Tampere University

Arttu Ylikotila

UNCERTAINTY IN SOFTWARE DEVELOPMENT

A threat and a possibility

Faculty of Information Technology and Communication Sciences
M. Sc. Thesis
June 2023

ABSTRACT

Arttu Ylikotila: Uncertainty in software development – a threat and a possibility
M.Sc. Thesis
Tampere University
Master's Degree Programme in Computer Sciences
June 2023

Uncertainty is a pervasive and inevitable phenomenon in software development. It affects most if not all stakeholders in software projects in different ways. Mostly uncertainty is seen as a risk or threat that is one of the causes behind the failures of software projects. But there are also possibilities or opportunities that can be found from uncertainty.

Uncertainty has been researched in academia, but not often from the viewpoint of software development. Understanding the causes and effects of uncertainty is needed to be able to mitigate the negative and to increase the positive aspects. Understanding the subject may also help in coping with the effects of inevitable uncertainties. This thesis explores the subject by conducting a literature review. The aim of this work is to increase understanding of causes and effects of uncertainty and how uncertainty can be managed in software development projects.

This thesis discusses different types and sources of uncertainty that affect software development projects. The type represents what the uncertainty is about, and the source represents what causes the uncertainty. The presented types include for example requirements, stakeholders, and situation. The examined sources contain ambiguity, complexity, and lack of trust among other things. The effects of uncertainty on development projects and individual developers are discussed as well. The effects on projects include for example delays in schedule, decreased product quality, and poor estimates, while the effects on developers include stress, feelings of inadequacy, or increased motivation among other things. Discussion of uncertainty management is divided into reducing uncertainty and coping with uncertainty. The former can be achieved for example by maintaining continuous and direct communication with stakeholders and by doing the development of the project in short, repeated iterations that builds the project in small steps. Coping with uncertainty can be facilitated by high autonomy of the team and trust between project members among other things. Also, the suitability of different software development processes in relation to uncertainty are discussed with the conclusion being that project type is a major factor in what software development process should be used.

Keywords and terms: uncertainty, software development, threat, risk, possibility, opportunity, cause, effect

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

Contents

1 Introduction

It is common for software development projects to fail [Wang and Liu 2006]. Exceeding the project budget or not being able to deliver all the features agreed in contract are common and even total failures happen way too often [Böhle *et al.* 2016]. According to Hughes and others [2017], more than half of software development projects are not successful, but the success rates are improving slowly [Hughes *et al.* 2017].

One of the reasons for software development project failures is uncertainty [Na *et al.* 2004; Sillitti *et al.* 2005; Ibrahim *et al.* 2009; Dönmez and Grote 2015]. Uncertainty is ubiquitous [Hillson 2002; Laplante and Neill 2005; Dönmez and Grote 2018; Madsen 2007] and unavoidable [De Meyer *et al.* 2002; Na *et al.* 2004; Ibrahim *et al.* 2009; Letier *et al.* 2014; Dönmez and Grote 2015; Ubayashi *et al.* 2019] part of software development. While uncertainty in projects is mostly seen as a threat, uncertainty can also enable opportunities by e.g., acting as a driver for creativity and innovation [Dönmez and Grote 2013].

There are many sources of uncertainty [Hillson 2002] and they can affect software development in many different ways [Dönmez and Grote 2018]. Among the many sources are e.g., requirement specifications [Laplante and Neill 2005], design [Ibrahim *et al.* 2009], limited availability of information [Madsen 2007], changing requirements [Ubayashi *et al.* 2019] and unexpected events [Dönmez and Grote 2013]. Some of the effects of uncertainty include delaying or blocking action [Lipshitz and Strauss 1997], increasing the difficulty of estimating the time and effort needed to complete project [Jiang *et al.* 2009], unnecessary work [Taipalus *et al.* 2020], and causing stress [Greco and Roger 2003] as well as anxiety [Carleton 2016] to developers. Uncertainty has also positive effects like being a source of motivation [Taipalus *et al.* 2020], enabling evolution [Perminova *et al.* 2008] and innovation [Dönmez and Grote 2018].

Managing uncertainty in software development projects is important [Laplante and Neill 2005] and associated with success of projects [Hillson 2002], but many project managers have difficulties in handling it [De Meyer *et al.* 2002]. The two prevalent ways of managing uncertainty are minimizing uncertainty and coping with uncertainty [Dönmez and Grote 2018]. Minimizing uncertainty is done by increasing control and eliminating as many sources of uncertainty as possible [Dönmez and Grote 2018]. Full control over uncertainty or eliminating all sources of uncertainty is usually not possible [Dönmez and Grote 2018] and therefore other strategies are needed as well. Coping with uncertainty requires flexibility and has no standardized procedures [Dönmez and Grote 2018].

Traditionally software projects have been managed with plan-driven development processes like the waterfall model, but during recent decades alternative Agile methodologies

have been increasing their popularity as the preferred management method [Taipalus *et al.* 2020]. Agile approach to software development enables developers to respond to uncertainty and unexpected events by being flexible and responsive [Dönmez and Grote 2018], but Agile methods are not suitable in all projects [Cockburn and Highsmith 2001] and some Agile customs can lead to increasing uncertainties [Aitken and Ilango 2013].

Project management has previously been extensively studied [Taipalus *et al.* 2020], but uncertainty as a concept has not received much attention [Madsen 2007] in this context. Most of the research on topic has been written from the viewpoint of project manager, which leads to too narrow view of the phenomenon, since uncertainty affects all of the stakeholders in software development projects [Taipalus *et al.* 2020]. Uncertainty is a complex issue [Perminova *et al.* 2008] and in order to understand it, it has to be viewed from as many angles as possible [Madsen 2007]. Understanding the sources of uncertainty is required [Marinho *et al.* 2014], but it is not enough for full picture. Understanding how humans feel and act when facing uncertainty is also required, since software development projects are conducted by humans.

The focus of this thesis is uncertainty in the context of software development. The topic is explored using a literary review. Since there is a lack of studies focused especially on uncertainty in software development, studies from other disciplines are used as well in cases where studies have enough similarities to the context of software development. Slightly multidisciplinary approach is used to find information on how uncertainty affects humans from the perspective of psychology. The purpose of this thesis is to increase the understanding of uncertainty in software development from many angles, because understanding is essential for reducing the negative effects and increasing the positive aspects of the phenomenon.

The literature review was conducted by first doing searches in the following databases: ScienceDirect, Scopus, IEEE Xplore, and SpringerLink. Additional searches were performed with Andor search service. The search strings used were “uncertainty AND ″software development″ AND (threat OR risk OR possibility)” and “uncertainty AND ″software engineering″ AND (threat OR risk OR possibility)”. The search results were screened by reading first the title and keywords and based on them, the abstract was read if the content of the study seemed to be relevant. If also the abstract was considered to be relevant, the study was selected for a closer look. After it seemed that the search had reached a saturation point where no more relevant studies were found, the search in the databases was stopped. The initial amount of retrieved relevant studies was not satisfactory and therefore the references from the relevant studies were followed to find more studies relevant to the topic.

The content of the found studies was evaluated based on whether the study was about uncertainty in software development from any perspective or if the topic of the study was about uncertainty and the study had enough similarities with the context of software development. The similarities accepted included for example if the context was project management in some other industry with similar qualities to project management in software development, or if the topic was for example decision making, which is central to software development. Some studies from psychology that related to effects of uncertainty were included as well. In total 41 studies discussing uncertainty or related effects were found and included in the literature review. Also, studies discussing software development processes were retrieved for background information.

The selected studies were analysed from the point of view of the research questions of this thesis and the data found was organized to support a meaningful presentation of the findings. The research questions this work seeks answers to are as follows:

RQ1: What causes uncertainty in software development?

RQ2: What are the effects of uncertainty in software development?

RQ3: How uncertainty can be managed in software development?

RQ4: How suitable are different software development processes in relation to uncertainty?

In the Chapter 2 the key concepts of the thesis are defined. Chapter 3 describes the types and sources of uncertainty found in the literature. Chapter 4 explains what kind of effects uncertainty can have on the software development project and the developers of the project. Chapter 5 discusses how uncertainty can be managed according to literature. Chapter 6 gives an overview of two software development processes and how suitable they are in relation to uncertainty. Finally, the Chapter 7 summarizes contents of the thesis.

2 Software development and uncertainty

In order to establish the context for this thesis, the key concepts of the topic are described and defined in this chapter.

2.1 Software development

Software development is activity that is aimed at creating, improving, or maintaining different kinds of computer programs. At its core software development is about problem solving [Aitken and Ilango 2013] and decision making [Ibrahim *et al.* 2009]. Software development is a complex task [Na *et al.* 2004; Clarke and O'Connor 2012; Letier *et al.* 2014] that typically requires collaboration between people with diverse skills [Na *et al.* 2004; Jun *et al.* 2011] and different kinds of stakeholders [Whitaker 2009, p. 52].

In a typical case software development consists of four activities: gathering the requirements, analysing the problem that will be solved, designing the solution for the problem, and implementing the solution [Aitken and Ilango 2013]. All the requirements are not always known at the start of the project [Whitaker 2009, p. 249]. Instead, more is learned about the project during the development and therefore the requirements are not stable [Sillitti *et al.* 2005]. The design decisions in software development are highly complex, since they often involve multiple interrelated decisions that may be conflicting, hard to define and hard to compare [Letier *et al.* 2014]. Making these decisions with absolute certainty is not possible in most cases [Ibrahim *et al.* 2009].

Software development has similarities with other engineering disciplines, but there are differences as well. Engineering processes can be either defined or empirical [Williams and Cockburn 2003]. A defined process is one that can be done with predefined actions and following these actions will always produce the same result [Williams and Cockburn 2003]. Manufacturing a car is an example of a defined process. Software development inherently contains too much change during the development, that predefined actions will most likely not achieve the desired outcome, hence software development is an empirical process [Williams and Cockburn 2003].

Software development is usually done as a project. Most of the projects have some typical characteristics. For example, they have a specific purpose aiming at a specific result, they are temporary i.e., they have a defined start and end, and they can be progressively developed in steps [Whitaker 2009, p. 49]. Software projects especially are often about creating something novel, which might not be done again in future [Bannerman 2008]. Because of this one-off nature, learning from past patterns is not always possible in software development [Bannerman 2008].

There are differences between software development projects and some of the projects have only a few uncertainties, while most of them have several different types of uncertainties [De Meyer *et al.* 2002]. Depending on the characteristics of the project, the uncertainties and their level vary [Marinho *et al.* 2014]. For example, innovation projects tend to have a higher level of uncertainties and a higher probability of failure [Marinho *et al.* 2014]. Additionally, the uncertainties of the project change and develop during the project life cycle [Atkinson *et al.* 2006]. Especially in the beginning of the project, there are lots of uncertainties since definitions are still vague and many decisions are likely to change during the project [Marinho *et al.* 2014].

2.2 Uncertainty

Ironically, the term uncertainty is abstract [Ubayashi *et al.* 2019] and ambiguous, and therefore it is uncertain in itself. There are multiple conceptualizations of uncertainty [Lipshitz and Strauss 1997] in the literature and there is no consensus about exact definition of uncertainty in the context of software development [Ibrahim *et al.* 2009]. Uncertainty has many facets [Madsen 2007], and it has been studied in multiple disciplines including at least economics, engineering, and psychology [Saunders *et al.* 2015]. The term uncertainty can mean different things depending on who is using the term [Ibrahim *et al.* 2009; Lipshitz and Strauss 1997], but the definitions usually share at least the aspect of incomplete information or doubt [Dönmez and Grote 2018] in one way or another.

The most common definition of uncertainty found in this literary review was shared in eight studies with slight differences in details. They defined uncertainty as lack of information or knowledge [Atkinson *et al.* 2006; Ibrahim *et al.* 2009; Axelsson 2011; Letier *et al.* 2014; Marinho *et al.* 2014; Mehta *et al.* 2014; Grote 2015; Dönmez and Grote 2018] to make a decision [Ibrahim *et al.* 2009; Axelsson 2011], or about the consequences of alternatives [Letier *et al.* 2014], or to predict the project outcomes [Mehta *et al.* 2014]. Marinho and others [2014] added to their definition the inability to predict the probability that an event would happen. Grote [2015] included ambiguous information to their definition. Dönmez and Grote [2018] specified, that they mean lack of information in respect to the information that is required.

Three studies had partly interrelated definitions of uncertainty. Madsen [2007] defined uncertainty as "perceived level of not knowing the appropriate course of action and/or its outcome at a given point in time". In partly similar vein, among the multiple definitions presented by Perminova and others [2008], was a definition of uncertainty from perspective of psychology as "a state of mind characterized by a conscious lack of knowledge about the outcomes of an event". Lipshitz and Strauss [1997] defined uncertainty as "a sense of doubt" in the context of action, which partly overlaps with the definition by Madsen [2007].

Howell and others [2010] defined uncertainty simply as “lack of certainty”. They included in their wide definition “probabilistic or undefined outcomes” and also “ambiguity and lack of clarity over situational parameters”.

One clearly distinct definition of uncertainty was presented by Taipalus and others [2020]. In their definition uncertainty is “an emotion caused by ambiguity”. Ambiguity can be caused by multiple sources and when the level of ambiguity reaches a certain subjective threshold, it will cause uncertainty [Taipalus *et al.* 2020].

Perminova and others [2008] described multiple definitions of uncertainty from different disciplines found from literature, but the definition of their own was “a context for risks as events having a negative impact on the project’s outcomes, or opportunities, as events that have beneficial impact on project performance”.

As can be seen from the above definitions, the term uncertainty is not self-explanatory [Perminova *et al.* 2008] and it is equivocal in itself. Ambiguity is a term often mentioned in association with uncertainty. Ambiguity refers to possibility of multiple different interpretations, which is linked to confusion and lack of understanding [Atkinson *et al.* 2006]. Notable aspect of confusion is that it may originate from lack of information as well as from abundance of information, especially if there are conflicting meanings or undifferentiated alternatives [Lipshitz and Strauss 1997]. This is contradicting with defining uncertainty simply as lack of information.

Risk is another term that is often associated with uncertainty. These two terms have even been used interchangeably [Taipalus *et al.* 2020] in literature, but they are distinct terms [Laplante and Neill 2005; Perminova *et al.* 2008; Axelsson 2011; Taipalus *et al.* 2020] that have fundamental differences [Dönmez and Grote 2018; Taipalus *et al.* 2020]. Risk in software projects is commonly understood as exposure to factors that present a threat to the outcomes of the project [Bannerman 2008] and threat is defined as an event with negative consequences [Whitaker 2009, p. 111; Johansen *et al.* 2014; Dönmez and Grote 2018]. Hence, the traditional view of risk is negative [Hillson 2002]. Uncertainties, on the other hand, can also have positive outcomes [Perminova *et al.* 2008; Lechler *et al.* 2012; Dönmez and Grote 2018; Taipalus *et al.* 2020] or they can be neutral [Martinsuo *et al.* 2014].

Another important distinction between risk and uncertainty is the amount of knowledge regarding the probabilities [Perminova *et al.* 2008] and the possibility to prepare for them beforehand [Taipalus *et al.* 2020]. For example, Lechler and others [2012] define uncertainties as unknown-unknowns and risks as known-unknowns. The important distinction here is that known-unknowns have been identified to possibly exist, but unknown-unknowns are not even identified [Geraldi *et al.* 2010]. The former can be predicted, and its

occurrence may be estimated, while the latter cannot be prepared for and cannot be estimated with any confidence [Taipalus *et al.* 2020].

One suggestion in the literature has been that uncertainty is an umbrella term, that includes risk and opportunity as two distinct varieties [Hillson 2002; Perminova *et al.* 2008]. In this line of thought, the risk is an uncertainty with negative effect and opportunity is an uncertainty with positive effect [Hillson 2002]. Therefore, risk and opportunity could be seen as an outcome or consequence of uncertainty or uncertain events [Perminova *et al.* 2008] instead of being direct aspects of uncertainty itself.

3 Types and sources of uncertainty

It is common in studies, that uncertainty is discussed as a singleton or that only one type of uncertainty is considered, instead of examining many different types of uncertainties [Dönmez and Grote 2018]. But it is important to recognize different uncertainty types since uncertainty has multiple facets [Madsen 2007] and different types of uncertainties may need to be approached with various strategies [Dönmez and Grote 2018]. The most common type of uncertainty found in literature is requirements uncertainty [Dönmez and Grote 2018] and many studies concentrate solely on that. Other frequently discussed types include for example stakeholders [Ward and Chapman 2008], environmental uncertainty [Marinho *et al.* 2015], technological uncertainty [Lechler *et al.* 2012] and uncertainty in estimates [Atkinson *et al.* 2006].

Categorizing the uncertainties is necessary in order to distinguish between different kinds of uncertainties. There are many ways of categorizing the uncertainties in the literature [Dönmez and Grote 2018] and some of them are not compatible with each other because of fundamental differences in their approaches. This thesis uses a categorization that is similar to the classification of Lipshitz and Strauss [1997], where the uncertainties are categorized according to their type and their source. Type represents broadly what the uncertainty is about, and the source represents what is the cause of the uncertainty. This classification is simple, yet expressive enough to make it possible to differentiate uncertainties from each other in a meaningful way.

Next an overview of the uncertainty types and sources found in the literature is given categorized according to the approach of this thesis where applicable.

Lipshitz and Strauss [1997] discuss uncertainty and decision makers. Since their context differs from this thesis, their results are not entirely in line with the content of this work. But since the software development is essentially about making decisions, the results of Lipshitz and Strauss [1997] are also relevant in the context of this thesis. The classification of Lipshitz and Strauss [1997] is not exactly the same as the one used in this work, even if both the approaches share the same idea. Therefore, the uncertainties in their study are organized to better suit the categorization method of this thesis and shown in Table 1. The type of uncertainty “situation” in the table means the decision makers uncertainty about the situation they are. The “role” refers to uncertainty about the role of the decision maker. The type “outcome” is concerned with the possible outcomes of decisions of the decision maker.

Table 1. Conceptualizations of uncertainty [Lipshitz and Strauss 1997] adapted to the classification of this thesis.

Type of uncertainty	**Source of uncertainty**
Situation	Complete lack of information
	Partially lacking information
	Unreliable information
	Inadequate understanding owing to equivocal information
	Inadequate understanding owing to novelty
	Inadequate understanding owing to instability
Role	Complete lack of information
	Inadequate understanding owing to equivocal information
	Inadequate understanding owing to novelty
	Conflict among alternatives owing to incompatible role demands
Outcomes	Complete lack of information
	Partially lacking information
	Conflict among alternatives owing to equally attractive outcomes

Sillitti and others [2005] are concerned solely with changing requirements and the related uncertainty. They classify these requirement uncertainties to external and internal with the distinction that the former cannot be controlled by the customer or the development team, while the latter can be controlled or managed by them. The categorization is similar to the one used in this thesis, but still fundamentally different, since it is concerned with only one type of uncertainty, that is divided in two distinct varieties that have their own sources. Those requirement uncertainties are shown in Table 2.

Table 2. Uncertainties in requirements [Sillitti et al. 2005].

Requirements variability	**Source**
External	Changes in technology
	Changes in regulations
	Changes in company factors, such as corporate politics, marketing plans, financial conditions
Internal	Limited knowledge of the application domain
	Customer's initial uncertainty. The customer is not able to define a complete set of requirements at the beginning of the project
	Relational and communication problems among the subjects involved in a project

Another set of uncertainty sources related to requirements is presented by Moynihan [2000]. Their sources are "attributes of the application", "attributes of the users", "attributes of the analysts/developers" and "wider aspects of the organization". In addition to sources, they give some examples of these sources. According to Moynihan [2000], these sources are common to many definitions of requirements uncertainty. Many of these sources can also be seen to relate to other uncertainty types identified in different studies and not only the requirements uncertainty. The sources are shown in Table 3 adapted to the categorization of this work so that their sources are converted to types of uncertainty and examples are converted to sources of uncertainty. While the classification between the works differs, the main content remains the same.

Table 3. Requirements uncertainty sources [Moynihan 2000] using the classification of this work.

Type of uncertainty	**Source of uncertainty**
Attributes of the application	Complexity
	Stability
	Novelty
	Level of change involved
Attributes of the users	Number (amount of)
	Previous computer experience
	Diversity of their needs
	Their understanding of the application
Attributes of the analysts/developers	Knowledge of the application
	Knowledge of the business
Wider aspects of the organisation	Any unhelpful `politics' etc.

A wider perspective to uncertainty sources is offered by Saunders and others [2015], who explore determinants of uncertainty from the point of view of project managers in civil-nuclear and aerospace projects. Their context differs from the context of this thesis and their determinants are not strictly the same as sources of uncertainty as in this thesis, but there is a significant overlap. Saunders and others [2015] group their determinants of uncertainty around six different themes that they call perspectives, and the determinants are the actual sources of uncertainty. Environmental perspective refers to uncertainties coming from external factors like regulation or markets. Individual perspective is concerned with uncertainty sources felt by an individual actor. The complexity perspective consists of uncertainty sources coming from attributes of the product or technology for example. Information perspective refers to missing information and lack of understanding and such. Temporal perspective is about changes in uncertainties during the progress of the project. Capability perspective centres primarily around the capabilities of the project team and organization. Summary of these determinants is shown in Table 4.

Table 4. Determinants of uncertainty [Saunders *et al.* 2015].

Perspective	**Determinants of uncertainty**
Environmental	Political, Economic, Social, Technological, Legal and Environmental influences
	Stakeholder demands
	Market and Industry
	Institutional norms & decision making
	Regulatory framework
Individual	Psychological profile
	Internal state of mind
	Bounded rationality
	Fallacy of rational decision making
Complexity	Size of project
	Product Requirements
	Diversity of actors
	Technology choices
	Nature of supply chain
Information	Lack of information, knowledge, understanding of cause and effect relationships
	Estimating ability
	Clarity of project objectives
Temporal	Lifecycle state
	Project tempo & timescale
	Turbulence
	Organisational priority of the project
Capability	Skills/expertise of project team
	Project management maturity of the organization
	Resource availability at project and industry level
	Supply chain capability

Ward and Chapman [2003] argue that project risk management could be enhanced by focusing on uncertainty rather than risk. Their context is not software development, but

rather project management in general. Despite the differences in their context and the context of this thesis, there are enough similarities for the uncertainties mentioned to be relevant to this thesis. Ward and Chapman [2003] categorize the uncertainties in different way than this thesis, but they are compatible with the classification used here. Their causes of uncertainties are classified into five areas, namely "variability associated with estimates", "basis of estimates", "design and logistics", "objectives and priorities", and "relationships between project parties". These areas of uncertainty contain different causes of uncertainty. The uncertainties found in their work are adapted to the categorization used in this work and shown in Table 5.

Table 5. Causes of uncertainty [Ward and Chapman 2003] adapted to classification of this thesis.

Type of uncertainty	**Source of uncertainty**
Variability associated with estimates	Lack of a clear specification of what is required
	Novelty, lack or experience of this particular activity
	Complexity in terms of the number of influencing factors and inter-dependencies between these factors
	Limited analysis of the processes involved in the activity
	Possible occurrence of particular events or conditions
Uncertainty about the basis of estimates	Subjective estimates
Uncertainty about design and logistics	Nature of the project deliverable
	Process for producing deliverable
Uncertainty about objectives and priorities	Project objectives
	Relative priorities between objectives
	Acceptable trade-offs
Uncertainty about fundamental relationships between project parties	Specification of responsibilities
	Perceptions of roles and responsibilities
	Communication across interfaces
	Capability of parties
	Contractual conditions and their effects
	Mechanisms for coordination and control

Lechler and others [2012] discuss uncertainties and opportunities in project management. They identify multiple sources of uncertainties across different kinds of projects. Some of the projects are software development projects, but not all of them. Nonetheless, most if not all of the sources are still relevant also in the context of this thesis. Lechler and others [2012] classified their identified sources of uncertainties to six different categories: "contextual turbulences", "stakeholders", "technological uncertainties", "organizational uncertainties", "project uncertainty", and "malpractice". Here the contextual turbulences refer to external changes that can impact the project. Stakeholder uncertainty can arise for example from customer induced changes. Technological uncertainty relates to issues with technology. Organizational uncertainty can arise for example from a corporate merger that causes unexpected changes to project teams. Project uncertainty refers for example to complexities within a particular project. Malpractice relates to absence of project management standards for example. Uncertainty sources of Lechler and others [2012] are shown in Table 6 using the categorization of the authors.

Table 6. Sources of uncertainties [Lechler *et al.* 2012].

Uncertainty categories	**Uncertainty sources**
Contextual turbulences	External legal context
	External market context (dynamic)
	Regulatory uncertainty
Stakeholder uncertainty	Customer-induced changes/contracts/diverse needs
	Inability of the vendor or contractor
	Inexperienced project manager, subcontractor, or outside designers
	Unknown project ownership
	Contractor-customer relations
	False assumptions about capabilities of contractor
Technological uncertainty	Technical issues
	Tight technical specifications
Organizational uncertainty	Organizational changes
	Incompatibility of management system
Project uncertainty	Unknown complexity
Malpractice	Self-induced uncertainty

Martinsuo and others [2014] propose a framework on uncertainties and their management in project portfolios. They identify three distinct types of uncertainties that are further divided into different uncertainty sources. The categorization they use seems to be the same or at least very similar to the classification used in this thesis. Their context is not software development, but rather project portfolios and mostly in other kinds of industries than software. Despite the differences, there is a distinct overlap between these works.

The uncertainty types identified by Martinsuo and others [2014] are “environmental uncertainty”, “organizational complexity” and “single project uncertainties”. Environmental uncertainties are mostly dealing with society or markets like for example changes in legislation, developments in markets, or competitors launching new products. Organizational complexity is about interproject relations like resource allocation, project prioritization, or changing organizational structures. Singe project uncertainty relates to schedules, technical problems, scope of the project and changes in the scope. Table 7 presents a summary of the uncertainties identified by Martinsuo and others [2014] with examples of the uncertainties.

Table 7. Summary of the uncertainties and their examples [Martinsuo et al. 2014].

Type of uncertainty	Source of uncertainty	Examples
Environment	Society	Legal developments, regulations, safety, global economy downturn
	Markets	Customers, market development, price erosion, difficulties to estimate project business impact
	Industry	Competitors, technological development
Organizational complexity	People	Organizational structure, technology push, function interaction, strategy
	Company	Organizational structure, technology push, function interaction, strategy
	Inter-project relations	Resource allocation, project scheduling, project priorities
Single projects	Evaluation	The business impact of one project, failure, learning from single projects, goal complexity
	Project characteristics	Special and large customer supply projects, product development site relocation decision
	Scope	Product features, component features, platform-development
	Cost	Project budget, product cost
	Schedule	Project duration

Dönmez and Grote [2013] present different types of uncertainties in software development and how agile teams manage the uncertainties. The classification of uncertainties they use is in essence the same that is used in this thesis. They have four different categories that are “resource uncertainties”, “requirements uncertainties”, “task uncertainties”, and “output uncertainties”. Resource uncertainties has sources like technological artefacts, infrastructure, and human resources. Requirements uncertainties include for example lack of details, ambiguous information, and unexpected changes. Task uncertainties relate to sources like missing knowledge about the scope of task or not knowing the optimal solution. Output uncertainties include for example the amount of accomplishable work, time required for the task, or quality of the product. Categorized uncertainties are shown in Table 8.

Table 8. Types and sources of uncertainties [Dönmez and Grote 2013].

Type of uncertainty	**Source of uncertainty**
Resource uncertainties	Availability of process artefacts
	Quality of input
	Availability of human resources
	Duration for new team members to become productive
Requirements uncertainties	Lack of details about demanded functionality
	Ambiguous information
	Unexpected requirement changes
Task uncertainties	Quality of a problem solution
	Unexpected difficulties
	Task sequence or process uncertainty
Output uncertainties	Time required to accomplish a task
	Amount of accomplishable work
	Project status
	Quality of the product

In their later work Dönmez and Grote [2018] explore the same topic and they present almost the same types and sources of uncertainties again. The difference in the presenta-

tion is that they have decided to remove the type “Output uncertainties”, the sources “Duration for new team members to become productive” and “Project status” and organized the remaining sources somewhat differently into the remaining types.

Taipalus and others [2020] discuss causes, effects, and coping mechanism of uncertainty in software development. They use a taxonomy of three levels to categorize the causes of uncertainties, which is fundamentally different from the classification used in this thesis. The highest level in their taxonomy has three items: “causes stemming from within the development organization”, “causes stemming from the client organization”, and “causes stemming from outside the involved organizations”. The middle level is another categorization that has eight items, which all belong to one of the categories above them. Items in the middle level are “personal matters”, “inefficient conventions”, “organizational pathoses”, “lack of interdisciplinary knowledge”, “lack of problem understanding”, “conflicts of interest”, “technical considerations”, and “causes outside the scope of influence”.

Personal matters include causes lack of trust in e.g., developers own abilities, fear for e.g., asking for help, and personal problems outside work. Inefficient conventions contain for example problems in communication due to large team size or unsuitable communication channels or differences in personal working methods. Organizational pathoses include themes like inconsistent resource allocation or inability to handle failure on organizational level. Lack of interdisciplinary knowledge relates to problems where the client and the developers can’t understand each other. Lack of problem understanding refers to incomplete requirements and lack of commitment from the client. Conflicts of interest can arise from situations where the people making decisions on the client side are those who use the product the least or when different clients have conflicting requirements for the product. Technical considerations relate to for example evaluation of different technologies and new, complex enterprise architectures where the new systems have to be integrated. Causes outside the scope of influence are factors that the development or client organization cannot prevent like for example changes in business or legislative environments, international clients operating in different legislative environments or problems with acquiring suitable workforce.

The lowest level contains 24 items or themes in total that all belong to one of the middle level categories in the taxonomy. This lowest level could be comparable with how the “source of uncertainty” category is used in this thesis. The causes of uncertainty discussed by Taipalus and others [2020] are shown in the Table 9 using the taxonomy of three levels they used in their work.

Table 9. Summary of the causes of uncertainty [Taipalus et al. 2020].

Causes stemming from within the development organization	Personal matters	Lack of trust
		Fear
		Personal problems outside work
	Inefficient conventions	Large team size
		Lack of knowledge concerning roles
		Unsuitable communication channels
		Different personal working methods
		Agile methods
		Incompetence
	Organizational pathoses	Inconsistent resource allocation
		Organizational complexities
		Failure handling
Causes stemming from the client organization	Lack of interdisciplinary knowledge	Client does not understand software
		Team does not understand the business domain
	Lack of problem understanding	Lacking initial requirements
		New features arise
		Lack of commitment from the client
	Conflicts of interest	Authority-involvement discrepancy
		Prioritization
Causes stemming from outside the organizations	Technical considerations	Complex technical environments
		Technology evaluation
	Causes outside the scope of influence	Changes in surrounding environments
		Complexities in surrounding environments
		Lack of suitable workforce

4 Effects of uncertainty

Uncertainty can have various effects for projects ranging from a total disaster to a positive surprise [Hillson 2002]. The effects of uncertainty are not limited only to project and its objectives, but instead the effects have an impact on everyone on the project [Taipalus *et al*. 2020]. Even if uncertainties can have both positive and negative effects, the potential opportunities are often neglected [Böhle *et al*. 2016], because project managers usually concentrate almost entirely on the negative effects of uncertainty [Hillson 2002]. The opportunities presented by uncertainty could be beneficial to project value if identified and exploited [Lechler *et al*. 2012].

The following two sections present both the negative and positive effects of uncertainty on the project and on the developers according to the literature.

4.1 Effects on project

The accumulated negative effects of uncertainties in software development projects include delays in schedule, higher cost, lower product quality and disappointed customers [Ibrahim *et al*. 2009]. Uncertainty can also lead to toxic working culture, which can create misunderstandings and lack of trust [Taipalus *et al*. 2020], which are both additional sources of uncertainty themselves. This effect of creating other uncertainties is one rather typical attribute of uncertainties.

Requirement uncertainty can create delays, budget overruns [Na *et al*. 2004] and increase the difficulty of predicting the time and effort needed in the project [Jiang *et al*. 2009]. Inadequate requirements are also associated with creeping user requirements, that can cause project overruns, decreased product quality and issues with team morale [Whitaker 2009, p. 161]. Ambiguous requirements may lead to rework, confusion, wasted time and difficulties in testing the produced features [Whitaker 2009, p. 161].

Ambiguous or inconsistent requirements and frequent changes also increase the difficulty of understanding the scope of the project and the requirements [Jiang *et al*. 2009], which can lead, among other things, to gold plating i.e., creating something more than was necessary [Whitaker 2009, p. 84]. Volatile requirements and disagreements between stakeholders about the requirements make it harder to agree about the project objectives, scope, and evaluation metrics, which can lead to difficulties in estimating the schedule and cost [Jiang *et al*. 2009]. The direct consequences of the poor estimates include difficulty to allocate resources correctly, schedule pressure and unrealistic expectations [Jiang *et al*. 2009].

On the positive side, requirements uncertainty may increase process performance by promoting more interaction between users and creating more learning opportunities [Na *et al*. 2004]. Embracing the uncertainty enables exploration and divergent thinking, which

can create new ideas [Grote 2015]. Uncertainty enhances innovation and can be a seedbed for creativity [Taipalus *et al*. 2020]. Also, according to Taipalus and others [2020], new business domains and technical environments may keep team members vigilant for new solutions and challenge the current ones.

Lechler and others [2012] identified four categories of opportunities related to uncertainty: technology opportunity, implementation process opportunity, project business opportunity and future project business opportunity. Technology opportunity can be for example a technical innovation or alternative technology. Implementation process opportunity could be a new standard process like common build process that can be used in multiple projects to improve efficiency. Project business opportunity can refer to early market penetration or new market solution for example. Future project opportunities can create value that spans broader than the current project like new contracts or gaining knowledge that is useful in future projects as well. [Lechler *et al.* 2012]

4.2 Effects on developers

Software developers are stressed workers [Ostberg *et al*. 2020] and there is evidence that supports uncertainty being a powerful stressor [Greco and Roger 2003]. It can also cause fear and anxiety [Carleton 2016]. The effects of stress in short term include concentration problems and increased error rate, in long term stress may cause dissatisfaction, resignation, and depression among other things [Ostberg *et al*. 2020]. Stress is also negatively associated with software developers' performance [Rezvani and Khosravi 2019] and high stress may eventually result in burnout [Sonnentag *et al*. 1994]. On team level, stress can have adverse effects on morale and motivation, communication, and cooperation, which can consequently lead to lower software quality [Ostberg *et al*. 2020].

According to Taipalus and others [2020], a common effect of uncertainty is dysphoria i.e., a mental state of unease. In addition to unhealthy amounts of stress, developers may feel dissatisfied with their implemented solutions, since sometimes there is no certainty if the solution is a good one or not. The inability to ensure high quality of the work can also cause feelings of inadequacy, that affects the ability to perform. [Taipalus *et al*. 2020]

If the uncertainty level is too high, the individual software developer may experience general reduction in job and cognitive performance as well as reality distortion. Reduction in job performance may include for example resistance to change, confrontational behaviour or social withdrawal. Cognitive performance reduction can include for example inability to concentrate in the moment or on all the relevant information. Reality distortion may happen in very critical, high uncertainty situations when actors are unable to concentrate, start to over-generalize and lose their objectivity as well as sense of proportion. [Madsen 2007]

When the uncertainty level is too high, a group of software developers may engage in groupthink or competitive rivalry. In such a situation, a well-integrated group is more likely to unite and engage in groupthink, while inchoate groups might disintegrate and resort to competitive rivalry within their group or with some other group. [Madsen 2007]

Despite the negative effects of too much uncertainty, some amount of uncertainty is perfectly natural and can act as a driver for software developer in search of alternative approaches, novelty, and change [Madsen 2007]. Uncertainty can be perceived as positive, if the context is controllable and predictable enough to enable the individual to either cope with or make sense of the uncertainty [Carleton 2016].

Possibility of self-improvement is a commonly recognized positive effect of uncertainty. It encourages to learn new technologies, improve working methods, and help social bonding with team members. Uncertainty can also improve motivation, because working with diverse projects in different business domains and creating novel systems keeps the work interesting. [Taipalus *et al*. 2020]

If the level of uncertainty is right, the work is enjoyable, and the developers can concentrate in the moment and on all the relevant information. They can pay attention to and undertake constructive conversations that may help to make sense and take control of the situation. With appropriate level of uncertainty, a group of developers may co-operate towards the task goals by employing collaborative critical inquiry and evaluation of alternatives. [Madsen 2007]

5 Managing uncertainty

Uncertainty management is not about removing all the uncertainties in software development projects [Dönmez and Grote 2013]. In practice, completely removing all uncertainties would be costly [Ibrahim *et al.* 2009] and practically impossible [Perminova *et al.* 2008]. Eliminating all the uncertainties could also be counterproductive since uncertainties can turn into opportunities [Dönmez and Grote 2018].

Traditional risk management can help with uncertainty management by reducing some of the uncertainties in a project. Risk management and uncertainty management should be seen as two distinct, complementary processes [Marinho *et al.* 2018]. They should both be incorporated to all decisions and evaluations made during the project [Jaafari 2001] and they should cover the whole life cycle of project [Ward and Chapman 2008]. Risk management practice is not discussed here in further detail because it is outside the scope of this work.

When dealing with uncertainty, the success and practicality of a plan-oriented action is limited [Böhle *et al.* 2016]. Keeping the options of the project open is recommended, since forecasting the future is not possible with any certainty [Jaafari 2001]. No plan will ever be perfect, and changes might be required during the project [Ibrahim *et al.* 2009].

Uncertainty management requires the ability to be flexible and keep focus in the project objectives, while accepting the existence of uncertainty and lack of definitive answers [Saunders *et al.* 2015]. Identifying the uncertainties is important part of the process, because not all of them are critical [Perminova *et al*. 2008] and different uncertainty types require distinct approaches in managing them [Dönmez and Grote 2018].

According to Dönmez and Grote [2018], the two prevalent uncertainty management practices are minimizing the uncertainty and coping with uncertainties. Goal of minimizing the uncertainty is to remove those uncertainties that can be removed in order to gain more control. Coping with the uncertainties on the other hand, accepts that uncertainties are inevitable and aims to manage uncertainty flexibly. Coping approach is at least partly ad-hoc activity, that has no standards to follow. [Dönmez and Grote 2018]

The following two sections discuss different ways of reducing the uncertainty and coping with uncertainty according to the literature.

5.1 Reducing uncertainty

There are various strategies and actions that can reduce different uncertainties in software development projects. At the beginning of the project, the project should be analysed and characterized to identify relevant uncertainties and the most appropriate management model should be then selected according to the project type [Marinho *et al.* 2018]. Different types of techniques that can be used in finding threats and opportunities in the

project include brainstorming, expert interviews i.e., Delphi method, situation map and checklists of typical uncertainties [Johansen *et al.* 2014]. The checklists are problematic though, because the projects and their scope are so diverse, that no checklist will ever be comprehensive enough. Using checklist as a starting point that is adapted to the project context is recommended [Bannerman 2008].

Stakeholders are a major source of uncertainty in projects [Ward and Chapman 2008] and they should be identified early along with their interests to find those that can affect the project positively or negatively [Marinho *et al.* 2018]. Expectations of stakeholders should be managed in a way that they will be tolerant to changes [Marinho *et al*. 2015], as they are likely to happen. The communication with the stakeholders is very important and it should be continuous during the project [Jiang *et al.* 2009]. There is a risk of misunderstandings due to different backgrounds of developers and stakeholders and in order to accomplish clarity, agreement and to evade misunderstandings, a shared terminology concepts should be established [Jiang *et al.* 2009].

Direct and open communication between the developers and users will help to decrease uncertainty in the requirements [Jiang *et al.* 2009]. To clarify the requirements, they can also be gathered from an existing system, inspecting user activities, or uncovered by experimenting with prototypes for example [Jiang *et al.* 2009]. If the stakeholders can't agree on requirements, one strategy to achieve self-protection could be going bureaucratic and insist writing everything down in great detail [Moynihan 2000].

Uncertainty associated with changing requirements can be mitigated by doing the development in short cyclical phases that makes it possible to refine the project in steps and to react to changes as they occur [Dönmez and Grote 2018]. The incremental way of building the project can ensure better adaptability [Moynihan 2000] and as an additional benefit, there are less uncertainties that have to be dealt with at the same time [Axelsson 2011]. Small individual task sizes can help with uncertainties related with the tasks and simultaneously improve estimations concerning them [Dönmez and Grote 2018].

Uncertainty in decisions can be reduced by getting more information before doing decisions or postpone uncertain decisions until more knowledge will be available [Lipshitz and Strauss 1997]. It should be noted though, that getting more information can also increase the uncertainty [Grote 2015] when the information is for example ambiguous or conflicting [Lipshitz and Strauss 1997]. There is also the problem that sometimes the information might be misleading or even false [Mehta *et al.* 2014]. Finding the information is only the first part, then the developers have to make sense of the information [Mehta *et al.* 2014] in order to gain benefit from the information. Acquiring and analysing the information comes with a cost [Axelsson 2011] and the found information may have low value [Letier *et al.* 2014], which can lead to net benefit of the action to be negative.

Lessons learned and information acquired during past projects can be very useful in making estimations about the current project and in reducing the overall uncertainty [Atkinson *et al.* 2006], but the knowledge gained during the project may be lost after the project finishes [Perminova *et al.* 2008]. Despite the importance of past data, organizations do not always collect the data, or its availability is too limited to make it useful [Atkinson *et al.* 2006]. Investing in proper knowledge management and sharing can be beneficial for the success rates of projects.

Dönmez and Grote [2018] found ten practices that are used to manage uncertainty by agile software development teams. They organized their findings in four groups of principles where practices within the group share common characteristics. The principles are "uncertainty anticipation", "information accrual", "solution inspection", and "role-based coordination". The ten practices found by Dönmez and Grote [2018] are "incorporating uncertainty in plans", "developing vigilance", "incremental feedback", "team based task analysis", "knowledge sharing", "prototyping", "creating alternatives, "creating functional roles, "stakeholder integration, and "task switching". These principles and practices are shown in Table 10 with more detailed descriptions of each practice. The practices partly overlap and supplement uncertainty reducing actions discussed above.

Table 10. Uncertainty management practices [Dönmez and Grote 2018].

Principle	Practice	Description
Uncertainty anticipation	1. Incorporating uncertainty in plans	Acknowledging that changes will occur, teams try to anticipate and focus on product requirements least expected to change, yet remain flexible to adapt their plans.
	2. Developing vigilance	Team members strive to remain alert to opportunities that present themselves.
Information accrual	3. Incremental feedback	Team members frequently collect feedback from colleagues and external project stakeholders.
	4. Team based task analysis	Team members collectively analyze requirements before they create and plan tasks.
	5. Knowledge sharing	Knowledge management structures are established to manage resource uncertainty.
Solution inspection	6. Prototyping	Preliminary versions of the final product foster discussions on functionality and indicate the quality of different possible solutions.
	7. Creating alternatives	Developers strive to explore several solutions in parallel to determine which best fits customer expectations.
Role-based coordination	8. Creating functional roles	Team member roles are created temporally in order to handle unexpected events efficiently.
	9. Stakeholder integration	Teams value close collaboration with suppliers and clients and design decision structures accordingly.
	10. Task switching	Teams aim to create structures that permit developers to flexibly distribute tasks to freed resources.

5.2 Coping with uncertainty

While multiple different strategies and practices for reducing uncertainty in software development projects can be found in the literature, the methods for coping with uncertainty are by far fewer [Dönmez and Grote 2018]. The approaches found in this literature review can be roughly categorized in approaches for project manager and those of individual developers.

Marinho and others [2014] state that while project managers can try to reduce the uncertainty, it will never be a complete success. Unexpected situations will arise during the project and there are four strategies available that can be used to cope with the uncertainty. Suppression is aimed at minimizing the impact of unexpected situation so that the project can be steered back to the initial plan. Adapt is to accept some level of uncertainty and to be ready to react and contain the impacts of unexpected events if needed. Detour aims to divert away from the areas of uncertainty, but most of the time this is not an available option. Not all uncertainties can be avoided, or the action might just introduce more uncertainties. Reorient is a complete overhaul of the project objectives and should be used only if it is the only option. [Marinho *et al.* 2014]

High degree of freedom of the project team is connected to better handling of unexpected events while micromanagement and tight control by higher hierarchical levels does the opposite [Geraldi *et al.* 2010]. Trust between the project stakeholders increases the team cohesion and more cohesive teams can perform better with uncertainties [Marinho *et al.* 2018]. Trusting the skills and understanding of other team members can in itself also facilitate better success when facing uncertainty [Geraldi *et al.* 2010].

Taipalus and others [2020] identified four mechanisms that can be used to cope with uncertainty in software development: “change in attitude”, “emphasizing roles”, “openness in communication”, and “involvement with the client”. Some of these mechanisms can be actualized by project management or the organization and others by individual developers. These uncertainty coping mechanisms are shown in Table 11 with examples that give further details of the coping mechanisms.

Table 11. Uncertainty coping mechanisms [Taipalus *et al.* 2020].

Coping mechanism	**Example**
Change in attitude	Accept uncertainty as an integral element in software development and that learning new technologies etc. is a continuous process when working with software.
	Organizations should support developers to experiment and be tolerant to resulting inevitable failures. The failures could be taken as lessons learned if handled properly.
Emphasizing roles	Developers need to know who they can ask help in technical or managerial problems.
	Developers should know their role in the organization and the roles of other people related to their work.
	Distribution of responsibilities should be clear within team.
	Organization should make it clear to the client that the developers are experts in software development and the client is the expert in their own business domain.
Openness in communication	Trust is important in software development and open admitting of possible problems, uncertainties or failures increases the trust with the client and within the development team. Finding out problems that have been left untold decreases the trust between project parties.
Involvement with the client	Active participation of the client during the whole project life cycle can prevent unnecessary work and psychological problems.
	Developers and the client should invest time in the initial requirements analysis and the client should also describe their business domain thoroughly enough that the developers understand the reason for the needed features.
	The end-users should be involved with the development in activities like acceptance testing or participatory design.

6 Uncertainty and software development processes

There are a lot of differences between software development projects, and they should be characterized properly based on their attributes so that the correct management method can be chosen [Marinho *et al.* 2018]. Type and amount of uncertainty is one such attribute [De Meyer *et al.* 2002]. Another important factor to consider when choosing management approach and structure for project is how critical the project is [Howell *et al.* 2010].

Two commonly compared distinct software development processes are waterfall software methodology and agile software methodology [Aitken and Ilango 2013]. Waterfall approach is usually today referred to as the traditional or plan-driven software development process. It is said to originate from Winston Royce's publication from 1970 [Clarke and O'Connor 2012]. Agile methodology is based on Agile Manifesto that was published in 2001 and has since made a considerable change to how software development is done today [Dingsøyr *et al.* 2012].

Comparing these two alternative approaches of software development management is somewhat of a straw-man comparison [Aitken and Ilango 2013]. In reality, it is common for software development projects to use elements from both of these methodologies and pure adaptations of these approaches are rare in practice. Projects that are managed with a plan-driven process usually use also iterative and incremental workflows [Aitken and Ilango 2013], which are defining features of Agile methods. According to Aitken and Ilango [2013], projects managed with Agile process may also use some elements usually addressed to the waterfall approach.

Even though the comparison of waterfall process and Agile process is somewhat problematic, that is the approach used in this thesis. As project teams tend to use elements from both methodologies, they should both be understood. In the following sections an overview of both of these software development processes is given first, and finally these processes are discussed in relation to uncertainty.

6.1 Traditional software development

Traditional software development process is specifically characterized by doing most of the planning and design work in the start of the project and then proceeding with sequentially completed steps that tend to be large independent parts of the project. The approach is linear and predictable [Gemino *et al.* 2021]. The purpose of emphasizing the upfront collecting of requirements and planning is to anticipate and avoid surprises [Howell *et al.* 2010] so that the process can be done in steps that are completed in a rigid manner from start to finish one at a time. The sequential flow of the steps in waterfall method is shown in Figure 1 as presented by Whitaker [2009, p. 252]. There is a slight overlap between the steps in time, but generally the steps are flowing from previous to the next one in a manner similar to a waterfall [Whitaker 2009, p. 252].

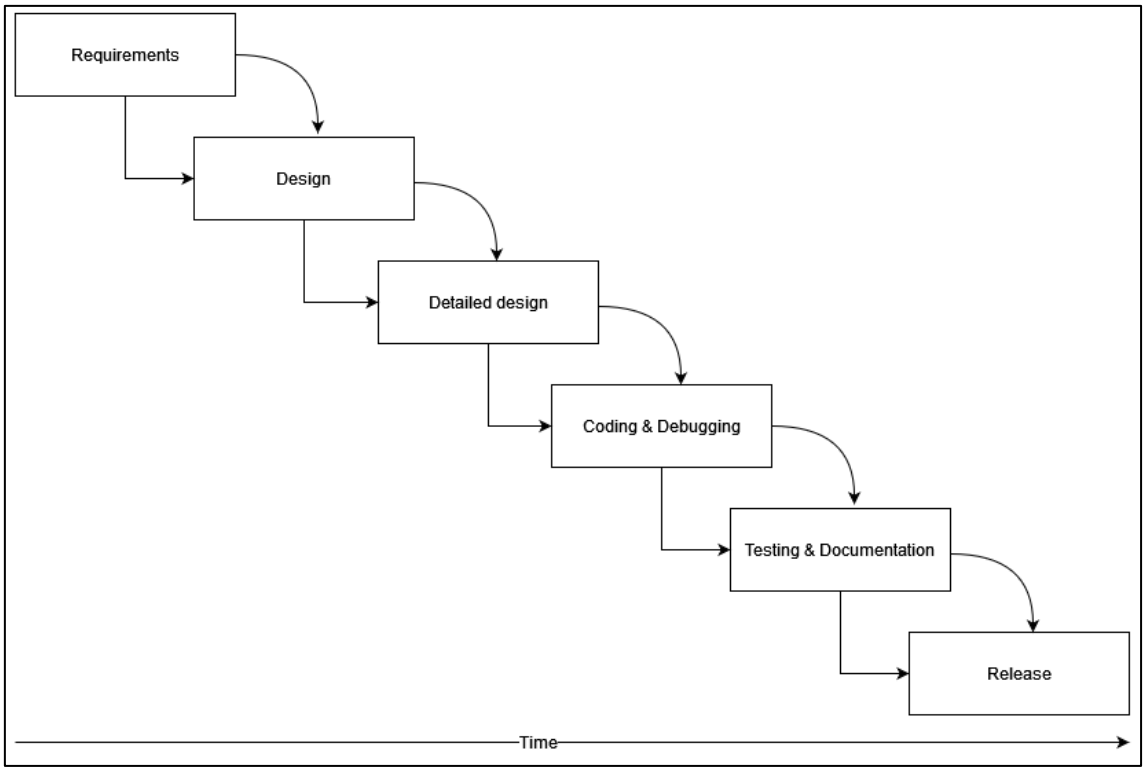

Figure 1. Waterfall software methodology flow [Whitaker 2009, p. 252].

In addition to heavy upfront planning and sequential workflow there are also other typical characteristics in traditional software development process. These are shown in Table 12 according to Whitaker [2009, p. 251].

Table 12. Characteristics of waterfall software methodology [Whitaker 2009, p. 251].

Feature	**Description**
Specifications	Exactly defined and lots of them.
Schedules	Made with precise delivery dates.
Sequence of events	One process following another in linear manner.
Adaptability to changes	Rigid and not adaptable.
Understandability	Easy to understand even without technical background.
Usefulness to the team	Not very useful, but management may like it.
Customer involvement	Usually at or near the end of project.

6.2 Agile software development

Agile software development is a more modern, fundamentally different development methodology than waterfall approach. Agile way of working originates from the year 2001 when the Agile Manifesto was released by a group of developers, who wanted to improve how software was developed. Agile Manifesto highlights a set of four values, that should be the guiding principle in developing software: "individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan" [Beck *et al.* 2001]. In addition to the four core values, the manifesto declared twelve principles that should be followed including welcoming change in requirements, frequent and constant delivery of working software, and preferring face-to-face communication [Beck *et al.* 2001].

Agile Manifesto defines the values and principles quite broadly and different people can interpret them in various ways [Aitken and Ilango 2013]. After the publication of the manifesto there has been multiple different methodologies that are more or less based on Agile Manifesto [Dingsøyr *et al.* 2012]. Some of these methodologies include eXtreme Programming aka XP, Scrum, Crystal Methods, and Feature Driven Development aka FDD [Dingsøyr *et al.* 2012]. These methodologies have gained wide popularity amongst the software developers and changed the industry along the way [Eloranta *et al.* 2016]. Particularly Scrum has become extremely popular [Eloranta *et al.* 2016].

When compared to traditional software development, agile has completely different approach to change: the former abhors the change while agile embraces the change [Sillitti *et al.* 2005]. The work is divided in multiple short iterations, where every iteration goes through about the same steps that are included in the waterfall process [Whitaker 2009, p. 256] and the project progresses step by step towards a more complete product. Iterations are typically between two to four weeks long and at the end of each iteration, a working version of the software should be released if possible [Whitaker 2009, pp. 256-257]. Requirements are gathered in all iterations, which allows the customers to specify their needs more accurately as the work progresses and more is understood about the project [Sillitti *et al.* 2005]. Customers are kept close to the development team to be able to get direct feedback from them [Dingsøyr *et al.* 2012], which can help shaping the project further towards the goal [Whitaker 2009, p. 256]. The architecture of the project is kept as flexible as possible by creating first only the minimum architecture needed to implement the current requirements and delaying the binding architectural decisions until more is known when possible [Sillitti *et al*. 2005].

According to Cockburn and Highsmith [2001], agile development might not be suitable for all teams. Agile software development is best suited for co-located teams with no more

than 50 people at maximum [Williams and Cockburn 2003]. Teams need to have mutual trust and respect, and they must be able to deal with ambiguity [Cockburn and Highsmith 2001]. The projects developed by agile teams would better not be life-critical [Williams and Cockburn 2003]. Organization culture may also turn out to be a problem causing a failure in adopting agile principles [Cockburn and Highsmith 2001]. Organizational skills are important for individual developers working with agile methods and because of this, developers with poor organizational skills may experience severe problems when working in agile environment [Venkatesh *et al.* 2020].

Since Scrum is the most popular Agile method at the moment [Hoda *et al.* 2018], it will be described in further detail. It should be noted though, that companies usually do not follow all practices of Scrum to the letter, but instead they use the ideas and practices that fit them the best [Eloranta *et al.* 2016]. This approach is known as ScrumBut [Eloranta *et al.* 2016].

The teams in Scrum should be cross-functional and self-organized. In practice this means that teams should have all the needed expertise to build the software products and that teams members coordinate their work by themselves. There are only three distinct roles in Scrum teams: Product owner, Scrum master and the developers. Product owner is the representative of the customer and manages the so-called Product Backlog. Scrum master is responsible for mentoring the team, removing any obstacles slowing the team down and ensures that the Scrum process is followed. Developers are responsible for developing the project. [Eloranta *et al.* 2016]

The project developed using Scrum is divided into Sprints, which are short iterations lasting usually from two to four weeks. Project requirements are collected in Product Backlog as Product Backlog Items. In one Sprint, a set of Product Backlog Items are chosen to be implemented during that Sprint. The chosen items are transformed into a task list, which forms a plan for implementing the chosen Product Backlog Items. The items on the list are called Sprint Backlog Items and the list itself is called Sprint Backlog. During the Sprint these Sprint Backlog Items are implemented. The team have a short meeting daily which is called Daily Scrum. Ideally after the Sprint, a working version of the product can be delivered to the customer. After the Sprint is finished, two meetings are held. Sprint review is a meeting where the improvements implemented during Sprint are shown to key stakeholders to gather feedback. Retrospective is a meeting where the team to discusses how the working process could be improved and what could be done better in future. [Eloranta *et al.* 2016]

6.3 Processes in relation to uncertainty

The literature discusses surprisingly little how suitable different software development processes are in relation to uncertainty, considering that development processes have such

fundamental differences between them. There seems to be a consensus that major factors in suitability of the development process are the type and characteristics of the project. When the level of uncertainty in the project is high, the recommendation is to not use the waterfall approach [Moynihan 2000]. The difficulties in using the waterfall methodology might increase considerably if the project has long timespan in addition to high level of uncertainty [Johansen *et al.* 2014].

The projects tend to range from those that have clear and precisive goals at the beginning to those that have ill-defined purposes and stakeholders may disagree with the goals or have different expectations about the end product [Atkinson *et al.* 2006]. In a typical project the uncertainty is at its highest at the beginning and will gradually get lower as the project progresses, but in projects developed in changing environments and with high complexity the uncertainty may stay on high levels for the whole duration of the project [Jaafari 2001].

If the level of uncertainty in project is low, the planning and control approach works well [Jun *et al.* 2011] and with high levels of uncertainty the role of flexibility and learning is emphasised, and planning is less effective [De Meyer *et al.* 2002]. Obviously in innovation projects the need for flexibility is high, while projects involving risk for human life or environment the need for control is of paramount importance [Grote 2015].

Flexibility is needed for managing uncertainties and unexpected events [Dönmez and Grote 2013], while stability is required for control [Grote 2015]. While at first glance these attributes might seem to exclude each other, they might actually support each other in some cases and they both offer distinct advantages [Grote 2015]. Use of routines and standards create stability and increase control and predictability, which consequently reduces the uncertainties involved, while flexibility helps in learning and adapting to situations with uncertainty [Grote 2015]. Both flexibility and stability are needed in projects and a correct balance between them is important for succeeding in managing the uncertainties [Dönmez and Grote 2013].

The problems with relying too much on plans made in the early phase of the project are clear. Unexpected events are bound to happen eventually and coping with them is simply not possible by calculating them beforehand or managing them with sticking to the plan [Böhle *et al.* 2016]. The plans are made with assumptions about variety of things and with increased uncertainty these assumptions are farther from the actual reality [Howell *et al.* 2010]. Precise early planning is often not an option, since usually the projects contain initial unknown variables, knowledge needed for crucial decisions might not be available or situations may change during the course of the project [Jaafari 2001].

The agile approach to executing the development in short iterations helps to address these problems. When the plans and the project are created gradually step-by-step during the

iterations, the information needed for decisions is easier to get at the right time and the inherent flexibility of the process helps to react to unexpected changes. The tendency to break the project into small tasks and get continuous feedback from the customer also supports in managing the uncertainty [Dönmez and Grote 2018]. Including the user expert as part of the development team enables rapid feedback on advancements and at the same time, this makes it easier to notice if the developers have misunderstood requests or if some of the user requests are not truly working as intended [Cockburn and Highsmith 2001].

In general sense agile software development is better suited to managing uncertainty than traditional software development, but the uncertainty management in agile approaches is indirect and passive [Dönmez and Grote 2018]. Better understanding of uncertainty and deliberate managing of it could increase the success of dealing with uncertainty. It should be kept in mind though, that some aspects of agile might also increase the uncertainty like high levels of autonomy related to self-organizing teams [Aitken and Ilango 2013] and the strive to embrace the change.

In some projects the traditional software development might still be the better choice. When the project requires strict control, stability and adherence to standards, regulations etc., the waterfall-style project management delivers these qualities. Waterfall approach has some obvious flaws though, like the inability to react properly to unexpected changes and the difficulty to create a foolproof plan at the start of the project when many relevant variables are still unknown.

7 Summary

It is common for software development projects to not succeed as intended and one of the reasons for this is uncertainty. Uncertainty is an inherent, ever-present phenomenon in software development that has many types and sources and affects all stakeholders in the projects. In most cases, uncertainty is seen solely as a threat, but sometimes it can also be a possibility.

This thesis explored the concept of uncertainty in the context of software development by conducting a literary review on the topic. Due to lack of academic papers focused especially on uncertainty in the context of software development, some studies are used from other disciplines as well. The purpose of this thesis was to develop deeper understanding of uncertainty by examining causes, effects and managing options of uncertainty, and how suitable different software development processes are in relation to uncertainty.

Definition of the term uncertainty is elusive and ambiguous, hence there is no consensus about the exact definition of uncertainty. As a concept it is multifaceted and it can mean different things depending on who uses the term, but most studies define uncertainty as lack of information or knowledge with slight variations in deeper details. Also concepts of ambiguity, sense of doubt, and unexpected events are linked to uncertainty. Term risk has been commonly associated with uncertainty, but they are distinct concepts that have similarities but are fundamentally different.

Uncertainty has different types and sources, which should be recognized since different uncertainties require different approaches. To distinguish these from each other, a categorization is necessary. This thesis used a classification where type represents broadly what the uncertainty is about, and the source represents what is causing the uncertainty. Some of the uncertainty types included for example requirements, situation, variability, and technology. The sources of uncertainty included complexity, ambiguity, lack of trust and lack of information for example.

Effects of uncertainty are wide ranging, and they have impact on all stakeholders of the project. Those effects may be positive or negative, but most of the time the positive effects are ignored. Negative effects on the development projects include for example delays in schedule, budget overruns, decreased product quality, wasted time and poor estimates. Positive effects on the project can be for example innovation and new solutions. Development can also lead to components that can be used in multiple projects to improve efficiency. On the individual developers the negative effects of uncertainty include stress and dysphoria, feelings of inadequacy and reduction in cognitive performance for example. The positive effects of uncertainty on individual developer can be like possibility of self-improvement and motivation.

Uncertainty management is divided into reducing uncertainty and coping with uncertainty. Removing all uncertainties is practically impossible, and it would not be recommended anyway, since the uncertainties can also turn into opportunities. There are many ways to reduce the uncertainty and the approach depends on the type of uncertainty. For example, uncertainty related to stakeholders can be reduced by maintaining direct and continuous communication with them during the project and making sure that the developers and stakeholders have a shared terminology to evade misunderstandings. Uncertainty related to changing requirements can be reduced by dividing the development of the project to short cyclical iterations, where the project is completed step by step gradually progressing towards the completed project. Coping with uncertainty is easier when the development team has high degree of autonomy, and they trust in each other. Mechanisms that facilitate coping with uncertainty are for example change in attitude and emphasizing roles.

When considering the suitability of a software development process in relation to uncertainty, major influencing factors are the type and characteristics of the project. If the level of uncertainty is low or the project is dealing with a life-critical product, then traditional software development process might be a good choice. When the level of uncertainty is high, then the use of agile software development process is recommended. Traditional software development process offers stability and control but lacks in ability to respond to changes. Agile software development process offers flexibility and ability to respond to changes among other things, but at the same time may the process increase uncertainties due to some aspects of the methodology.

This thesis has limitations related to the research method, source material used, and the subject itself. A more in-depth systematic literature review could give more robust results, but it was not considered a possibility for this work because of the fuzziness involved with the topic and the lack of studies exploring exactly the same context. Since the amount of studies concentrated on uncertainty in the context of software development is limited, this narrows the viewpoint and the depth of this literary review. Another significant limitation relates to the definition of the term uncertainty. Since there is no consensus of the exact definition, it causes the whole subject to be confusing to a degree.

More research on the subject is needed. Some directions for the future research could include defining the term uncertainty with more confidence and creating a solid classification scheme for different types and sources of uncertainty. From the context of software development, the psychological side of uncertainty should be researched more for the sake of finding more coping methods for the inevitable uncertainty.

References

Aitken, A., & Ilango, V. (2013). A comparative analysis of traditional software engineering and agile software development. In: *2013 46th Hawaii International Conference on System Sciences*, 4751–4760. https://doi.org/10.1109/HICSS.2013.31

Atkinson, R., Crawford, L., & Ward, S. (2006). Fundamental uncertainties in projects and the scope of project management. *International Journal of Project Management*, 24(8), 687–698. Scopus. https://doi.org/10.1016/j.ijproman.2006.09.011

Axelsson, J. (2011). On how to deal with uncertainty when architecting embedded software and systems. In: I. Crnkovic, V. Gruhn, & M. Book (Eds.), *Software Architecture* (pp. 199–202). Springer. https://doi.org/10.1007/978-3-642-23798-0_20

Bannerman, P. L. (2008). Risk and risk management in software projects: A reassessment. *Journal of Systems and Software*, 81(12), 2118–2133. https://doi.org/10.1016/j.jss.2008.03.059

Beck, K., Beedle, M., Bennekum, A. van, Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001). Manifesto for Agile Software Development. http://agilemanifesto.org/

Böhle, F., Heidling, E., & Schoper, Y. (2016). A new orientation to deal with uncertainty in projects. *International Journal of Project Management*, 34(7), 1384–1392. Scopus. https://doi.org/10.1016/j.ijproman.2015.11.002

Carleton, R. N. (2016). Into the unknown: A review and synthesis of contemporary models involving uncertainty. *Journal of Anxiety Disorders*, 39, 30–43. https://doi.org/10.1016/j.janxdis.2016.02.007

Clarke, P., & O'Connor, R. V. (2012). The situational factors that affect the software development process: Towards a comprehensive reference framework. *Information and Software Technology*, 54(5), 433–447. https://doi.org/10.1016/j.infsof.2011.12.003

Cockburn, A., & Highsmith, J. (2001). Agile software development, the people factor. *Computer*, 34(11), 131–133. https://doi.org/10.1109/2.963450

De Meyer, A., Loch, C. H., & Pich, M. T. (2002). Managing project uncertainty: From variation to chaos. *MIT Sloan Management Review*, 43(2), 60–67.

Dingsøyr, T., Nerur, S., Balijepally, V., & Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software*, 85(6), 1213–1221. Scopus. https://doi.org/10.1016/j.jss.2012.02.033

Dönmez, D., & Grote, G. (2013). The practice of not knowing for sure: How agile teams manage uncertainties. *Lecture Notes in Business Information Processing*, 149, 61–75. Scopus. https://doi.org/10.1007/978-3-642-38314-4_5

Dönmez, D., & Grote, G. (2015). The two faces of uncertainty: Threat vs Opportunity management in agile software development. *Lecture Notes in Business Information Processing*, 212, 193–198. Scopus. https://doi.org/10.1007/978-3-319-18612-2_16

Dönmez, D., & Grote, G. (2018). Two sides of the same coin – how agile software development teams approach uncertainty as threats and opportunities. *Information and Software Technology*, 93, 94–111. https://doi.org/10.1016/j.infsof.2017.08.015

Eloranta, V.-P., Koskimies, K., & Mikkonen, T. (2016). Exploring ScrumBut—An empirical study of Scrum anti-patterns. *Information and Software Technology*, 74, 194–203. Scopus. https://doi.org/10.1016/j.infsof.2015.12.003

Gemino, A., Horner Reich, B., & Serrador, P. M. (2021). Agile, traditional, and hybrid approaches to project success: Is hybrid a poor second choice? *Project Management Journal*, 52(2), 161–175. Scopus. https://doi.org/10.1177/8756972820973082

Geraldi, J. G., Lee-Kelley, L., & Kutsch, E. (2010). The Titanic sunk, so what? Project manager response to unexpected events. *International Journal of Project Management*, 28(6), 547–558. https://doi.org/10.1016/j.ijproman.2009.10.008

Greco, V., & Roger, D. (2003). Uncertainty, stress, and health. *Personality and Individual Differences*, 34(6), 1057–1068. https://doi.org/10.1016/S0191-8869(02)00091-0

Grote, G. (2015). Promoting safety by increasing uncertainty – Implications for risk management. *Safety Science*, 71, 71–79. https://doi.org/10.1016/j.ssci.2014.02.010

Hillson, D. (2002). Extending the risk process to manage opportunities. *International Journal of Project Management*, 20(3), 235–240. Scopus. https://doi.org/10.1016/S0263-7863(01)00074-6

Hoda, R., Salleh, N., & Grundy, J. (2018). The Rise and evolution of agile software development. *IEEE Software*, 35(5), 58–63. https://doi.org/10.1109/MS.2018.290111318

Howell, D., Windahl, C., & Seidel, R. (2010). A project contingency framework based on uncertainty and its consequences. *International Journal of Project Management*, 28(3), 256–264. https://doi.org/10.1016/j.ijproman.2009.06.002

Hughes, D. L., Rana, N. P., & Simintiras, A. C. (2017). The changing landscape of IS project failure: An examination of the key factors. *Journal of Enterprise Information Management*, 30(1), 142–165. https://doi.org/10.1108/JEIM-01-2016-0029

Ibrahim, H., Far, B. H., Eberlein, A., & Daradkeh, Y. (2009). Uncertainty management in software engineering: Past, present, and future. In: *2009 Canadian Conference on Electrical and Computer Engineering*, 7–12. https://doi.org/10.1109/CCECE.2009.5090081

Jaafari, A. (2001). Management of risks, uncertainties and opportunities on projects: Time for a fundamental shift. *International Journal of Project Management*, 19(2), 89–101. https://doi.org/10.1016/S0263-7863(99)00047-2

Jiang, J. J., Klein, G., Wu, S. P. J., & Liang, T. P. (2009). The relation of requirements uncertainty and stakeholder perception gaps to project management performance. *Journal of Systems and Software*, 82(5), 801–808. https://doi.org/10.1016/j.jss.2008.11.833

Johansen, A., Halvorsen, S. B., Haddadic, A., & Langlo, J. A. (2014). Uncertainty management – A methodological framework beyond "The Six W's". *Procedia - Social and Behavioral Sciences*, 119, 566–575. https://doi.org/10.1016/j.sbspro.2014.03.063

Jun, L., Qiuzhen, W., & Qingguo, M. (2011). The effects of project uncertainty and risk management on IS development project performance: A vendor perspective. *International Journal of Project Management*, 29(7), 923–933. https://doi.org/10.1016/j.ijproman.2010.11.002

Laplante, P. A., & Neill, C. J. (2005). Uncertainty: A meta-property of software. In: *29th Annual IEEE/NASA Software Engineering Workshop*, 228–233. https://doi.org/10.1109/SEW.2005.48

Lechler, T. G., Edington, B. H., & Gao, T. (2012). Challenging classic project management: Turning project uncertainties into business opportunities. *Project Management Journal*, 43(6), 59–69. Scopus. https://doi.org/10.1002/pmj.21304

Letier, E., Stefan, D., & Barr, E. T. (2014). Uncertainty, risk, and information value in software requirements and architecture. In: *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*, 883–894. Scopus. https://doi.org/10.1145/2568225.2568239

Lipshitz, R., & Strauss, O. (1997). Coping with uncertainty: A naturalistic decision-making analysis. *Organizational Behavior and Human Decision Processes*, 69(2), 149–163. https://doi.org/10.1006/obhd.1997.2679

Madsen, S. (2007). Conceptualising the causes and consequences of uncertainty in IS development organisations and projects. In: *Proceedings of the 15th European Conference on Information Systems (ECIS) 2007*. 855–864.

Marinho, M., Sampaio, S., Luna, A., Lima, T., & Moura, H. (2015). Dealing with uncertainties in software project management. In: *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, 889–894. https://doi.org/10.1109/CIT/IUCC/DASC/PICOM.2015.133

Marinho, M., Sampaio, S., & Moura, H. (2014). Uncertainties in software projects management. In: *2014 XL Latin American Computing Conference (CLEI)*, 1–10. https://doi.org/10.1109/CLEI.2014.6965153

Marinho, M., Sampaio, S., & Moura, H. (2018). Managing uncertainty in software projects. *Innovations in Systems and Software Engineering*, 14(3), 157–181. Scopus. https://doi.org/10.1007/s11334-017-0297-y

Martinsuo, M., Korhonen, T., & Laine, T. (2014). Identifying, framing and managing uncertainties in project portfolios. *International Journal of Project Management*, 32(5), 732–746. Scopus. https://doi.org/10.1016/j.ijproman.2014.01.014

Mehta, N., Hall, D., & Byrd, T. (2014). Information technology and knowledge in software development teams: The role of project uncertainty. *Information & Management*, 51(4), 417–429. https://doi.org/10.1016/j.im.2014.02.007

Moynihan, T. (2000). Coping with 'requirements-uncertainty': The theories-of-action of experienced IS/software project managers. *Journal of Systems and Software*, 53(2), 99–109. https://doi.org/10.1016/S0164-1212(00)00049-2

Na, K.-S., Li, X., Simpson, J. T., & Kim, K.-Y. (2004). Uncertainty profile and software project performance: A cross-national comparison. *Journal of Systems and Software*, 70(1), 155–163. https://doi.org/10.1016/S0164-1212(03)00014-1

Ostberg, J.-P., Graziotin, D., Wagner, S., & Derntl, B. (2020). A methodology for psycho-biological assessment of stress in software engineering. *PeerJ Computer Science*. https://doi.org/10.7717/peerj-cs.286

Perminova, O., Gustafsson, M., & Wikström, K. (2008). Defining uncertainty in projects – a new perspective. *International Journal of Project Management*, 26(1), 73–79. https://doi.org/10.1016/j.ijproman.2007.08.005

Rezvani, A., & Khosravi, P. (2019). Emotional intelligence: The key to mitigating stress and fostering trust among software developers working on information system projects. *International Journal of Information Management*, 48, 139–150. https://doi.org/10.1016/j.ijinfomgt.2019.02.007

Saunders, F. C., Gale, A. W., & Sherry, A. H. (2015). Conceptualising uncertainty in safety-critical projects: A practitioner perspective. *International Journal of Project Management*, 33(2), 467–478. https://doi.org/10.1016/j.ijproman.2014.09.002

Sillitti, A., Ceschi, M., Russo, B., & Succi, G. (2005). Managing uncertainty in requirements: A survey in documentation-driven and agile companies. In: *11th IEEE International Software Metrics Symposium (METRICS'05)*, 10–17. https://doi.org/10.1109/METRICS.2005.29

Sonnentag, S., Brodbeck, F. C., Heinbokel, T., & Stolte, W. (1994). Stressor-burnout relationship in software development teams. *Journal of Occupational & Organizational Psychology*, 67(4), 327–341. https://doi.org/10.1111/j.2044-8325.1994.tb00571.x

Taipalus, T., Seppänen, V., & Pirhonen, M. (2020). Uncertainty in information system development: Causes, effects, and coping mechanisms. *Journal of Systems and Software*, 168, 110655. https://doi.org/10.1016/j.jss.2020.110655

Ubayashi, N., Kamei, Y., & Sato, R. (2019). Modular programming and reasoning for living with uncertainty. In: M. van Sinderen & L. A. Maciaszek (Eds.), *Software Technologies* (pp. 220–244). Springer International Publishing. https://doi.org/10.1007/978-3-030-29157-0_10

Venkatesh, V., Thong, J. Y. L., Chan, F. K. Y., Hoehle, H., & Spohrer, K. (2020). How agile software development methods reduce work exhaustion: Insights on role perceptions and organizational skills. *Information Systems Journal*, 30(4), 733–761. Scopus. https://doi.org/10.1111/isj.12282

Wang, Q. Z., & Liu, J. (2006). Project uncertainty, management practice and project performance: An empirical analysis on customized information systems development projects. In: *2006 IEEE International Engineering Management Conference*, 341–345. https://doi.org/10.1109/IEMC.2006.4279882

Ward, S., & Chapman, C. (2003). Transforming project risk management into project uncertainty management. *International Journal of Project Management*, 21(2), 97–105. https://doi.org/10.1016/S0263-7863(01)00080-1

Ward, S., & Chapman, C. (2008). Stakeholders and uncertainty management in projects. *Construction Management and Economics*, 26(6), 563–577. https://doi.org/10.1080/01446190801998708

Whitaker, K. (2009). *Principles of Software Development Leadership: Applying Project Management Principles to Agile Software Development*. Course Technology, 2009.

Williams, L., & Cockburn, A. (2003). Agile software development: It's about feedback and change. *Computer*, 36(6), 39–43. https://doi.org/10.1109/MC.2003.1204373