and maintainability. The architecture must document and communicate how these characteristics are achieved.

Fundamentally, the documentation of the software architecture serves three purposes:

1. A way of introducing new project participants to the system. In this context, project participants may be new team members, external parties or even a new architect.
2. A mechanism for communicating with stakeholders – see the section on target group
3. The basis of a detailed design

The software architecture documentation must balance these varied purposes. It must be sufficiently abstract to allow it to be understood quickly by new project participants. It must be sufficiently detailed so as to act as a working drawing for the build. At the same time, it must have enough information to act as a basis for analysis and later maintenance of the architecture.

# Contents of O0500

## Architectural aims and framework

This section must cover:

* A summary of non-functional requirements that significantly affect the software architecture
  + Why are these requirements architecturally significant?
* A summary of the architectural framework that significantly affects the software architecture.
  + Why does this affect the architecture significantly?
  + Why are the constraints applied?
  + Are the constraints tactical or strategic?
* A summary of the architectural principles selected by the architect himself
  + Architectural principles stem from either the architect's own experience or inspiration from other architects
  + For the project, architectural principles are what the architecture adopts
  + Architectural principles improve consistency

Examples of non-functional requirements

* capacity requirements (e.g. 1,000,000 million orders a year)
* performance requirements (e.g. 1,000 simultaneous users or response time below 1.0 second)
* scalability requirements (e.g. the capacity can be increased by adding extra hardware)
* user-friendliness requirements (e.g. the system can be used by the visually impaired)
* accessibility requirements (e.g. the architecture must be able to deal with failure of a single component)
* security requirements (e.g. the functionality of the system is compliant with the requirements of the Personal Data Act, revision requirements)

Examples of architectural frameworks

* requirements relating to compliance with architectural or design-related guidelines or policies within the customer's organisation (e.g. the system is based on .NET, SOA, etc.).
* requirements for specifically approved technologies
* requirements for standards (e.g. OIOXML)
* requirements for protocols and message formats

Examples of architectural principles

* A web-based channel is used for customers' self-service solution so that they can access their accounts
* The system must present customers' services by orchestrating activities and not re-implementing business logic
* The system must only save data related to the channel
* All webpages must be implemented according to the MVC pattern
* The web application is stateless and HTTP sessions must not be used

All requirements of this type which impact upon the architecture are documented here, possibly with reference to external documents.

## Architectural model

The insight requirements of the various stakeholders are met by illustrating the architecture from various views. Each view covers a specific   
perspective on the whole thing and provides an overview of the deliverables involved. This breakdown into views is known as the architectural representation or architecture model.

Philippe Kruchten's 4+1 architecture model is recommended:

* **Logical view** describes a functional breakdown of the system, expressed in functional terms and the language of the solution domain (the language of the business)
* **Process view** describes the simultaneity and synchronisation aspects of the system.
* **Implementation view** describes the static organisation into modules, packages and layers.
* **Deployment view** describes the depiction of the logical view and the process view in a physical deployment environment.

You start with a logical view and work horizontally or vertically towards the deployment view. Finally, the views are Held up against the functional requirements of the system reflected in the use case view in order to check that the description is comprehensive.

The choice of views will be dependent on the project in question. A subset of 4+1 can be used, or other views can be introduced which are relevant in context.

|  |  |  |  |
| --- | --- | --- | --- |
| View | Summarises | Stakeholders | Recommendation |
| Use case view | Functional requirements | All | Strongly recommended |
| Logical view | Functional subsystems/components  Use case implementation | Designers, developers | Strongly recommended |
| Process view | Performance  Simultaneity (including Quality of Service) | Developers | Recommended for distributed systems, e.g. separate databases or multiple system processes |
| Implementation view | Use case implementation  Development | Developers | Strongly recommended |
| Deployment view | Technical infrastructure | Developers, operators | Strongly recommended |
| Data view | Organisation of data | Developers, DBA | Recommended if there is a need to summarise the logical datamodel |
| Security view | Security requirements | All | Recommended if communication of system features is required from a security angle |

### Use case view

The use case view is a brief summary of the use case model A0140 - Use Case and its division of the system into functional areas of responsibility.

A selection of architecturally essential use cases are also identified. This may, for example, include use cases if scenarios involve central critical functionality, or use cases that span many different parts of the system.

### Logical view

The logical view provides an overview of the Analysis model by describing the system's most essential Analysis classes, their most essential attributes and methods. The Analysis model is a model of functionality of the system in the language of the solution domain, i.e. without technical terms. In the description, emphasis is placed on providing clear delimitation of areas of responsibility between classes rather than attempting to provide a complete description of each individual class. Classes are described in sufficient detail so that their role in the design is highlighted and well-defined. UML component and class diagrams are suitable for this purpose. The Analysis model is often prepared in a UML model equivalent to the Design model in DD130 - Detailed Design.

Implementation of particularly important use cases are also identified in use case view. Use case implementation is a review of a scenario that shows how the classes interact.

UML sequence and collaboration diagrams are suitable for this purpose.

### Process view

Process view provides an overview of "what is running"

It describes the system's simultaneity and synchronisation aspects, including a breakdown into processes and threads, and may be a good place to highlight non-functional challenges such as

* Performance
* Response times
* Load distribution
* Accessibility
* Scaling options
* Synchronisation and replication of distributed databases
* Strategies for transaction management
* Runtime dependencies

Documented as a UML Component Diagram (or Class Diagram). Each process or thread is represented as a Component (or Class) in the diagram.

Process view is often included only if the system's deployment structure is particularly complex (distributed) or is not simply managed by standard software such as a single application server process.

### Implementation view

Implementation view provides an overview of "how it should be implemented".

Implementation view provides an overview of the system's static organisation into modules, layers and packages and is an overview of or extract from the Design model prepared together with DD130 - Detailed Design. Implementation view is supplemented with an extensive range of architectural principles (options) which define the general framework and rules for implementation of the system.

The following topics should be discussed and documented at a general level in the software architecture document:

* Principles for organisation of code. Package diagrams with dependencies,, including dependencies to a potential standard platform. This standard platform may, for example, be Microsoft SharePoint Server, Microsoft CRM, ASP.NET MVC, Oracle WebLogic
* Principles for layering and distribution of responsibilities within the layers. The distribution of responsibilities should come up for example, describe the positioning and responsibilities of the domain logic and the positioning and responsibilities of the application logic.
* Principles for interfaces between layers (and tiers). E.g. are DTO objects, domain objects, weakly typed objects used?
* Principles for general data access and integration patterns. E.g. are Active Record, Domain Model, Transaction Script used?
* Transaction life cycle principles – which objects own, start, cancel and execute commit on the transaction
* Principles for compensating transactions in distributed systems. E.g. how a partial error is dealt with in a distributed scenario where one transaction goes well but another does not.
* Principles for testability. E.g. what demands are made of the code and its structure so as to make it sufficiently testable?
* Principles for performance improvements. E.g. caching, eager load, lazy load
* Principles for error handling. E.g. how are Exceptions thrown up through the layers, and how are they logged?

All implementation class types must be identified. The degree of detail of the implementation class types may be varied as required. The system may include a number of subsystems, each of which has different implementation class types. For example, the same system may include the implementation class types SharePoint webparts, SharePoint event handlers, Microsoft CRM workflows, custom application components and batch jobs.

The general description is documented in the software architecture. The detailed description is documented in DD160 - Programming Guidelines. E.g., how transaction management is specifically coded is described in DD160. O0500 describes the principles.

### Deployment view

Deployment view provides an overview of "where it is installed".

The physical perspective provides an overview of the depiction of the other views in the physical deployment environment. This view should present a diagram of the physical deployment environment and provide a brief description of the logical components.

* Where are the software components deployed?

The system's deployment aspects can also be described here.

The details are described in O0400 - Technical Infrastructure.

### Data view

This perspective can be included in O0500 if there is a need to express a comprehensive overview of the system based on data.

There are two different dimensions:

1. More precise description of data the logical data model
2. How data moves

If the system includes Data Warehouse, this perspective will be obvious to include.

If data is replicated in several places, as a starting point for each piece of data there should be a single source of truth, so it is always clear which system has the true value of the data.

This section can, for example, be used to provide an overview of where data arises, how it is processed and exhibited in the end. Processing of data can for example be cognitive computing.

### Security view

The position on how the architecture and design of the system mitigates known vulnerabilities. Here one must be concrete and not fluffy - it should not be a statement of intent, but a concrete description of what choices have been made at the architecture level, to ensure that the final system supports the expected security requirements – including, but not lilited to: choice of central frameworks, authentication model, access control model, and OWASP Top10 Mitigation.

The following topics should be discussed and documented overall in the software architecture document:

* Authentication
* Authorization
  + Describe how the software architecture supports the authorization model. Consider on what logical layer validations and access control mechanisms should reside.
  + High level description of the set of rules that governs what actions a user or group of users can perform. I.e. ACL, attribute based or role-based models.
* Transport safety
* Certificates
* How passwords are stored
* Service Accounts
* Logging
  + Design structured revision and security logging according to customer and legal requirements.
* “Hardening”: Eg
  + Server versions do not appear in headers
  + Error messages are not displayed to end users
  + Unsafe SSL / TLS ciphers are turned off
  + HTTP Strict Transport Security is always used
* Production versus non-production: Is anonymized data from production used in testing?
* Threat Model (if your project has an O0210 Security plan the threat model should be in O0210).
  + E.g. based on OWASP Top Ten

Think Security by design. Preferably, the developers who design / implement the business functionality of the system should not have to spend much of their daily work ensuring that the system is secure. It should be impossible / very difficult to design / develop code that introduces vulnerabilities.

* Consideration should be given to whether a given security feature can be bypassed by the developers and if so, how this can be done without compromising the entire system's security. How can a developer safely step beyond the framework set up ?. Eg.
  + The system's HTTP response header is defined centrally for all responses. If there is a need for a single response to have a different configuration of a given header, how can this be done without changing the central setup.

Think Defense in Depth. Be sure to have multiple layers of security and evaluate how a failure of one layer affects the overall security of the system, what options it gives an attacker and what defenses the next layer offers.

* If the system consists of different subsystems, then using different database users for each subsystem will allow different database rights to be specified in the database. Eg. some subsystems will only be able to read given data, while other subsystems will be able to create / edit that data.
* Consider whether specific vulnerable parts of the system (eg a customer-facing self-service) should only be able to access data via SOAP / REST services so that the database is not directly accessible. This allows to minimize the data that can be retrieved from the back-end system and a compromise of the self-service solution will not give an attacker direct access to the database.

Describe how it is ensured that the developers design and develop the system in the desired way and how we detect if this does not happen. Consider how the source code can be "monitored" when the security architecture is not compromised. is done by using Pull Requests and specifying which packages / files require review by a particular group of people. Changes to key safety components must not be uncontrolled. Describe static or dynamic code analysis with special focus on identifying potential vulnerabilities in implementation.

#### CIA (Confidentiality, Integration, Availability)

Does the system store or handle sensitivity information that requires special precautions to be taken regarding encryption (Encryption of data at rest).

* Encryption of data in database. This protects data in case an attacker gets access to the the database files.

Is there data where it is particularly relevant to ensure data integrity so that changes behind the system will be identified.

* Eg. a basis of calculation that has been approved. Here it may be relevant to sign the calculation basis, so that any. changes made behind the system are not used for calculation.

Is there data that is particularly relevant to accessibility.

* If the main purpose of a system is to pay out social benefits on a regular basis, it may be relevant to design a fall-back plan in case the system is unable to calculate and make the payments. In this case, data on last period’s payments may be relevant, allowing for the system to generate the same payments as the previous month.

#### Important issues to consider

The following OWASP points should be considered as a minimum (years indicate when the point was last included in OWASP Top10).

* Injection (2017)
  + Not just SQL injection. Consider whether the system may be vulnerable to other injection types.
* Broken Authentication and Session Management (2017)
* Sensitive Data Exposure (2017)
* XML External Entities (XXE) (2017)
* Broken Access Control (2017)
  + Insecure Direct Object References (2013)
  + Insufficient Function Level Access Control (2013)
    - Failure to Restrict URL Access (2010)
* Security Misconfiguration (2017)
* Cross-Site Scripting (XSS) (2017)
* Insecure Deserialization (2017)
* Using Components with Known Vulnerabilities (2017)
* Insufficient Logging & Monitoring (2017)
* Cross-Site Request Forgery (CSRF) (2013)
* Unvalidated Redirects and Forwards (2013)
* Insecure Cryptographic Storage (2010)

Subjects that are not on the OWASP Top10 but which are also important:

* Managing file uploads (virus / macro virus, zip bomb, executable files).
* Server-side parameter validation (Parameter Tampering mitigation)
* HTTP Headers setup
  + OWASP Secure Headers Project (particularly Content-Security-Policy is interesting). This may affect how, among other things, using javascript is implemented in the application and is therefore important to decide early in the project.

##### Cross-Site Scripting (XSS)

Take into account that there may be requirements / wishes / needs for the system to specify HTML formatted input (either from the users or when specifying the system texts), eg. use of TinyMCE. Certain XSS mitigation options will also reject that kind of input, thus rendering the system unusable.

##### Using Components with Known Vulnerabilities (2017)

Describe how it is ensured that particular third-party components (eg libs) included in the solution are "monitored" for updates and kept up-to-date. Also consider whether criteria should be set for acceptance of a third-party component so that from the outset it is ensured that a component that is not maintained is selected - or that

##### Secure HTTP Headers

Determine if and which HTTP headers the project will use and what is generally allowed. Eg. can be specified to header:

* Content-Security-Policy is used and:
  + In-line javascript or javascript from sources other than the system itself is not allowed.
  + Pages must not be embedded in frames.
  + All violations of rules are submitted to the system.
* Strict-Transport-Security is used and:
  + That max-age is set to one week.
  + This setting does not apply to sub domains