TECHNICAL REPORT

ISO/IEC TR 7052

First edition
2023-09

Software engineering — Controlling frequently occurring risks during development and maintenance of custom software

Ingéniérie du logiciel — Contrôle des risques fréquents au cours du développement et de la maintenance d'un logiciel sur mesure

Reference number
ISO/IEC TR 7052:2023(E)

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC had not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 7, *Software and systems engineering*.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Introduction

Information and communication technology (ICT) projects run many risks. ICT projects often have to contend with delay, budget overruns, and an end result of low quality.

ICT projects in which custom software is developed and/or maintained often run extra risks, on top of the risks that are part and parcel of ICT projects in general[31]. This seems to be caused by the sheer size and complexity of such custom software projects, and by a failure to mitigate the risks inherent to software development in general, despite the fact that they are well known, and that there are suitable controls for their management.

This document describes controls for some of the risks inherent in custom software development. The purpose of this document is that during the development of custom software stakeholders can avail themselves of a collection of suitable controls. The controls included are common of themselves, making this collection of controls a logical starting point for assuring the quality of custom software development. Controls were selected for inclusion based on the experience and opinion of the subject matter experts contributing to this document.

Two target groups are important when mitigating risks during the development of custom software:

a) the software development acquirers and suppliers;

b) the end users and maintainers of the software developed.

This document details risks and controls specific to custom software development. Risk management in the context of software development is covered in ISO/IEC/IEEE 12207 and its elaboration standard ISO/IEC/IEEE 16085. Generic risk management is covered by ISO 31000 and its related standards.

This document is based on NPR 5326 developed by Royal Netherlands Standardization Institute Foundation (NEN, https://www.nen.nl/).

Software engineering — Controlling frequently occurring risks during development and maintenance of custom software

1 Scope

This document:

— describes frequently occurring risks during development and maintenance of custom software;

— describes possible controls for frequently occurring risks;

— describes the related:

  — activities, facilities and roles typically used for these controls;

  — properties of products and processes;

  — standards, measurements, testing and assessment of the properties of products and processes.

2 Normative references

There are no normative references in this document.

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

ISO and IEC maintain terminology databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1
acceptance test-driven development
ATDD
development method where team members with various backgrounds [*developers* (3.18), testers and business analysts] jointly write the acceptance tests prior to development of the relevant functionality

Note 1 to entry: Although the name of the method, acceptance test-driven development, suggests that ATDD is a specialization of *test-driven development* (3.46) (TDD), this is not the case. Rather, TDD focuses on driving development by writing *unit tests* (3.48) first and ATDD focuses on driving development by writing acceptance tests first.

[SOURCE: NEN NPR 5326:2019, 3.1, modified —Added note 1 to entry.]

3.2
acquirer
stakeholder that acquires or procures a product or service from a *supplier* (3.44)

Note 1 to entry: Other terms commonly used for an acquirer are buyer, customer, owner, purchaser or internal/organizational sponsor.

[SOURCE: ISO/IEC/IEEE 12207:2017, 3.1.1]

3.3
agile development
development approach based on iterative development, frequent inspection and adaptation, and incremental deliveries in which requirements and solutions evolve through collaboration in cross-functional teams and through continuous stakeholder feedback

Note 1 to entry: Any use of the word "agile" in this document refers to methodology.

[SOURCE: ISO/IEC/IEEE 26515:2018, 3.1, modified — In note 1 to entry, removed the reference to ISO/IEC/IEEE 26515:2018, Annex A.]

3.4
backlog
collection of agile features or stories of both functional and non-functional requirements that are typically sorted in an order based on value priority

Note 1 to entry: This can be used as a list of product requirements and *deliverables* (3.13) not part of current work, to be prioritized and completed.

[SOURCE: ISO/IEC/IEEE 26515:2018, 3.4, modified — Removed note 1 to entry.]

3.5
behaviour-driven development
BDD
development method where team members with various backgrounds [*developers* (3.18), testers and business analysts] jointly describe the behaviour of the intended functionality prior to development of the relevant functionality

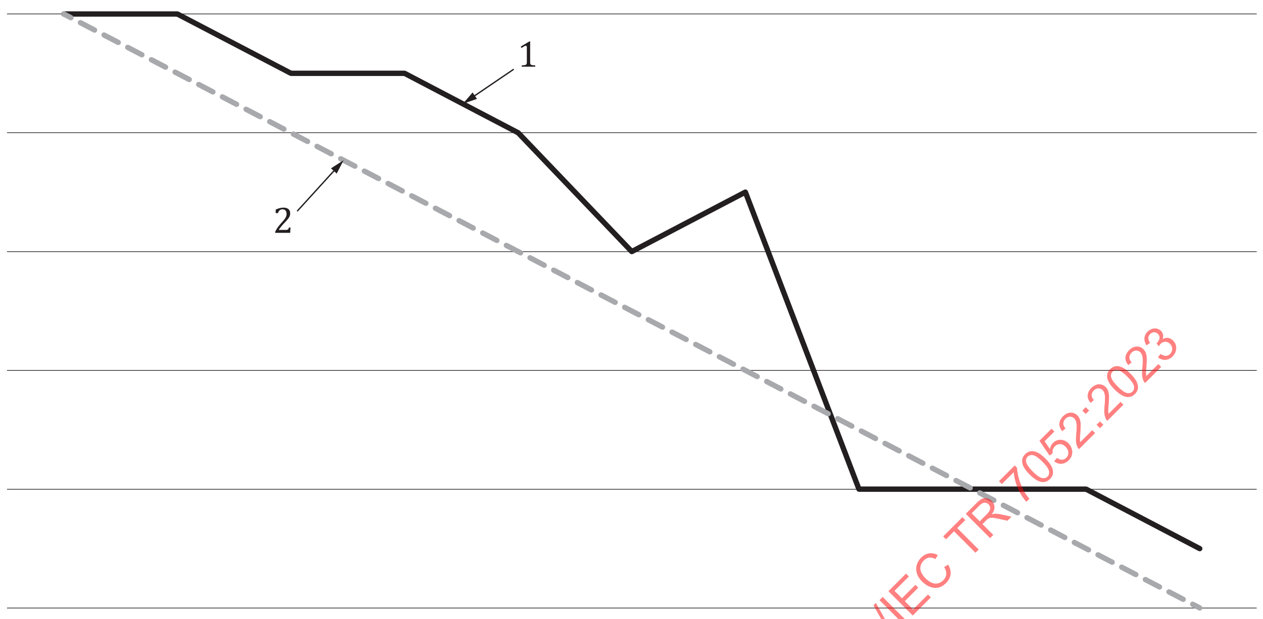
[SOURCE: NEN NPR 5326:2019, 3.5]

3.6
burndown
indicator of the work completed and estimate of remaining work to be completed or remaining effort needed to complete a product development iteration cycleNote 1 to entry: Work is measured as all work done to deliver story points, stories, features, functions, *function points* (3.25), *user stories* (3.50), *use cases* (3.49), or requirements during a product development iteration.

[SOURCE: ISO/IEC/IEEE 24765:2017, 3.437, modified — Restructured into definition and note 1 to entry.]

3.7
burndown chart
graph that represents the work remaining to do on a project

Note 1 to entry: See Figure 1 for an example of a burndown chart.



Key

1 solid line, representing the actual work-remaining

2 stippled line, representing the ideal burndown of the work

Figure 1 — Example of a burndown chart with time on the horizontal axis and work-remaining on the vertical axis

Note 2 to entry: This can be used as a chart to show the amount of the work done versus the amount of the work still to do.

Note 3 to entry: This can be presented per *sprint* (3.42), as well as per release or iteration.

Note 4 to entry: For example, user *story points* (3.51) or *function points* (3.25) can be used to measure the amount.

[SOURCE: ISO/IEC/IEEE 26511:2018, 3.1.6, modified — Added the example figure and notes to entry.]

3.8
business impact analysis
process of analysing the impact over time of a disruption on the organization

Note 1 to entry: The outcome is a statement and justification of business continuity requirements.

[SOURCE: ISO 22300:2021, 3.1.24]

3.9
code review
activity where one or more *developers* (3.18) establish the *quality* (3.35) of (part of) the *source code* (3.41) by going through it

Note 1 to entry: There are a variety of ways to conduct code reviews which range from formal to very informal and from discrete to continuous.

[SOURCE: NEN NPR 5326:2019, 3.10, modified — Modified definition and added note 1 to entry.]

3.10
configuration management database
CMDB
database that is used by an organization to store information about the hardware and software in use

[SOURCE: NEN NPR 5326:2019 3.11]

3.11
consequence
outcome of an *event* (3.23) affecting *objectives* (3.30)

Note 1 to entry: An event can lead to a range of consequences.

Note 2 to entry: A consequence can be certain or uncertain and can have positive or negative effects on objectives.

Note 3 to entry: Consequences can be expressed qualitatively or quantitatively.

Note 4 to entry: Initial consequences can escalate through knock-on effects.

[SOURCE: ISO Guide 73:2009, 3.6.1.3]

3.12
control
measure that is modifying *risk* (3.37)

Note 1 to entry: Controls include any process, policy, device, practice, or other actions which modify risk.

Note 2 to entry: Controls cannot always exert the intended or assumed modifying effect.

[SOURCE: ISO Guide 73:2009, 3.8.1.1, modified — In note 2 to entry, changed "may not" to "cannot".]

3.13
custom software
software product developed for a specific application from a user requirements specification

[SOURCE: ISO/IEC 25000:2014, 4.3]

3.14
data protection impact assessment
DPIA
tool described in the *General Data Protection Regulation* (3.27) (GDPR) to assess in advance the privacy risks of data processing and then to be able to implement *controls* (3.12) to reduce the *risks* (3.37)

Note 1 to entry: See https://gdpr.eu/article-35-impact-assessment/.

[SOURCE: NEN NPR 5326:2019 3.12, modified — Changed "take measures" to "implement controls". Added note 1 to entry with link to the GDPR Article 35.]

3.15
deliverable
any unique and verifiable product, result, or capability to perform a service that must be produced to complete a process, phase, or project

[SOURCE: ISO/IEC/IEEE 24765:2017, 3.1098, definition 1]

3.16
Delphi method
information-gathering technique used as a way to reach consensus of experts on a subject

Note 1 to entry: The Delphi method is applied as consensus tool for determining weights of indicators/sub-indicators in this document.

Note 2 to entry: A facilitator uses a questionnaire to solicit ideas about the important project points related to the subject. The responses are summarized and are then recirculated to the experts for further comment. Consensus can be reached in a few rounds of this process.

[SOURCE: ISO/IEC/IEEE 24765:2017, 3.1102, modified — Restructured into definition and notes to entry.]

3.17
design thinking
methodology for solving (very complex) problems where these are defined from the human needs

Note 1 to entry: In design thinking solutions are determined with brainstorming sessions, where *prototypes* (3.34) are produced to test the intended solution in practice.

[SOURCE: NEN NPR 5326:2019, 3.15]

3.18
developer
individual or organization that performs development activities (including requirements analysis, design, testing through acceptance) during the system or software life-cycle process

Note 1 to entry: Activities of the developer of *custom software* (3.13) include setup and analysis of functional and non-functional system and software requirements, design, programming and testing.

Note 2 to entry: Designers, programmers and testers are therefore all developers.

[SOURCE: ISO/IEC 25000:2014, 4.6. modified — Added notes to entry.]

3.19
development pipeline
build pipeline
set of tools aimed at the managed and controlled build of a package with which an application can be taken into use

Note 1 to entry: As part of this various tests are carried out to determine and assess *quality* (3.35).

[SOURCE: NEN NPR 5326:2019, 3.30]

3.20
development team
team that develops and/or maintains software

Note 1 to entry: The *acquirer* (3.2), *product owner* (3.33) and future maintainer can form part of the development team.

Note 2 to entry: Development teams can be broken down by function (e.g. a team of designers, a team of programmers, a team of testers) or be multidisciplinary (each team has e.g. design, programming and test expertise).

[SOURCE: NEN NPR 5326:2019, 3.31]

3.21
DevOps
set of principles and practices which enable better communication and collaboration between relevant stakeholders for the purpose of specifying, developing, and operating software and systems products and services, and continuous improvements in all aspects of the life cycle

[SOURCE: IEEE 2675-2021 3.1]

3.22
distributed denial of service attack
DDoS attack
unauthorized access to a system resource or the delaying of system operations and functions in the way of compromising multiple systems to flood the bandwidth or resources of the targeted system, with resultant loss of availability to authorized users

[SOURCE: ISO/IEC 27039:2015, 3.7]

3.23
event
occurrence or change of a particular set of circumstances

Note 1 to entry: An event can be one or more occurrences and can have several causes.

Note 2 to entry: An event can consist of something not happening.

Note 3 to entry: An event can sometimes be referred to as an ‘incident’ or ‘accident’.

Note 4 to entry: An event without *consequences* (3.11) can also be referred to as a ‘near miss’, ‘accident’, ‘near hit’ or ‘close call’.

[SOURCE: ISO Guide 73:2009, 3.5.1.3]

3.24
extreme programming
XP
form of agile software development where procedures for iterative planning and working are combined with technical procedures, such as test-driven design and pair programming

[SOURCE: NEN NPR 5326:2019, 3.20]

3.25
function point
FP
unit of measure for functional size

[SOURCE: ISO/IEC 20926:2009, 3.35, modified — removed “as defined within this International Standard”.]

3.26
function point analysis
FPA
method for measuring functional size

[SOURCE: ISO/IEC 20926:2009, 3.36, modified — removed “as defined within this International Standard”.]

3.27
General Data Protection Regulation
GDPR
European Regulation that standardizes the rules for processing personal data by private businesses and government bodies in the whole European Union

Note 1 to entry: See https://gdpr.eu.

[SOURCE: NEN NPR 5326:2019, 3.3]

3.28
integration testing
testing in which software components, hardware components, or both are combined and tested to evaluate the interaction between them

[SOURCE: ISO/IEC/IEEE 24765:2017, 3.2034, modified — Removed note 1 to entry.]

3.29
minimum viable product
MVP
version of a work product with just enough features and requirements to satisfy early customers and/or provide feedback for future development

[SOURCE: IEEE 2675-2021, 3.1]

3.30
objective
predetermined result to be achieved

Note 1 to entry: To achieve an objective several tools and activities are generally needed, one of which can be *custom software* (3.13).

[SOURCE: NEN NPR 5326:2019, 3.18]

3.31
performance test
test type conducted to evaluate the degree to which a test item accomplishes its designated functions within given constraints of time and other resources

Note 1 to entry: A distinction is often made between load, stress and endurance tests. Load tests simulate a normal load on the system. Stress tests allow the load to increase to determine the maximum load at which the system still functions. Endurance tests load the system for a longer period in order to discover memory leaks or other problems which only manifest themselves after some time.

[SOURCE: ISO/IEC/IEEE 29119-1:2022, 3.58, modified — Changed the term from "performance testing" to "performance test"; changed "type of testing" to "test type"; added note 1 to entry.]

3.32
product breakdown structure
PBS
decomposition of the product into its components

Note 1 to entry: The PBS begins with the complete product at the top of the hierarchy and below this the main components, which can also each be broken down again into components, etc.

[SOURCE: ISO 21511:2018, 3.7, modified — Added the abbreviated term and note 1 to entry.]

3.33
product owner
stakeholder responsible for the capabilities, acceptance and use of a product

Note 1 to entry: The product owner shares the product vision, required features and their priorities, and acceptance criteria.

[SOURCE: ISO/IEC TR 24587:2021, 3.12]

3.34
prototype
model or preliminary implementation suitable for evaluation of system design, performance, and production potential, or for better understanding or determination of the requirements

[SOURCE: ISO/IEC 2382:2015, 2122670, modified — Removed notes to entry.]

3.35
quality
ability of a product, service, system, component, or process to meet customer or user needs, expectations, or requirements

Note 1 to entry: For the quality of software and IT systems, ISO/IEC 25010 offers a breakdown into aspects, such as:

— reliability;

— security;

— usability;

— functional suitability;

— maintainability;

— portability;

— performance efficiency;

— compatibility.

[SOURCE: ISO/IEC/IEEE 24765:2017, 3.3259, definition 2, modified — Added note 1 to entry.]

3.36
regression test
test type used following modifications to a test item or to its operational environment, to identify whether regression failures occur

[SOURCE: ISO/IEC/IEEE 29119-1:2022, 3.64, modified — Changed to term from "regression testing" to "regression test"; changed "testing performed" to "test type used"; changed "failures in unmodified parts of the test item" to " regression failures": removed notes to entry.]

3.37
risk
effect of uncertainty on *objectives* (3.30)

[SOURCE: ISO Guide 73:2009, 1.1, modified — Notes to entry removed]

3.38
risk treatment
process to modify a *risk* (3.37)

Note 1 to entry: Risk treatment can involve:

— avoiding the risk by deciding not to start or continue with the activity that gives rise to the risk;

— taking or increasing risk in order to pursue an opportunity;

— removing the risk source;

— changing the likelihood;

— changing the *consequences* (3.11);

— sharing the risk with another party or parties (including contracts and risk financing and;

— retaining the risk based on an informed decision.

Note 2 to entry: Risk treatment that deal with negative consequences is sometimes referred to as ‘risk mitigation’, ‘risk elimination’, ‘risk prevention’ or ‘risk reduction’.

Note 3 to entry: Risk treatment can create new risks or modify existing risks.

[SOURCE: ISO Guide 73:2009, 3.8.1]

3.39
Scrum
iterative project management framework used in *agile development* (3.3), in which a team agrees on development items from a requirements *backlog* (3.4) and produces them within a short duration of a few weeks

3.40
security test
test type conducted to evaluate the degree to which a test item, and associated data and information, are protected so that unauthorized persons or systems cannot use, read, or modify them, and authorized persons or systems are not denied access to them

Note 1 to entry: Depending on the amount of information on the system to be tested that the security tester has when carrying out the test, the terms black box (minimum information), grey box (the same knowledge and access as an ordinary user) or white box security tests [access to the whole system, documentation and *source code* (3.41)] are used.

[SOURCE: ISO/IEC/IEEE 29119-1:2022, 3.74, modified — Changed to term from "security testing" to "security test"; added note 1 to entry.]

3.41
source code
computer instructions and data definitions expressed in a form suitable for input to an assembler, compiler, or other translator

[SOURCE: ISO/IEC/IEEE 24765:2017, 3.3882, modified — Removed note 1 to entry.]

3.42
sprint
time box of one month or less during which a 'done', usable and potentially releasable product increment is created

[SOURCE: NEN NPR 5326:2019, 3.44]

3.43
story mapping
making of a two-dimensional representation of (part of the) *backlog* (3.4) that is a visual aid for the user to cut up, group and arrange stories logically in the backlog, which gives an overview of the relations between all the *user stories* (3.50)

[SOURCE: NEN NPR 5326:2019, 3.45, modified — Changed the term from "story map" to "story mapping".]

3.44
supplier
organization or an individual that enters into an agreement with the *acquirer* (3.2) for the supply of a product or service

Note 1 to entry: Other terms commonly used for supplier are contractor, producer, seller, or vendor.

Note 2 to entry: The acquirer and the supplier sometimes are part of the same organization.

[SOURCE: ISO/IEC/IEEE 12207:2017, 3.1.60]

3.45
technical debt
deferred cost of work not performed at an earlier point in the product life cycle

Note 1 to entry: *Suppliers* (3.44) are often not able to explain properly to *acquirers* (3.2) why combatting and avoiding technical debt is an important point to consider in the development of *custom software* (3.13), which can make it difficult to avoid and deal with technical debt.

EXAMPLE:

— copying existing functionality to add a new function quickly without the resolving the duplication of *source code* (3.41) that has arisen here;

— not including certain test cases in an automated *regression test* (3.36);

— not upgrading libraries or frameworks used to a more recent version;

— not setting up and performing integration testing with systems to be linked;

— not updating documentation;

— not making domain concepts used in the source code explicit (e.g. passing on ‘strings’ with street and post code instead of creating a class for the concept ’address’).

[SOURCE: ISO/IEC/IEEE 24765:2017, 3.4181, modified — Added note 1 to entry and EXAMPLE.]

3.46
test-driven development
TDD
development method where the *developers* (3.18) write *unit tests* (3.48) prior to implementing the relevant functionality

[SOURCE: NEN NPR 5326:2019, 3.47]

3.47
T-shirt sizing
method of making relative estimates by comparing *user stories* (3.50) and dividing these into the categories extra-small, small, medium, large and extra large

Note 1 to entry: T-shirt sizing clarifies the interrelationships without time being wasted on false precision.

[SOURCE: NEN NPR 5326:2019, 3.48]

3.48
unit test
testing of individual routines and modules by the *developer* (3.18) or an independent tester

[SOURCE: ISO/IEC/IEEE 24765:2017, 3.4429 definition 1]

3.49
use case
description of behavioural requirements of a system and its interaction with a user

Note 1 to entry: A use case describes the users' goal and the requirements including the sequence of interactions between users and the system.

[SOURCE: ISO/IEC/IEEE 26515:2018, 3.15]

3.50
user story
short description of what a user intends to achieve with part of the system

Note 1 to entry: A user story normally consists of a few sentences of ordinary spoken language of the *acquirer* (3.2)/user which states what the user does or will be able to do to achieve a certain *objective* (3.30). In addition, a user story describes what acceptance criteria are used to consider the functionality produced as complete.

Note 2 to entry: A commonly used format for user stories is: ‘As <role> I can <activity> so that <purpose>’. For example: ‘As an employee I can specify my expenses so that I have these reimbursed by my employer’.

3.51
user story point
relative measure for the scope of *user stories* (3.50), often used in *agile development* (3.3) methods

[SOURCE: NEN NPR 5326:2019, 3.52]

4 Abbreviated terms

ICT	information and communication technologies
NEN	Royal Netherlands Standardization Institute Foundation
SEI	Software Engineering Institute

5 Explanatory note on terms

5.1 The term risk

Risk is the effect of uncertainty on achieving objectives. An effect is a deviation from the expectation – positive and/or negative. A risk is often characterised by reference to possible events and consequences, or a combination of these.

In the context of this document the term risk is interpreted as follows:

Uncertainty is caused by the chance of events occurring. There is only a risk if there is a chance of events occurring that affect achieving objectives. If the chance of such events occurring is zero, there is no risk. For example, for software containing no data that can be traced to persons, there is also no chance of such data being stolen. If the chance of the relevant event occurring is one (i.e. 100 %), there is also no risk. The event does occur with certainty.

An example of an event with an effect on achieving an objective is 'the systems to be linked are not modified on time'. The chance of this event can be determined in various ways. One way is extrapolation of historical data. Say that of the 30 projects that an organization has carried out that involve linking systems to be modified, 13 were found not to have been modified on time. A (naive) probability calculation then gives a chance of $13/30 \cdot 100\ \% = 43\ \%$ that a project will have to deal with systems to be linked that have not been modified on time. Of course, a risk assessment that considers more factors, such as the number of systems to be linked, is more accurate.

In addition to the chance of an event occurring, it is at least as important to know what the effect is on achieving objectives. An event that has no effect on the achievement of an objective does not form a risk. If the cabinet falls, but this has no effect on realising the project objective because it is not a government project, the event does not form a risk. An example of an event that does have an effect on achieving objectives is unauthorized reading of data from the system developed. Such an event can have several effects: the privacy of users is infringed, but also the reputation of the relevant organizations can be damaged, and penalties imposed.

The effects relate to different objectives; in this case safeguarding the privacy of users and having a reliable reputation.

Often there is only a risk if an event occurs which is certain to have a negative consequence on achieving an objective. In the case of an event with a positive effect this is generally called an opportunity. In complex (joint ventures of) organizations an apparently positive effect is also regarded as a risk. Examples here are delivering functionality too early, so the organization that has to use this product is not ready for it.

5.2 The term control

A control is a measure by which a risk is modified.

In the context of this document the term control is interpreted as follows: A control modifies the chance of a risk occurring or changes the effect of a risk occurring, or both.

An example of a control for the event 'a distributed denial of service (DDoS) attack occurs on the system' is placing a hardware appliance in the IT infrastructure that filters network traffic and so reduces the effect of the attack.

An example of a control that would reduce the chance of a DDoS attack is disallowing direct access to the system via the Internet, and only allow access via an intranet or virtual private network (VPN).

Note that a control can also consist of the omission of acts. For example, take the risk that the efficiency of a development team is negatively affected because the acquirer often interrupts the work of the development team with new wishes to be dealt with or faults to be repaired immediately. This risk can be reduced by agreeing that the development team divides the time into time boxes (in some software development approaches referred to as sprints) and also agreeing that the acquirer cannot alter the list of tasks on which the team is working during the current time box.

Controls can change risks in various ways. This is also called risk treatment. Examples of risk treatment in the domain of custom software are as follows.

— Avoidance: the risk is avoided by not carrying out the activity that gives rise to the risk. For example, by a system not downloading data via an interface from another system but storing all the data in its own database, the risk is avoided that the other system cannot be accessed.

— Removal: the risk cannot occur because the control removes a risk source. For example, by not building a function for which clear acceptance criteria have been formulated, the risk is removed that the function does not meet the (implicit) requirements and wishes of the acquirer.

— Changing the likelihood: the control affects the likelihood of the risk occurring. For example, by developing and running automated regression tests, the likelihood of not detecting faults in the software in time becomes smaller.

— Changing consequences: the control affects the consequences of the risk occurring. For example, by issuing a new version of the software for part of the users instead of for all the users at the same time, the consequences of any faults in the software are limited to that part of the users.

— Sharing: the control ensures that the risk is shared with one or more other parties. For example, developing software together with other organizations in a joint venture reduces the risk for each of the participating organizations.

— Acceptance: the control consists of the justified maintenance of the risk (acceptance). The risk of performance problems can for example be accepted if the number of intended users and the amount of data to be processed is small.

— Transfer: the control consists of the transfer of the risk to another party. The risk of the development environment not being available can for example (partly) be transferred to another party by purchasing the development environment as a service from the relevant party.

6 Risks

6.1 General

This clause describes commonly occurring risks during the development and maintenance of custom software. The risks described in this document are based on:

— the risk taxonomy from taxonomy-based risk identification [CMU/SEI-93-TR-06];

— the contribution of experiences with the development and maintenance of custom software of the experts within the NEN standards committee;

— input from the field in the form of review comments during the drafting of NEN NPR 5326.

Annex A cross-references the risks and controls described in this document. Annex B provides one method of performing a risk inventory or risk assessment.

6.2 Product-related risks

6.2.1 General

Product-related risks arise from work that is necessary to develop and/or maintain the custom software.

6.2.2 Risk 01: The quality of the software is reduced because of modifications to it

Making changes to software involves the risk of the introduction of new faults or a re-occurrence of previously repaired faults. Faults can occur due to incorrect assumptions or omissions in updating the software, as well as because a more recent version of a software library is used.

Faults can emerge in the form of reductions in quality immediately which can be observed by users, such as functions that do not work, lower performance or poorer usability, as well as in the form of reductions in quality not immediately observed by users, such as reduced security or maintainability. See ISO/IEC 25010 for a summary of various software quality aspects.

This type of regression often leads to dissatisfied acquirers and users, and many other types of damage (injury, financial damage, reputation damage, etc.) but regression can also have a potential secondary effect: it can lead to parties involved in the development and/or maintenance of the software becoming reluctant to release software and take new versions into use. This makes it more difficult to implement a number of controls that this document recommends for other risks. The importance of mitigating the risks of regression is therefore greater than it would appear.

This risk can be reduced by applying the following controls:

— Control 09: Setting up automated development pipeline.

— Control 10: Constantly meeting the requirements with regression tests.

— Control 13: Using a quality-driven development method.

— Control 15: Proper handover.

6.2.3 Risk 02: The quality of the software is reduced because of modifications to the environment in which it runs

Even without software being changed its quality can be reduced or it can stop working properly. Without the source code of the software being altered, faults can occur, or faults can come to light. This phenomenon is also called software decay and can often be explained by changes to the environment in which the software is run.

Examples of changes in the environment that affect software quality are:

— newly discovered security vulnerabilities in libraries used that make the software more insecure;

— interfaces of other systems that change so that the software can no longer download data from these systems;

— a new version of an operating system;

— more recent versions of browsers on the computers of end users;

— change in use of the software which brings to light faults that were not previously visible;

— network settings that change.

Most controls that help reduce the risks of changes in software also help here, provided they are used regularly, and not only when the software is changed.

This risk can be reduced by applying the following controls:

— Control 09: Setting up automated development pipeline.

— Control 10: Constantly meeting the requirements with regression tests.

— Control 13: Using a quality-driven development method.

6.3 Project-related risks

6.3.1 General

Project-related risks arise from the environment in which the development and maintenance of custom software takes place.

6.3.2 Risk 03: Planned functionality is not completed on time because of underestimation of the amount of work involved

Acquirers often need certain functionality to be ready at a specific time, for example because legislation or policy changes with effect from a specific date. However, estimating the amount of work necessary to implement a certain piece of functionality is known to be difficult. For custom software (with the exception of an existing system being rebuilt) this often involves creating something that does not yet exist. There is therefore a risk that the estimation of the amount of work is too low, with the consequence that the functionality is not completed on time. This can bring about various consequences, such as team members not being available after the previously planned end date, additional work costs for suppliers or the perception of a failed project, with termination as the ultimate consequence. Not completing the functionality on time can lead to publicity damage, political damage or loss of turnover.

The later that it is discovered that the required functionality will not be ready at the agreed time, the more difficult that it is for an acquirer to implement alternative solutions.

Furthermore, over-estimating the amount of work is also a risk. This can lead to unnecessary deployment of resources and wasting time and other resources.

This risk can be reduced by applying the following controls:

— Control 01: Identifying and involving stakeholders.

— Control 04: Product breakdown into incrementally deliverable parts with business value.

— Control 05: Identifying, clarifying and systematically resolving technical debt.

— Control 06: Exploring solutions.

— Control 07: Incremental delivery of the product.

— Control 08: Iterative development approach.

— Control 11: Progress monitoring with burndown charts.

6.3.3 Risk 04: The product is not delivered on time and within budget because the scope is changed

During the development of custom software scope changes can be made. There are various possible reasons for this. External events can affect the required functionality and require that adjustments are made. Incremental delivery and taking the software into use can produce new insights. A failure to adjust the existing scope before adding new items often leads to delivery being delayed and budgets being overrun. Failure to drop functionality can also lead to a higher work pressure within the development team, again putting pressure on quality.

This risk can be reduced by applying the following controls:

— Control 04: Product breakdown into incrementally deliverable parts with business value.

— Control 07: Incremental delivery of the product.

— Control 12: An official product owner with mandate.

6.3.4 Risk 05: The software does not meet the requirements laid down because the team does not have the required expertise

For the development of software many different types of expertise are necessary, for example knowledge of the application domain, knowledge of the chosen technology, knowledge of related systems and specific specialist knowledge, such as knowledge of security, performance and quality of use.

If the required expertise is not present within the team at times when it is needed, or if people with the required expertise leave the team, the risk arises that the software delivered does not meet the stated quality requirements. In addition, over the course of time a team also builds up shared knowledge and expertise of the custom software under development. As a result, it can be difficult to guarantee continuity of knowledge and expertise if people leave.

If the team does not have the right technical knowledge, there is the risk that the wrong approach or the wrong solution is chosen, and suitable patterns and frameworks are not applied. This can mean that the software is more expensive to develop and maintain than necessary.

If the team does not have the required domain expertise, the risk arises that the wrong assumptions are made about the required support of domain-specific processes. This can mean that the software does not, or does not properly, meet stated requirements and other wishes.

This risk can be reduced by applying the following controls:

— Control 01: Identifying and involving stakeholders.

— Control 02: Identifying important non-functional requirements.

— Control 03: Identifying important functional requirements.

— Control 13: Using a quality-driven development method.

— Control 15: Proper handover.

— Control 16: Supporting teams with specialist knowledge and tools.

6.3.5 Risk 06: The product does not offer the right functionality because of inadequate management of the work

The development and maintenance of custom software often involves various parties and various people with varying wishes and requirements. The absence of an official product owner, a lack of transparent consultation structures or the lack of a clear business case, lead to a product being built that lacks functionality which was needed to achieve project objectives.

One factor that can contribute to this risk is untimely interventions, often caused by a lack of understanding of the development method, for example a manager who intervenes in the work of a team or assigns a team member an external task during a ‘sprint’.

This risk is greater if several development teams have to work together to produce the product. Development teams that belong to different organizations, increase this risk even further.

This risk can be reduced by applying the following controls:

— Control 01: Identifying and involving stakeholders.

— Control 12: An official product owner with mandate.

6.3.6 Risk 07: The product lacks the right non-functional properties because functional requirements were given too much priority

During the development of custom software, the focus of those (directly) involved is placed on the functionality of the software product to be developed. Although this is logical in itself (given that it is the functionality that is the primary aim of the product) this can mean that too little priority (and hence time and attention) is paid to other product properties, such as quality of use, performance, security, maintainability and management functionality. This risk often leads to stakeholders like end users, functional managers and technical managers not being involved, or being involved too little or too late.

The damage that can arise from this risk, can be expressed in a need to an extension of the scope of the work, with consequences for lead time and/or costs, unnecessary costs for carrying out rework on software already produced in order that it still meets the requirements. It can also take the form of unforeseen costs after implementation, such as scaling up managers or extensive training of end users.

The damage can take the form of technical debt that manifests itself in the long term in lower maintainability. The result in the long term can be a vicious circle: technical debt and poor maintainability leads to more emphasis on functionality (because it takes so much time to make changes), and this in turn then leads to an increase in technical debt and lower maintainability.

This risk can be reduced by applying the following controls:

— Control 01: Identifying and involving stakeholders.

— Control 02: Identifying important non-functional requirements.

— Control 05: Identifying, clarifying and systematically resolving technical debt.

— Control 07: Incremental delivery of the product.

— Control 08: Iterative development approach.

— Control 10: Constantly meeting the requirements with regression tests.

6.3.7 Risk 08: Misunderstandings occur because the communication between stakeholders is poor

During custom software development it is essential that the various stakeholders communicate clearly about product vision, requirements and wishes; misunderstanding caused by poor communication forms an important risk. Physical distance between the development team and the users can exacerbate this. If they rarely (or never) speak to one in person, the chance of misunderstandings is increased. In case of near- or offshoring of custom software development, cultural differences and unfamiliarity with content matter can multiply this risk.

The consequences of misunderstandings consist primarily of repair work needed because otherwise correctly functioning software does not or cannot do what the stakeholders had in mind. This repair work can lead to rising costs, certainly if contractual costs are associated with proposed changes and later delivery. In the most extreme case, a project can even completely fail because the functionality of the developed software is too far from the vision of the stakeholders, or because they lose confidence in the product.

Traditionally this problem was addressed by setting out vision, requirements and wishes as clearly as possible in documentation beforehand, sometimes even in the form of formal specifications. Agile development methods emphasize the need for clear communication between the stakeholders at specific times during the development, and for continued refinement and adjustment of requirements and wishes.

This risk can be reduced by applying the following controls:

— Control 01: Identifying and involving stakeholders.

— Control 08: Iterative development approach.

— Control 12: An official product owner with mandate.

6.3.8 Risk 09: Custom software does not (demonstrably) meet obligations because development, use and maintenance were not sufficiently auditable

The development and maintenance of custom software often involves specific obligations with regard to the use and management of the system, such as requirements of external stakeholders, acceptance criteria of the management organization, criteria for being able to connect to a supply system, contractual obligations and statutory obligations, such as accessibility and compliance with public records laws. If such obligations are the subject of an audit, a tendering process or a conflict, it is important to be able to demonstrate that a system complies with them. This need is also referred to as 'traceability' and it can be ensured by for example, audit trails of user actions, logging of system operations, documentation and the archiving of the state of the system as well as changes to it, and the traceable management of requirements and wishes.

If insufficient attention has been paid to this during the development of custom software, the risk arises that the software or management organization is not equipped to (demonstrably) meet these obligations. This can mean that it is not possible to demonstrate what the state of the system is, or was in the past, what actions users have or have not carried out with the system, what information the system has delivered to other systems or what requirements the system meets and in what way these requirements are achieved.

The delegation of tasks to other parties can increase the chance of this risk occurring. This can occur if development and/or maintenance are handed over to an external service provider (or another provider), or if data or functionality in a custom software system are migrated to another system. Also, the involvement of many different parties in the development and/or maintenance of the software can increase this risk due to lack of clarity or misunderstandings about who is responsible for the realization of particular obligations.

This risk can be reduced by applying the following controls:

— Control 12: An official product owner with mandate.

— Control 14: Archiving.

— Control 15: Proper handover.

— Control 16: Supporting teams with specialist knowledge and tools.

6.3.9 Risk 10: The product is not delivered on time because a great deal of time was needed to set up for software development

For the development and maintenance of custom software, just as with almost all other business activities, certain preconditions need to be met for the development and maintenance of custom software. Examples of this are the acquisition of the right tools, their availability and the setup of workstations. The development and maintenance of custom software does however make very specific requirements for meeting these preconditions. If this is insufficiently supported in the organization, this can have negative consequences for the team.

It can seem challenging to provide resources, tools, or workstations via standardized processes, because the people carrying out these processes often do not have the specialist knowledge of development or maintenance processes. This can mean that in this way the specific needs of a software development team are insufficiently implemented. If the team has to adapt its own processes, avoids the standard processes or carries out these processes itself, that can be at the cost of lead time and budget.

Contractual agreements with external service providers to carry out standardized processes can affect the chance of this risk occurring and the severity of the consequences. Contracts in which process agreements established are too rigid can hamper the implementation of specific or changing needs. Also, the lack of agreements on exceptions in the procedure can limit the options for absorbing the consequences.

This risk can be reduced by applying the following control:

— Control 16: Supporting teams with specialist knowledge and resources.

7 Controls

7.1 General

An organization that opts to develop and/or maintain its own custom software often does not limit this to one application because of the associated costs and the required expertise and tools. This document therefore assumes that an organization is developing and/or maintaining several custom applications in parallel, and that within the organization there are several organizational units (teams, departments and/or projects) that are each developing and/or maintaining one or more custom applications. Depending on the context these organizational units are referred to in this document by the term ‘project’, ‘project team’ or ‘team’.

NOTE The breakdown of controls into organization-related controls and project-related controls is a simplified version of the breakdown of processes in ISO/IEC/IEEE 12207. Organizational project-enabling processes are in this document called organization-related controls. Agreement processes, technical management processes and technical processes are in this document called project-related controls.

Some controls are primarily applied by the custom software acquirer, some primarily by the supplier and many by the acquirer and supplier acting together. Examples of controls that are primarily carried out by the acquirer are identifying stakeholders (Control 01) and identifying important non-functional and functional requirements (Control 02 and Control 03). Examples of controls that are probably used primarily by the supplier, are setting up an automated development pipeline (Control 09) and clarifying technical debt (Control 05).

Annex A cross-references the risks and controls described in this document. Annex B provides information on performing a risk inventory or risk assessment.

7.2 Project-related controls

7.2.1 General

For project-related controls a distinction is made between controls carried out for the preparation of development and/or maintenance work (the preparation phase), controls carried out during the period (the realization phase) and controls that are carried out on completion of the development and/or maintenance (the completion phase).

These controls also apply if the work is not organized as a project but in the form of an ongoing service, as is often the case for maintenance. These controls also apply if a development method is used where these phases overlap with one another or take place partly at the same time.

During the preparation phase representatives of the acquirer, supplier and intended management party work closely together for an effective realization phase. They are preferably also those who are to be involved in the execution and realization of (some of) the products to be delivered. After the project preparation those directly involved have sufficient knowledge to be able to carry out the realization phase successfully. During the realization phase the building and maintenance of the software takes place (see 7.2.3). During the completion phase archiving and any handover of the work takes place (see 7.2.4).

The purpose of splitting into these phases is threefold:

— to make the principles, risks, and preconditions explicit prior to execution of the work;

— to ensure that the preconditions are met and controls are taken for as many product-specific risks as possible;

— to ensure that completion and handover of the work is given sufficient attention.

7.2.2 Project preparation

7.2.2.1 General

Project preparation is the phase in which those organizations involved in a project prepare for the execution of the software development and/or the maintenance.

7.2.2.2 Control 01: Identifying and involving stakeholders

The identification of stakeholders (also called stakeholder analysis) take at least account of each of the stakeholders, their interests and the way in which they are involved. This is typically laid down in a project plan or service delivery plan.

The following parties are often stakeholders in the development and/or maintenance of custom software:

— end users;

— subject experts;

— managers of the software. These can be functional managers, application managers and technical managers;

— developers, particularly if specific expertise is needed which only a limited number of designers, programmers or testers have;

— acquirer(s);

— product owner: the current owner of the system to be maintained or the intended owner of a new system to be developed. In agile methods, such as Scrum and extreme programming, the product owner is the primary person responsible for maximising the value of the system to be developed or managed; this is a role and does not necessarily also have to be the business product owner;

— managers of systems to be linked;

— managers of the technical infrastructure to be used.

A method to make the interests of users and other stakeholders tangible for the development team is the use of 'persona'. Persona forms a characterization of certain types of users, for example 'lockkeeper Johanna van der Riet' or 'Jan Bakker, father of two young children at junior school, in Edam (the name of the village where he lives)'.

In addition to identifying stakeholders it is important to involve them and keep them informed at an early stage. Stakeholders can for example be involved in the following ways in the development and maintenance of custom software:

— involving the stakeholders in drawing up requirements and acceptance criteria;

— organizing a start-up meeting; at this meeting all the stakeholders and the development team are introduced to one another, given the same information and have an opportunity to establish the contact necessary for the exchange of information later on;

— organizing regular demos during the course of the work in which the team demonstrates the software to the stakeholders;

— giving stakeholders access to a demo environment to try out and test the software;

— bringing users, managers and developers closer together to provide a better understanding of one another's wishes and problems. A DevOps approach can help here, where a so-called DevOps team is responsible for both developing the custom software and installing and managing the software in the production environment. This helps to reduce the tension commonly felt between development teams (who often want rapid change) and systems management teams (who often want stability) because the entire team is responsible for both changing the software and keeping it available.

By identifying the stakeholders and their interests at an early stage the following risks can be reduced;

— Risk 03: Planned functionality is not completed on time because of underestimation of the amount of work involved.

— Risk 05: The software does not meet the requirements laid down because the team does not have the required expertise.

— Risk 06: The product does not offer the right functionality because of inadequate management of the work.

— Risk 07: The product lacks the right non-functional properties because functional requirements were given too much priority.

— Risk 08: Misunderstandings occur because the communication between stakeholders is poor.

7.2.2.3 Control 02: Identifying important non-functional requirements

This control is to identify the important non-functional requirements to such a level that it is possible to estimate the execution phase with sufficient certainty about the required expertise, lead time and costs.

The important non-functional requirements can be identified by:

1) involving all the stakeholders at an early stage during drawing up and approval of requirements;

2) drawing up a business impact analysis (BIA);

NOTE 1 In a BIA the acquirer organization establishes how important information security is for its own operational management/processes. In addition to the susceptibility to incidents it is also established to what extent an organization is prepared to take risks (the risk appetite). Only the acquirer organization itself can make a pronouncement on this.

3) drawing up a data protection impact assessment (DPIA);

NOTE 2 A DPIA gives an idea of the risks that processing constitutes for those involved and of the controls that the responsible organization needs to cover the risks. In many cases this is compulsory under privacy laws, like the European General Data Protection Regulation (GDPR). It is a statutory duty for an organization to allow inspection of privacy-sensitive data and to take sufficient protection controls to have these taken and an organization cannot pass these on to another party.

4) identifying other statutory obligations;

NOTE 3 For example accessibility and public records laws.

5) identifying other frameworks such as reference architectures, protocols, and cooperation agreements;

6) prioritizing quality aspects, such as for example described by ISO/IEC 25010 and elaborating what is important.

The non-functional requirements can be set out in a program of requirements or as part of a project start-up architecture. If applicable, they are also included in user stories and put in a product backlog.

Non-functional requirements often lead to functional requirements. For example, security requirements that require that certain logging is carried out. The non-functional requirements and functional requirements derived from these together with the other functional requirements form input for the product breakdown (see Control 04).

By identifying the important non-functional requirements at an early stage, the following risks can be reduced:

— Risk 05: The software does not meet the requirements laid down because the team does not have the required expertise.

— Risk 07: The product lacks the right non-functional properties because functional requirements were given too much priority.

7.2.2.4 Control 03: Identifying important functional requirements

This control is to identify the important functional requirements to such a level that it is possible to estimate the execution phase with sufficient certainty. This can be done by involving all the stakeholders at an early stage during drawing up and approval of the requirements.

It is also sensible to establish the source of each requirement, so that its origin is traceable. Depending on the work agreements (possibly laid down in a 'definition of ready'), the functional requirements can also be made measurable and testable by adding acceptance criteria.

NOTE Functional requirements can for example be laid down in the form of 'use cases' or 'user stories'.

The functional requirements together with non-functional requirements form input for the product breakdown (see Control 04).

Use of a minimum viable product (MVP) and support techniques such as story mapping, design thinking and the lean start-up principle clarify what has to be delivered as a minimum on a certain date. This can be pursued by all the parties. The MVP is established in consultation with the acquirer.

The core of lean start-up is to learn quickly what works and what does not. One of the methods is for example to build a simple prototype instead of using an elaborated idea of a new product or service. Extensive testing of this is carried out to see if it is liked (measure). The testing takes place direct with the customer and this provides feedback and helps in the further development of the product (learn). Each time 'build-measure-learn' is used. This speeds up innovation and limits costs.

By identifying the important functional requirements at an early stage, the following risk can be reduced:

— Risk 05: The software does not meet the requirements laid down because the team does not have the required expertise.

7.2.2.5 Control 04: Product breakdown into incrementally deliverable parts with business value

This control is to make a product breakdown into separately deliverable parts where each delivery can realize value for the acquirer and/or users.

Making a product breakdown consists of the following activities:

a) making a product breakdown structure (PBS) of the product to be created with parts that can be delivered separately and that have a business value;

b) assigning the functional and non-functional requirements to the parts;

c) putting the parts in order of business value, taking into account technical dependencies (product backlog);

d) making objective or intersubjective size estimates of the parts;

NOTE 1 To determine the functional size there are methods such as function point analysis, T-shirt sizing and user story points. For non-functional requirements these methods are still in their infancy. For non-functional software requirements, ISO/IEC/IEEE 32430 (a trial use standard) offers a sizing method.

NOTE 2 A intersubjective size estimate is a size estimate that is based on several subjective estimates. By means of an iterative approach several subjective estimates are converted into one joint estimate. Examples of such an approach are the Delphi method and 'planning poker' with user story points.

e) defining the smallest set of parts that can be taken into use (minimum viable product).

A product breakdown into incrementally deliverable parts makes it possible to reduce the following risks:

— Risk 03: Planned functionality is not completed on time because of underestimation of the amount of work involved.

— Risk 04: The product is not delivered on time and within budget because the scope is changed.

7.2.2.6 Control 05: Identifying, clarifying and systematically resolving technical debt

The presence of technical debt has an adverse effect on the quality of end products but the occurrence of technical debt during the lifetime of the software is in practice unavoidable. Technical debt can even be a useful tool if used intentionally: for example, it is sometimes possible to complete a new function quickly by copying and adapting a piece of existing functionality. The technical debt incurred in the form of duplication between the original and the copied source code can be resolved at a later date.

If existing custom software has to be dismantled, maintained, rebuilt and/or reused, then it is important to determine the quality of this software and to identify any technical debt present. Also, if it is later found that software products not previously investigated need to be added to the scope of the work, an additional survey is first carried out of these software products.

In all cases it is sensible to know what technical debt exists. To prevent technical debt not being resolved and only increasing, consider taking a methodical approach to reducing technical debt.

A methodical approach typically includes the following steps.

a) investigating the completeness and quality of the existing software products; the investigation results in an overview of quality and any technical debt present;

b) during the development and maintenance of custom software continually determining the quality of source code and documentation:

   1) measuring the quality of source code by using automated tools to compare it with best practices in the area of programming guidelines, architecture and security;

   2) measuring the quality of the source code and documentation by having team members review one another's changes;

c) clarifying which versions of third-party components and frameworks are used in the software and how far these versions are behind;

d) keeping an issue log of technical debt issues;

e) planning the resolution of technical debt issues; including time to resolve technical debt in the estimate of adjustments;

f) resolving technical debt issues.

This control helps reduce the following risks.

— Risk 03: Planned functionality is not completed on time because of underestimation of the amount of work involved.

— Risk 07: The product lacks the right non-functional properties because functional requirements were given too much priority.

7.2.2.7 Control 06: Exploring solutions

If ambiguities or uncertainties are known to exist about a required product, then solutions can be explored to arrive at a better estimate of the amount of work. Feedback on the feasibility of a solution can be obtained relatively quickly by completing a limited version of the solution (or part of it), or it can be obtained relatively slowly by developing a more extensive version of the solution.

By developing one or more prototypes, feedback from users on the user interface and operation of the software can be obtained at an early stage. If the primary aim of a prototype is to determine whether a certain design is feasible, then this is also referred to as a 'proof of concept'. In all cases a prototype only contains some of the ultimate functionality to be produced or, in the case of a 'paper prototype', no functionality at all: the user interface is simulated on paper or in a presentation.

NOTE The use of prototypes involves a specific risk: prototypes are developed as throw away software, but there can be pressure to include the prototype software in the final software, without the time being taken to bring the quality to a sufficient level.

By developing a minimum viable product, extensive feedback from users on the user interface and operation of the software can be obtained. As opposed to a prototype, an MVP is put into operational use. On the one hand, this allows for obtaining more extensive feedback than is possible with a prototype. On the other hand, the cost and lead time to develop an MVP are much larger than to develop a prototype.

Also, during the development of software an additional survey can sometimes be necessary to determine the best solution for some of the work. In extreme programming this is referred to as a 'spike'.

This control helps reduce the following risk:

— Risk 03: Planned functionality is not completed on time because of underestimation of the amount of work involved.

7.2.3 Project execution

7.2.3.1 Control 07: Incremental delivery of the product

This control is to deliver the product in increments using a product breakdown (see Control 04). The risk of there being insufficient functionality available at the required time is greatly reduced as the product breakdown is arranged by business value for the acquirer. In principle an increment of the software can be taken into use and at least be accepted by the acquirer. Part of the delivery can consist of a product demonstration to the acquirer and users. Note here that the software is only really completed when it is taken into use and that at that time all sorts of problems can still be hidden within it.

This control helps reduce the following risks:

— Risk 03: Planned functionality is not completed on time because of underestimation of the amount of work involved.

— Risk 04: The product is not delivered on time and within budget because the scope is changed.

— Risk 07: The product lacks the right non-functional properties because functional requirements were given too much priority.

7.2.3.2 Control 08: Iterative development approach

In an iterative development approach, the team works on the software in iterations within each of which roughly the same cycle of work is carried out. Examples here include designing a change to the software, programming this change, carrying out a code review on the changed source code, testing the change, carrying out regression tests and carrying out security tests.

Iterations can be of a fixed length of time where a varying number of changes per iteration are completed, or they can be of a varying length of time because for each iteration a fixed number of changes are completed.

There are many opportunities for verbal communication because during iterations the whole team is working more or less simultaneously on a limited number of changes. There is less need to communicate via documents, certainly compared with a situation where a programmer implements a design that a designer has made weeks or months earlier.

The work is spread over iterations in order of priority so that changes with a higher priority are carried out sooner. Each iteration leads to a version of the software that is deliverable (Control 07) and meets the functional and non-functional requirements (Control 10).

NOTE Examples of an approach with iterations with a fixed length of time are Scrum and extreme programming. An example of an approach with iterations without a fixed length of time is Kanban.

This control helps reduce the following risks:

— Risk 03: Planned functionality is not completed on time because of underestimation of the amount of work involved.

— Risk 07: The product lacks the right non-functional properties because functional requirements were given too much priority.

— Risk 08: Misunderstandings occur because the communication between stakeholders is poor.

7.2.3.3 Control 09: Setting up automated development pipeline

The chance of regressions going unnoticed can be reduced by setting up an automated development pipeline that upon each change in the software automatically builds the software, carries out tests and quality controls, and reports on these to the developers.

It is not always feasible to automate the whole development pipeline because of the lead times for tests (in particular endurance tests) and (for example) licences for test tools. Ideally the development pipeline includes the following activities:

— software build;

— unit tests;

— integration tests;

— quality controls;

— functional tests;

— performance tests;

— security tests;

— installation of the software;

— delivery of the total product, that is including all the deliverables, in the form usable for and agreed with the acquirer.

The organization provides support teams and tools so that they can use this pipeline. The teams are themselves responsible for the correct operation of the development pipeline.

This control helps reduce the following risks:

— Risk 01: The quality of the software is reduced because of modifications to it.

— Risk 02: The quality of the software is reduced because of modifications to the environment in which it runs.

7.2.3.4 Control 10: Constantly meeting the requirements with regression tests

Regression tests are tests that verify whether software previously developed still works correctly after changes to the relevant software or the software environment.

Regression tests test the functionality of the software but can also test non-functional properties of the software. Examples are testing performance, testing security and testing accessibility.

NOTE 1 Examples for performance tests are load, endurance and stress tests. Often the same test scripts can be used for all three performance test types. For the load test the test script is run with a normal production load to measure whether the response times and processing times meet the requirements. For the endurance test the test script is run with the normal production load, but for much longer to identify possible memory leaks and other resource depletion. In stress tests the test script is run with an increasing load, until after the moment when the software can no longer process the load properly and the response times increase and/or faults occur.

NOTE 2 Examples for security tools are in any case a tool for the static analysis of source code for security risks, a tool for the dynamic analysis of the software for security risks and a tool for reporting on known vulnerabilities in libraries used.

NOTE 3 Examples for accessibility are tools that control (some of) the Web Guidelines [WCAG, 2018].

To monitor the quality of the regression tests the code coverage of the regression tests can be measured. Code coverage reports show what parts (modules, classes, functions and/or rules) of the source code are (or are not) carried out when running a regression test. A regression test can bring to light faults in parts not covered.

It is essential to the effectiveness of this control that the regression tests are maintained. If new functionality is introduced, the test scripts used for this need to be added to the regression tests. If functionality is changed, investigate whether existing regression tests also need to be changed.

Not all tests can be easily automated. Functionality and quality aspects that cannot be tested automatically are periodically tested manually. Examples here are having a periodic black box security test carried out by a specialist security tester and carrying out a user-friendliness survey with users. However, regression tests carried out manually are labour-intensive, fault-prone and often depend on the presence of certain employees. It is therefore preferable to automate regression tests where possible so that they are repeatable and can form part of the automated development pipeline (see Control 09).

Findings from the manual regression tests are recorded as part of the working stock for the development process.

This control helps reduce the following risks:

— Risk 01: The quality of the software is reduced because of modifications to it.

— Risk 02: The quality of the software is reduced because of modifications to the environment in which it runs.

— Risk 07: The product lacks the right non-functional properties because functional requirements were given too much priority.

7.2.3.5 Control 11: Progress monitoring with burndown charts

It is possible to monitor progress in terms of work still to be delivered by making a product breakdown and estimating the size of the parts (Control 04), incremental delivery (Control 07) and iterative working (Control 08). At any time, there is an overview of parts of the product breakdown that are still to be delivered (often referred to as backlog) and the size of these parts. After the first increments have been delivered, empirical information is also available on the speed at which the team delivers parts, and projections can be made of the times when certain parts (or the whole) are be delivered. A level of transparency about the progress is obtained by sharing or displaying the burndown chart.

A burndown chart is created by plotting the size of the parts still to be delivered on the Y-axis of a graph and the time on the X-axis (see Figure 1).

Based on the burndown chart the amount of work can again be estimated. If it is found that the amount of work has not been correctly estimated, the planning can be adjusted.

NOTE 1 Burndown charts can also be used to monitor progress within an iteration.

NOTE 2 An earned value management system can also be used to monitor work progress.

This control helps reduce the following risk:

— Risk 03: Planned functionality is not completed on time because of underestimation of the amount of work involved.

7.2.3.6 Control 12: An official product owner with mandate

This control is to identify a product owner with domain expertise. The product owner is a role derived from Scrum that ensures that the development team works on the right user stories, in order of priority. The product owner continuously collects wishes and requirements, prioritizes them, and communicates them to the development team. The product owner needs a sufficient mandate to decide what work the development team carries out.

This control helps reduce the following risks:

— Risk 04: The product is not delivered on time and within budget because the scope is changed.

— Risk 06: The product does not offer the right functionality because of inadequate management of the work.

— Risk 08: Misunderstandings occur because the communication between stakeholders is poor.

— Risk 09: Custom software does not (demonstrably) meet obligations because development, use and maintenance were not sufficiently auditable.

7.2.3.7 Control 13: Using a quality-driven development method

A quality-driven development approach begins with specifying acceptance criteria tests so that, prior to programming increments of stories, it is already specified what criteria is to be tested. This makes it clear to the team when the work on an increment or story is complete. In addition, it helps make the question and the solution more specific. Finally, the specifications can be used to obtain feedback from the acquirer before software is built.

A few known quality-driven development methods are test-driven development (TDD), acceptance test-driven development (ATDD) and behaviour-driven development (BDD).

This control helps reduce the following risks:

— Risk 01: The quality of the software is reduced because of modifications to it.

— Risk 02: The quality of the software is reduced because of modifications to the environment in which it runs.

— Risk 05: The software does not meet the requirements laid down because the team does not have the required expertise.

7.2.4 Completion of development and/or maintenance

7.2.4.1 Control 14: Archiving

The completion of work on development and/or maintenance of the software the is made explicit. All the documentation, source code, reference data and credentials that were needed or have been delivered, are archived and removed from employees' workstations. Archiving facilitates a restart of the work or handover of the product at a later time. Removal takes away an unnecessary risk of breach of confidentiality and safeguards employees and organization from suspicion and liability when an incident occurs.

This control helps reduce the following risk:

— Risk 09: Custom software does not (demonstrably) meet obligations because development, use and maintenance were not sufficiently auditable.

7.2.4.2 Control 15: Proper handover

If the custom software is to be maintained by another organization, a proper handover of the software, documentation and testware needs to be done. Before the handover, parties involved establish that:

— the documentation correctly reflects the development and test environment used;

— documentation is present on data models, the functional classification, description of links and report definitions and (work) processes;

— documentation is present on back-up/recovery, procedures in case of disasters, regularly recurring maintenance activities, start-up and completion procedures;

— a list of known shortcomings is present;

— the change history of source code and related documentation is present;

— the degree of duplication in the source code is limited or justified;

— individual units (e.g. classes, methods, and functions) of the source code each have a limited number of possible execution paths such that their operation can be tested;

— traces of work not completed are absent or limited in number or size;

— a sufficient proportion of the source code is covered during execution of testing; what constitutes sufficient is derived from the non-functional requirements;

— non-functional requirements are covered by test cases;

— all identified product risks are covered by a test plan and test cases;

— a regression test (manual or automated) is available that adequately guarantees the absence of regression failures;

— there is a traceable link between test cases and requirements;

— the structure of the test set is properly structured;

— version management and version history are part of the artefacts handed over.

This control helps reduce the following risks:

— Risk 01: The quality of the software is reduced because of modifications to it.

— Risk 05: The software does not meet the requirements laid down because the team does not have the required expertise.

— Risk 09: Custom software does not (demonstrably) meet obligations because development, use and maintenance were not sufficiently auditable.

7.3 Organization-related controls

7.3.1 Control 16: Supporting teams with specialist knowledge and tools

Many of the controls assume or require certain specialist knowledge and tools. It is often inefficient for each team to acquire, implement and maintain such knowledge and tools themselves and the organization elects to make them available to the teams. This involves the following tools:

a) templates for documents, such as project start architecture document, software architecture document, quality requirements, project plan and so on;

b) a tool that supports iterative and incremental working; such a tool provides for managing requirements, documenting logical test cases and linking logical test cases to requirements, keeping track of a work backlog, planning iterations and assigning requirements to iterations;

c) a tool that supports setting up and carrying out an automated development pipeline;

d) a tool that supports monitoring the quality of source code;

e) a tool that supports the release of software;

f) a tool that supports making test reports;

g) a tool that supports making quality reports;

h) version management software and, if necessary, a configuration management database (CMDB);

i) security software that checks the configuration of the application, and the environment within which this application is running, for known and commonly occurring vulnerabilities;

j) security software that scans the versions of external libraries, frameworks or other types of building blocks used by the application for known vulnerabilities;

k) security software that automatically checks the source code for the occurrence of known unsafe structures;

l) a tool that automatically assesses the source code for the use of coding standards.

The organization can elect to provide the capabilities listed above through a few multipurpose tools, through a more extensive set of specialized tools, or through a combination of both.

The organization maintains the tools referred to and supports teams in their configuration and application.

In addition to tools, the organization can make specialist knowledge available around quality aspects that are very important for most software products that the organization develops and maintains, such as:

— security;

— performance;

— quality of use and accessibility (accessibility is a statutory obligation for the public sector, see EN 301-549);

— privacy.

The organization also supports the teams in the hiring and/or recruiting of new employees and inducting them. Employees are new if they come from outside the organization, but in larger organizations it can be sensible to also regard employees who come from other departments as new.

The support consists of tools, such as standard profiles, a process for intakes/recruitment processes and screening, an onboarding process, training processes in the form of traineeships, e-learning facilities, etc.

The support also consists of guidelines for the selection of employees so that a consistent balance can be maintained between criteria such as quality, experience, training level and the period for which an employee is available.

The support also consists of guidelines for staffing new teams. Examples are guidelines for the required mix of employees as regards diversity, experience level and the extent to which they are familiar with the way of working within the organization.

This control helps reduce the following risks:

— Risk 05: The software does not meet the requirements laid down because the team does not have the required expertise.

— Risk 09: Custom software does not (demonstrably) meet obligations because development, use and maintenance were not sufficiently auditable.

— Risk 10: The product is not delivered on time because a great deal of time was needed to set up for software development.

7.3.2 Control 17: Continuous risk management

This document aims to give controls for commonly occurring risks in custom software development and maintenance but does not claim to be complete. Furthermore, every organization that carries out custom software development and maintenance, in addition to the generic risks referred to in Clause 6, also runs project- and organization-specific risks. To identify and mitigate these risks at an early stage it is recommended to carry out continuous risk management to continue identifying and mitigating both project- and organization-specific risks.

Continuous risk management is a process and consists of the following activities, that can be used for both organization, project and team:

a) identification of risks;

b) estimate of the consequences; determining, among others, costs, planning, hazards, technical performance, suitability, functionality;

c) analysing and prioritizing the risks from most critical to least critical;

d) risk mitigation; determining for what risks risk reduction must be used, why for these risks, by means of what control and what risks are accepted and why;

e) implementation; planning the implementation of controls and monitor the risk reduction results achieved;

f) monitoring; adding all the newly identified risks to a list and keep this up-to-date by periodically reviewing and updating the risk reduction results achieved.

In addition, it is sensible to set up a mechanism in order to analyse and mitigate risks organization wide that are identified in projects and teams, but actually go beyond project or team.

NOTE 1 Annex B provides one method of performing a risk inventory or risk assessment.

NOTE 2 See ISO/IEC/IEEE 16085 for additional guidance on risk management in the context of ISO/IEC/IEEE 12207.