**Netcompany – Methodology and Security**

**DD130 - Detailed Design**

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

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



**Document history**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Date | Author | Status | Remarks |
| 1.0 | 28-9-2015 | Leni Støvring Barfred | Final | Taken from the intranet |
| 2.0 | 04-05-2016 | Simon Shan Man Ng | Approved | 2016 New logo |
| 2.1 | 15-09-2020 | Leni Barfred | Approved | New colors |
| 2.2 | 12-11-2020 | Christian Hjorth Møller | Approved | Minor updates and additions |

**References**

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

Table of contents

[1 Introduction 3](#_Toc56764449)

[1.1 Purpose 3](#_Toc56764450)

[1.2 Scope 3](#_Toc56764451)

[2 Preparation guidelines 3](#_Toc56764452)

[2.1 Selecting a modelling tool 4](#_Toc56764453)

[2.2 Selecting models for detailed design 4](#_Toc56764454)

[2.3 Analysis model 5](#_Toc56764455)

[2.3.1 Use case realisations 6](#_Toc56764456)

[2.3.2 Analysis class design 6](#_Toc56764457)

[2.4 Design model (the actual class design) 7](#_Toc56764458)

[2.4.1 Use case realisation 8](#_Toc56764459)

[2.4.2 Class design 8](#_Toc56764460)

[2.5 Implementation model 9](#_Toc56764461)

[2.6 Format 9](#_Toc56764462)

[2.7 Review recommended 9](#_Toc56764463)

# Introduction

## Purpose

The purpose of DD130 - Detailed Design is to develop the implementation perspective from *O0500 - Software Architecture* and expand upon descriptions of components, classes, attributes, methods and relations.

The developers and architects who are to understand the design and implement or review the solution are the target group. The customer’s technical architect may also be interested in reviewing the document for quality assurance of the solution. However, the document is intended to be an internal document which does not have to be approved by the customer.

It is also a prerequisite for giving an exact build estimate. The estimate should always be validated after the detailed design is done.

## Scope

Although the classes are related to their position in the overall software architecture, the architecture should not be described in this document. Instead, please see *O0500 - Software Architecture* so that the architecture does not have to be maintained in two locations. It is preferred to link to relevant paragraphs in O0500 if the architecture needs to be addressed.

# Preparation guidelines

In practice, detailed design/class design is prepared as increasingly detailed use case realisations, identifying the components, subcomponents, classes and methods of the system. This process is inspired by RUP (see [RUP-A] and [RUP-D] for a detailed analysis of the corresponding RUP phases).

Detailed design can be prepared for example using a dedicated UML tool. The deliverable can then either comprise the model itself, or it will be possible to extract the main content for the deliverable from the model in the form of a Word/RTF file. The UML model is typically begun at the start of the design phase, and content for a number of detailed design end products is extracted from here. Detailed design can also be documented using other concepts than UML like flow charts, API design, Archimate models and so on. The selected form should always be described in *D0110 – Technical Design Guidelines*.

Irrespective of whether the deliverable is documented in a dedicated tool or in Word, it should include an introductory reading guide which also describes general design choices such as the tool used, the modelling of specific general aspects, the use of particular patterns or restrictions in terms of platform or standard packages. But the general architecture should not be described but instead referred.

## Selecting a modelling tool

Going forward, the project may benefit from selecting a tool which offers integration with development tools and/or forward/reverse engineering of the code.

* *Forward engineering* may make it easier to start the Build phase
* *Reverse engineering* (if it works reasonably – but unfortunately, it rarely does so) may make it easier to create the final *O0500 - Software Architecture* and to have updated class design for internal use as the project progresses.
* *Integration with development tools* makes it easier/quicker for developers to perform lookups in the class design and reduces the risk of failing to comply with the design. Moreover, it motivates developers to a greater extent to update the design regularly, thereby making it easier to create the final  *O0500 - Software Architecture* and to have updated class design for internal use as the project progresses.

Smaller projects can use Microsoft Visio for diagrams as a minimum, possibly supplemented with class diagrams created directly in the development tool (e.g. supported in Visual Studio) if documentation and prototyping are taking place simultaneously.

Microsoft Visio supports both forward and reverse engineering of C#/VB.NET code.

Another excellent modelling product which is not too complicated or expensive is Sparx Enterprise Architect, which has most of the features not available in Visio (teamwork, Visual Studio + Eclipse plugin, Java and .NET support).

Often it is also possible to do detailed design directly in the code. An example could be API design where it is often easier to implement a stub API with mock data than draw a design. The advantage is that the code can be easyly changed following a review and directly used in the following Build phase.

The same procedure can be used if the project uses a modern ORM-framework. Then updates to database models can be expressed directly in the related entity classes in the code.

If this approach is used, review can be done directly in pull requests in feature branches.

BUT remember to think and draw up the overall design before you start building.

## Selecting models for detailed design

It is necessary to consider which models are to be included in the detailed design:

* Analysis model – an object model that describes implementations of use cases and acts as an abstraction for the Design model
* Design model – an object model that describes implementations of use cases and acts as an abstraction for the Implementation model
* Implementation model – an identification of the physical parts of the implementation (files, including source code, data and executables)

One thing these models have in common is that they each try, via an enumerable set of elements, to represent a complete and comprehensive model of the system and its behaviour.

This involves a resource cost for the project for each model that it opts to document; the documentation of the model itself and the mapping between the models must be documented and maintained.

The following must be ensured when selecting the models

* The purpose of the models is clearly declared and apparent
* The models have an appropriate level of detail given the objectives above
* The models’ use of modelling mechanisms is appropriate in view of the problem
* The models are as simple as possible while also achieving their objectives
* The models must be capable of accommodating reasonable future changes

Often only the design and implementation models are relevant. Sometimes the Analysis model is provided by the customer as part of the tender.

The choice and form of models should be documented in *D0110 – Technical Design Guidelines*.

## Analysis model

The Analysis model consists primarily of Analysis classes and their associated artefacts, such as diagrams. The Analysis model may be a provisional product if it is being developed into the Design model, or it may continue to exist throughout the entire project – and possibly after the project as well – as a conceptual overview of the system. The Analysis model can be started as early as the Analysis phase for the project.

The Analysis model is optional

Analysis classes will be developed into Design classes in the Design model.

The objective for the Analysis model is to identify a mapping of (functional) requirements to a more developer-oriented model of the system. Refinement in terms of detail and precision takes place from the Analysis model when the project moves away from Analysis and through the Design activities. Consequently, the Analysis classes are flowing, open to changes and develop significantly before being crystallised in the Design model.

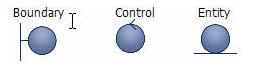
Consider whether a separate Analysis model is needed:

* a separate Analysis model may be useful when the system is to be designed for multiple target environments with separate design architectures (e.g. multiplatform). The Analysis model is a generalisation of the Design model which often omits details relating to specific technologies so as to be able to offer an overview of the functionality of the system
* The design of the solution is so complex that a simplified abstraction of the design is required in order to introduce the design to new project participants. Alternatively, a well-defined O0500 Software Architecture may serve the same purpose
* The extra work resulting from ensuring that the Analysis model and the Design model are consistent has to be balanced against the advantages of having a view of the system which depicts only the most important details. Maintaining a large amount of information between the Analysis model and the Design model may be very costly. A less ambitious approach would be only to maintain the most important domain classes in the Analysis model
* The value of the Analysis model dwindles quickly when it is no longer maintained. When the model is not maintained, it will cease to be useful as it will no longer precisely reflect the current design of the system. A decision to no longer maintain the Analysis may be appropriate (it has served its purpose), but this decision must be deliberate

If projects are complex/extensive, there may be too much of a gap between use cases and specific Design classes. In this case, the design may be initiated by implementing use cases via Analysis classes. As the Analysis model may be a temporary artefact in order to make the transition from use cases to Design classes, the Analysis model does not need to be maintained going forward – or otherwise be documented at all.

Analysis classes are rough abstractions, divided into three stereotypes:

* Boundary (user interface or integration interface),
* Control (a component that controls other classes) and
* Entity (a class representing data in the system that typically corresponds to an entity in the logical data model)



The Analysis classes are a rough division of business objects with appropriate encapsulation of responsibilities, but without taking into account technical details such as persistence, creation and discontinuation of objects.

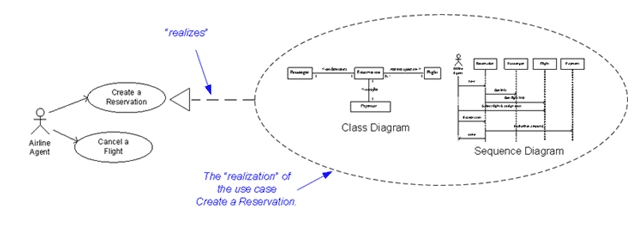
Typically, an Analysis class will be an abstraction of a collection of various Design classes.

### Use case realisations

Use case realisations are prepared by means of descriptive text. UML class diagrams illustrate the links between the Analysis classes included in the use case realisation. One or more UML interaction diagrams illustrate how the objects interact in order to implement the use case.

Sequence diagrams are typically the best choice, but collaboration diagrams can be used as well.

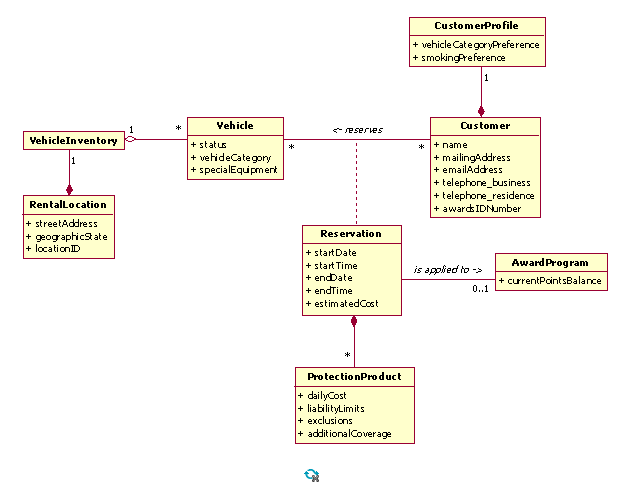
Each use case should typically be covered by one representative implementation, and consistent use cases can often be covered by selecting one representative use case realisation to which the others refer.



### Analysis class design

The various use case realisations are regularly collected and made consistent so that it is possible to create a collective Analysis model of classes and links between them. This collective description is made up of one or more class diagrams and descriptive text.

It is not necessary to identify precise attributes and data types for the Analysis classes, just as identification of methods for the Analysis classes is not the intention. Often this is documented in a concept model. The most important thing in this phase is to identify the significant business objects (including their obvious attributes) and relations between them.



## Design model (the actual class design)

Like the Analysis model, the Design model is a realization of functional scenarios expressed in user stories or use cases. The Design model essentially provides input for building and testing. The Design model is an extensive product that includes all Design classes and their associated artefacts. Unlike Analysis classes, Design classes are technology-specific and so accept responsibility for details such as persistence.

The Design model is mandatory and extensive and should be aligned with the projects technical architecture.

Like the Analysis model, the Design model is created by identifying the components, subcomponents, classes and methods of the system by means of use case realisations or user stories. The classes of the Design model are directly equivalent to classes/components to be implemented in the final system. Typically, all significant classes are identified as Design classes in the model prior to design work, while less significant classes/components are added on an ad hoc basis during the Build phase. It is not always relevant to update the detailed design during the build phase. Often the detailed design is just used as the starting point for the implementation.

It must be possible to comprehend, maintain and implement the Design model.

Consider, if relevant, whether the Design model is to be mapped to the Analysis model, and if so how.

Consider whether the Design model is to be mapped to the Implementation model, including code generation and reverse engineering, and if so how. Likewise consider if the Design model could actually be a build artifact.

### Use case realisation

As for the Analysis model, use case realisations or user stories are implemented by means of descriptive text and often a UML class diagram that illustrates the links between the Design classes included in the use case realisation. One or more UML interaction diagrams that illustrate how the objects interact in order to implement the use case. Sequence diagrams are typically the best choice, but collaboration diagrams can be used as well.

Again, each use case or user story should typically be covered by one representative realisation, and consistent use cases can often be covered by selecting one representative use case realisation. Use case realisations with Design classes – unlike Analysis classes – must explicitly illustrate creation (and, where appropriate, discontinuation) of objects, who is responsible for creating access to the necessary objects and hooking them up so that it is possible to service a given request, layering and detailed technology classes, i.e. manager classes and classes in front-end layers, data layers, logging, error handling, etc as defined by the software architecture.

An Analysis class may in practice end up becoming several Design classes, just as it may be practical to combine functionality from several Analysis classes into one Design class, although this is rarer.

### Class design

As in the case of the Analysis model, the various use case realisations or user stories are collected and made consistent so that a collective Design model of classes and links between them is created. This collective description is made up of one or more class diagrams and class descriptions, typically arranged and structured on the basis of continuous layers/components. Each individual layer/component begins with an overview of the classes involved (in the form of class diagrams with descriptive text).

Behaviour and responsibilities between the classes are described further as required, where necessary providing greater detail in the form of sequence diagrams, activity diagrams and design patterns used. Individual classes are described with systematic specification (typically in tabular form) of

* Name (mandatory)
* Brief description (mandatory)
* Class responsibilities (mandatory)
* Methods for the class (all public and also most significant) (to be described)
* Attributes for the class (all public and also most significant) (to be described)
* Specific conditions (as required)
* Comprehensive description of functionality in the form of status/sequence diagrams (as required)

Partitioning patterns are usually described in O0500. You should take the below into consideration when doing detailed design:

* The customer’s organisation – the system can be organised as a mirror of the customer’s organisation (e.g. partitioning matches departments). This pattern often occurs early on in the design because the existing (and, where applicable, mental) model of the business is organised in precisely this way. This pattern typically affects the top layers of the system
* Skills – the system can be partitioned according to skills in the development team (project). This often takes place in the lower layers of the system and reflects the specialisation in skills needed for development and support of technology infrastructure. Network and database administration are examples. This pattern can also be used in the upper layers if a specific skill in the problem domain is necessary in order to understand and support business functionality, e.g. creation of an insurance policy
* Distribution of the system – horizontal partitioning reflecting the physical distribution of functionality can take place in any layer of the system, e.g. solutions comprising an intranet and an extranet.
* Access levels – some systems, particularly ones requiring specific security access measures (e.g. staff vs. partners using an extranet) may require partitioning
* Variations – functionality that may be optional and supplied only in certain variants of the system must be organised as independent subsystems that can be developed and supplied independently of the mandatory functionality in the system.

## Implementation model

The Implementation model represents the physical composition of implementation via implementation elements, including files (including source code, executables) and data.

The Implementation model identifies the physical elements of the implementation so that they can be understood and managed more easily.

The Implementation model is optional. Projects often choose not to use documentation of the Implementation model as part of detailed design. Instead, they use file structure descriptions as defined by *DD160 - Programming Guidelines* and *D0140 – Configuration Management Plan*.

## Format

It is encouraged to create several DD130 documents as long as they are named DD130 – Detailed Design – [Subsystems])

**Note:**

The deliverable does not have to be prepared as a Word document. If the project using a dedicated UML tool (such as Sparx Enterprise or Visio) for the design, for example, the project may choose to use the UML model itself as the deliverable. However, for this it has to be possible to use the tool to create an overview and compile diagrams and text so that the model is self-explanatory. The deliverable may also be made up of a combination of a UML model and a reading guide/overview in Word, or it may be made up of combinations of Word documents and report extracts from UML tools.

The same is applicable to similar modelling tools for specific types of solution.

A decision on the form which the deliverable is to take should be made and documented in *D0110 - Technical Design Guidelines*.

## Review recommended

It is strongly recommended that the project request a review of this deliverable by a Netcompany architect before moving to implementation, as a sound detailed design is a significant factor for successful implementation.