**Netcompany – Methodology and Security**

**A0140 - Functional Scenarios**

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

.

|  |  |
| --- | --- |
| **Version:** | **2.1** |
| **Status:** | Approved |
| **Approver:** | Leni Støvring Barfred |
| **Author:** | Leni Støvring Barfred |



**Document history**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Date | Author | Status | Remarks |
| 1.0 | 18-09-2015 | Leni Barfred | Final |  |
| 2.0 | 04-05-2016 | Simon Shan Man Ng | Approved | 2016 New logo |
| 2.1 | 15-09-2020 | Leni Barfred | Approved | New colors + rewrite |

**References**

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

Table of contents

[1 Introduction 3](#_Toc51341909)

[1.1 Purpose 3](#_Toc51341910)

[1.2 Reference group 3](#_Toc51341911)

[2 Guideline to A0140 3](#_Toc51341912)

[2.1 User stories – hints 5](#_Toc51341913)

[2.1.1 User story content 6](#_Toc51341914)

[2.2 Use case hints 6](#_Toc51341915)

[2.2.1 Use case content 7](#_Toc51341916)

[2.3 Prioritisation of functionality 7](#_Toc51341917)

# Introduction

## Purpose

The purpose of this deliverable is to document the results of functional analysis, which is to be caried out BEFORE you start producing user interfaces.

The deliverable documents the needed functionality in the solution.

The deliverable may consist of one of the following, depending on the individual project and contract:

* + Epics
  + User stories
  + Use cases

## Reference group

The reference group primarily consist of the customer's business owners or product owners. However, everyone involved in the project is essentially part of the reference group.

# Guideline to A0140

For the majority of projects, passing on an understanding of the content and scope of the delivery from the customer to Netcompany presents a challenge, despite the fact there is a contract between the parties. Various reasons for that may be the case:

* The customer does not have the expertise to specify their wishes in a structured manner
* Requirements in the tender documentation are frequently not genuine requirements
* We do not have a complete knowledge of the customer's business. We do not necessarily understand the customer's domain/industry/processes
* The customer does not know what they want
* The customer has forgotten important requirements in the contract
* The customer is subject to a demand from their IT department crying *‘standard solution’*, which the business does not understand the consequences of
* The customer's requirements are ambiguous
* The customer's requirements conflict with one another

We want to define use cases or epics and user stories as a collective view of the functionality of the delivery.

Both methods offer a systematic and intuitive procedure to identify functional requirements, focusing on value for users. The formats are intuitive. There is no need for a complex form of notation. Natural language works best. Both methods are easy to read and comment on.

In an ideal situation, the use case model or epics and userstories will take precedence over the requirement specification after approval and can be used in later requirement mapping and negotiation. Unfortunately this is not always possible to achieve due to customer requirements and our development process becomes less agile than it could be.

It is desirable for use cases or epics to drive the subsequent development process in that design, implementation and testing use the A0140 as a reference.

A0140 is also extremely useful for dividing deliverables into iterations that delivers useful functionality to the customer at an early stage.

When a project is about to prepare A0140, the history of the project must be taken into consideration.

Depending on what is given in the contract, there are different purposes of A0140.

|  |  |
| --- | --- |
| Contained in contract/tender/offer | Purpose of A140 |
| descriptions of use cases, | refining and agreeing on the existing use cases |
| a traditional requirement specification | transforming requirements to functional scenarios in the form of use cases or epics/user stories that can be delivered in iterations. Ie. getting the requirement specification clarified to something users and we understand. |
| epics and user stories | refining and agreeing on the existing epics and user stories |
| do not include descriptions of use cases or a requirement specification | a full functional analysis of what is needed |

Use cases or epics and user stories are recommended instead of requirement specifications. If we start making up all kinds of nice functions and components without considering use cases or user stories, this will provide a list of functions that will not necessarily be related to what users need and it will not contribute to a positive business case for the customer. We have to know the users needs.

How do we know whether these functions are important, or whether they can be used at all? Whom do they help? Which process do they support? How much value will they add to the business?

One of Netcompany's roles in the preparation of functional scenarios is to facilitate discussion and help the customer to communicate their needs by identifying the functionality that will support all types of users in their different roles.

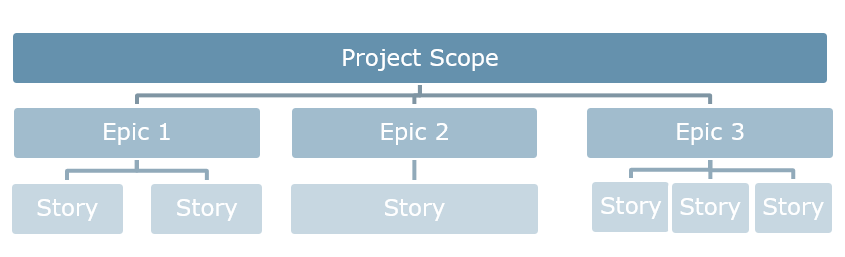
The process of identifying the functionality that is needed is quite simple if caried out in a systematic way:

* Identify the actors that will be using the solution. Remember that external systems and possibly “time” may be actors.
* Ask the customer what each actor needs the solution to do
* Prioritize the functionality. Not all functional scenarios are equally important.
* It is also our job to identify the non-functional requirements that often have significant influence over the architecture and user satisfaction. Non-functional requirements are not always formulated clearly by the customer. Non-functional requirements can be described together with the functional scenario they relate to or be maintained in a separate list.
* Remember to discuss topics such as:
  + user-friendliness
  + reliability
  + performance
  + maintenance requirements
  + legal requirements
  + legislative requirements
  + logging requirements
  + security requirements
  + standards like GDPR, ISO
  + infrastructure requirements

The analysis is considered complete when it captures all essential functional requirements correctly in a manner that can be understood by the customer, the users and the developers. Design, build and test can subsequently be planned and coordinated in iterations.

## User stories – hints

In the functional analysis the full scope is broken down into Epics, user stories and possibly into subtasks.



Agile litterature emphasises that user stories are verbal and not written. That they are not document but short sentences written on a post-it. Remember, that in Netcompany we choose to document our functional analysis in a more thorough way than the traditional agile ‘schools’. To write a user story on a post-it and hang it on a wall does not work, when the team might be spreas across Copenhagen, Ho Chi Minh and Warsaw. Besides we often need the customer to approve the user story, which requires a digital representation. We also need to visualize progress on the analysis and design discipline which is done by ‘counting’ number of approved user stories.

The principles behind user stories are well documented else where like here <https://www.agilebusiness.org/content/requirements-and-user-stories>

A user story consist of:

*As a < role>  
  
I need <requirement or feature>  
  
So that <goal / value>*

This could be interpreted as:

*As a Marketing Director,  
  
I need to improve customer service  
  
So that we retain our customers*

### User story content

Bill Wake’s INVEST model describes how to write a good user story.

|  |  |
| --- | --- |
| Independent | Stories should be as independent as possible from other stories, to allow them to be moved around with minimal impact and potentially to be implemented independently. If stories are tightly dependent, consider combining them into a single user story. |
| Negotiable | Stories are not a contract. They are “placeholders” for features which the team will discuss and clarify near to the time of development. |
| Valuable | Stories should represent features providing clear business value to the user/owner of the solution and should be written in appropriate language. They should be features, not tasks. |
| Estimable | Stories need to be clear enough to estimate (for the appropriate timeframe), without being too detailed. |
| Small | Stories should be small enough to be estimated. Larger “Epic” stories should be broken down into smaller User Stories as the project progresses. The stories after splitting still follow the INVEST criteria. |
| Testable | Stories need to be worded clearly and specifically enough to be testable. |

A good user story is clear, precise and complete. Check if it adheres to the following:

* It does not combine or overlap other user stories
* It is not in confict with other user stories.
* It respects standards and policies
* It can be traced back to a business requirement in the business case for the project

More user stories make up an Epic (or the other way around – an epic i broken down into more user stories).

Please find a presentation that explains the methodology here: [https://goto.netcompany.com/cases/GTO129/NCMETODE/Tools/User%20stories%20-%20guideline.pptx](https://goto.netcompany.com/cases/GTO129/NCMETODE/Tools/Definition,%20guidelines%20and%20tips%20for%20writing%20good%20user%20stories.pptx)

*Please remember that user stories rarely depicts the entire truth. They usually only covers the functional scenarios. Technical disciplines and non-functional requirements/needs have to be designed in th D- and O-deliverables.*

## Use case hints

The best use cases are the ones that add the most value for the customer. For instance, use the boss test: Would your boss like you to implement this use case all day long?

Use cases with negative value or which permit users to do things they should not be capable of doing are not use cases: they are abuse cases.

Use cases that have almost no value or are used very rarely are redundant, and so are useless cases. Examples: "Login via Windows Authentication, "Navigate in the portal.

Use cases are typically technology-agnostic initially. This means that they focus on system requirements, and not on how these requirements are implemented.

All actors plus use cases for a system go to make up the use case model.

Use case diagrams are used to illustrate a section from the use case model. A use case diagram shows the associations between a subset of actors and use cases

Use cases can be designed in many ways. The most important thing is for the customer to understand them.

### Use case content

The following list includes the typical information included in the description of a use case. Choose the content suitable for the project:

* Name – Name of the use case
* Description – Description of the use case, ideally with an idea of the value
* Actors – Which actors are involved in this use case
  + Are there any specific security conditions that must be present for a given actor to participate in a use case involving one or more data entities
* Stakeholders – Which stakeholders (actors) does this use case have, and what is the value of the use case to the stakeholder?
* Scenario – The scenario describes, in steps, how a use case is implemented
* Special requirements – Special requirements for the use case, typically non-functional requirements
* Prior conditions/Initial conditions – Requirements that have to be met before use cases can be implemented
* Concluding conditions/Success criteria – What has been achieved by implementing a use case?
* Frequency – Often is the use case implemented?
* Relationships with other use cases – Does this use case use or expand upon other use cases?
* Activity diagrams – Activity diagrams are a good visual tool for major scenarios with branches
* Relationship to use case diagram – Use cases should exist in at least one use case diagram

## Prioritisation of functionality

User stories, use cases and old fashioned requirements must be prioritized to ensure, that the most important functionality is implemented to begin with.

The MoSCoW prioritisation method can be useful: <https://www.agilebusiness.org/content/moscow-prioritisation>