St. Petersburg University
Graduate School of Management

Master in Information Technologies and Innovation Management

TECHNICAL DEBT MANAGEMENT IN RUSSIAN SOFTWARE DEVELOPMENT COMPANIES

Master's Thesis by the 2nd year student
Concentration – MITIM
Grinevskaia Iuliia

Research advisor:
Dmitry Kudryavtsev,
Associate Professor

St. Petersburg
2017

ЗАЯВЛЕНИЕ О САМОСТОЯТЕЛЬНОМ ХАРАКТЕРЕ ВЫПОЛНЕНИЯ ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Я, Гриневская Юлия Ивановна, студент второго курса магистратуры направления «Менеджмент», заявляю, что в моей магистерской диссертации на тему «Управление техническим долгом в российских компаниях-разработчиках программного обеспечения», представленной в службу обеспечения программ магистратуры для последующей передачи в государственную аттестационную комиссию для публичной защиты, не содержится элементов плагиата.

Все прямые заимствования из печатных и электронных источников, а также из защищенных ранее выпускных квалификационных работ, кандидатских и докторских диссертаций имеют соответствующие ссылки.

Мне известно содержание п. 9.7.1 Правил обучения по основным образовательным программам высшего и среднего профессионального образования в СПбГУ о том, что «ВКР выполняется индивидуально каждым студентом под руководством назначенного ему научного руководителя», и п. 51 Устава федерального государственного бюджетного образовательного учреждения высшего профессионального образования «Санкт-Петербургский государственный университет» о том, что «студент подлежит отчислению из Санкт-Петербургского университета за представление курсовой или выпускной квалификационной работы, выполненной другим лицом (лицами)».

_____ (Подпись студента)

29.05.2017 (Дата)

STATEMENT ABOUT THE INDEPENDENT CHARACTER OF THE MASTER THESIS

I, Grinevskaia Iuliia, second year master student, program «Management», state that my master thesis on the topic «Technical debt management in Russian software development companies», which is presented to the Master Office to be submitted to the Official Defense Committee for the public defense, does not contain any elements of plagiarism.

All direct borrowings from printed and electronic sources, as well as from master theses, PhD and doctorate theses which were defended earlier, have appropriate references.

I am aware that according to paragraph 9.7.1. of Guidelines for instruction in major curriculum programs of higher and secondary professional education at St.Petersburg University «A master thesis must be completed by each of the degree candidates individually under the supervision of his or her advisor», and according to paragraph 51 of Charter of the Federal State Institution of Higher Professional Education Saint-Petersburg State University «a student can be expelled from St.Petersburg University for submitting of the course or graduation qualification work developed by other person (persons)».

_____(Student's signature)

29.05.2017 (Date)

ABSTRACT

Master Student's Name	Iuliia I. Grinevskaia
Master Thesis Title	Technical debt management in Russian software development companies
Faculty	Graduate School of Management
Main Field of Study	Information Technologies and Innovation Management
Year	2017
Academic Advisor's Name	Candidate of Science, Associate Professor, Dmitry V. Kudryavtsev
Description of the goal, tasks and main results	The concept of technical debt is relatively new in scientific researches, moreover, this concept plays an important role in modern software development companies. In this paper, technical debt management in Russian software companies was investigated. The purpose of this research is to study the reasons of the emergence of technical debt, the ways to manage technical debt, and also to identify factors that affect the decision-making on technical debt management. Three Russian software companies were investigated. An important idea in the study of technical debt in these companies was to understand the context of software development, which includes the market in which the company operates, the development process, the structure and size of the development team, and the age and the history of the system development in the company. As results of this study, the common for all companies reasons for the emergence of technical debt, the ways of managing it, were identified. Furthermore, there were identified common factors that influenced the decision-making on the management of technical debt. In addition, the main differences in the methods of managing technical debt in companies operating in different markets were found as well as some recommendations were given.
Keywords	Technical Debt, Technical Debt management, software development

АННОТАЦИЯ

Автор	Гриневская Юлия Ивановна
Название магистерской диссертации	Управление техническим долгом в российских компаниях-разработчиках программного обеспечения
Факультет	Высшая Школа Менеджмента
Направление подготовки	Информационные технологии и инновационный менеджмент
Год	2017
Научный руководитель	Кандидат наук, доцент Кудрявцев Дмитрий Вячеславович
Описание цели, задач и основных результатов	Концепция технического долга является относительно новой в научных исследованиях, кроме того, эта концепция играет важную роль в сфере разработки программного обеспечения. В данной работе было исследовано управление техническим долгом в российских компаний-разработчиках программного обеспечения. Целью данного исследования является изучение причин возникновения технического долга, способов управления техническим долгом, а также выявление факторов, которые влияют на принятие решений об управлении техническим долгом. В работе были исследованы три российские компании-разработчики программного обеспечения. Важная роль в изучении технического долга в указанных компаниях отводилось пониманию контекста разработки программного обеспечения, который включает рынок, на котором оперирует компания, процесс разработки, состав и размер команды разработки, а также возраст и историю развития системы в компании. В результате данного исследования были выявлены общие для всех исследованных компаний причины возникновения технического долга, способы управления им, а также были выявленные общие факторы, которые оказывают влияние на принятие решений об управлении техническим долгом. Кроме того, были выявлены основные различия в способах управления техническим долгом в компаниях, оперирующих на разных рынках.
Ключевые слова	Технический долг, Управление техническим долгом, Разработка программного обеспечения

TABLE OF CONTENTS

INTRODUCTION

Technical debt being an emerging concept attracts high attention from researchers and practitioners. Despite wide discussions of this concept on IT conferences and practical workshops as well as in academic articles, the concept of technical debt is not fully discovered and still lacks of empirical studies and proven best practices (Falessi et al., 2014). Firstly appeared as a metaphor, that compares technical debt with financial debt in order to explain its meaning to non-technical stakeholders, technical debt concept has grown into independent area for the research. (Kruchten et al., 2012).

Technical debt management practices are also not fully investigated, moreover, it is confirmed by both – researchers and practitioners that technical debt management is context dependent, but research on technical debt and its context also remains underdeveloped and some of technical debt management activities still have lots of research areas uncovered (Fernandez-Sanchez et al., 2015, Li et al., 2015).

The aim of this research is to investigate technical debt management practices in Russian software development companies with high attention to the context of software development. The context includes several components: the type of the market on which companies operate, the structure and size of software development teams, the age of the system as well as its historical development and the processes of software development.

In order to study the concept of technical debt management in Russian software development companies the following research questions were conducted:

RQ1. How do software development process influence the sources of technical debt in Russian software development companies?

RQ2. How does the context influence technical debt management in Russian software development companies?

RQ3. What technical debt management activities could be considered as mature in Russian software development companies? What methods are used to support these activities?

RQ4. What factors should be considered during decision-making processes about managing technical debt?

In order to answer research questions above, a qualitative research was made. The empirical part of this study is represented by multiple case study. The case study investigates technical debt management peculiarities in three Russian software development companies that operate in different markets: B2C, B2B and B2G.

1. CHAPTER 1: THEORETICAL BACKGROUND

1.1.The concept of technical debt

The concept of technical debt was introduced in 1992 by the American computer engineer, Ward Cunningham, as a metaphoric definition aimed at explaining to different product stakeholders the need for refactoring (Kruchten, Nord and Ozkaya, 2012). In computer science, code refactoring is the process of changing an existing software system so that it improves its internal code structure, but does not influence its external behavior (Fowler, 2013). Cunningham explained debt metaphor in the following way. *'If we failed to make our program align with what we then understood to be the proper way to think about our financial objects, then we were going to continually stumble over that disagreement and that would slow us down which was like paying interest on a loan.'* ("Ward Explains Debt Metaphor", 2017)

Therefore, technical debt is a technical compromise, which, like a financial debt, has its interest and principal payments. Interest payments occur from extra work effort needed for the future code development, because of compromise made to the code design and structure in the past. Principal of technical debt is the process of refactoring of the existing code into better structured and designed. Companies could either continue with paying the interest on the technical debt, fully cover the debt by making code refactoring (Fowler, 2003).

The difference between financial and technical debt is presented in the Table 1

Table 1. Financial and Technical debt analogy (composed by the author)

	Financial debt	**Technical debt**
Concept definition	Amount of money owed by one party to another	Decision to defer necessary work to improve code imperfections
Interest payment	Amount of money which repays interest on a loan	Extra effort needed to the future system development, while keeping code imperfections as it is
Principal payment	Paying off the loan amount	Code refactoring

Since initially used as a metaphor to explain the technical term to non-technical stakeholders, technical debt concept has evolved and expanded from narrow coding perspective to more broad view including software architecture, design, requirements, testing and documentation,

largely due to the number of scientific researches made in that field (Kruchten, Nord and Ozkaya, 2012). The most recent studies define technical debt as follows.

'When taking short cuts and delivering code that is not quite right for the programming task of the moment, a development team incurs Technical Debt. This debt decreases productivity. This loss of productivity is the interest of the Technical Debt.' (Letouzey and Declan, 2016)

The first classification of the technical debt types was made by Steve McConnell; the representation of that taxonomy is presented in the Figure 1.

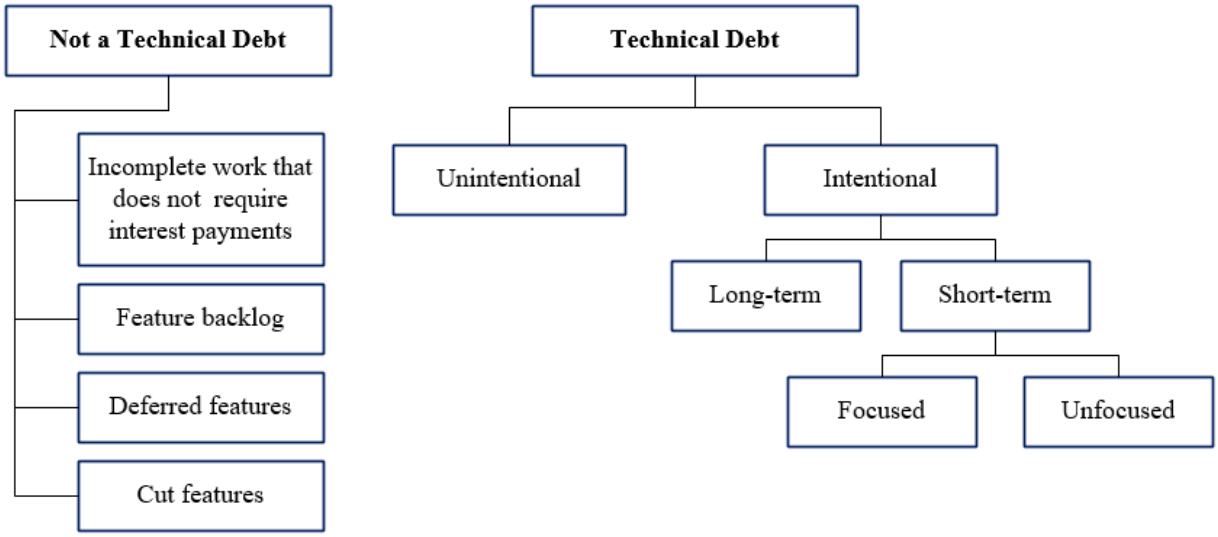

Figure 1: Technical debt classification (McConnell, 2007)

According to McConnell, the higher hierarchy of technical debt is composed of the unintentional and international debt. Unintentional technical debt usually occurs because of the poor quality of work, without any intention. For instance, it could happen when a junior computer engineer writes a low quality unstructured code, or it could be incurred unknowingly, when a company acquire another company with large amount of technical debt, which could not be identified before the acquisition.

Intentional technical debt is made by a company for a certain purpose or strategic reason by sacrificing the quality of the code to the present needs. Usually it is made in order to save time-to-market and not to lose competitive advantage in the present, to preserve startup capital or to delay development expenses. Short-term technical debt is the one that is paid off by the company for tactic reasons, which usually happens at a late stage of development or sprint to make the release possible; short-term technical debt is supposed to be covered quite often. Long-term technical debt is a strategic company's decision, which is not expected to be paid off shortly, if at all.

Furthermore, McConnell has identified what the technical debt is not. Not all incomplete work and code shortcomings should be addressed as a technical debt, but only those that require interest payments. Thus different cut and deferred features as well a feature backlog do not cause technical debt. (McConnell, 2007)

Another classification of technical debt was proposed by Martin Fowler in the form of 2x2 quadrant, where horizontal axis represents intention and vertical axis – prudence (see Figure 2). His model focused on the technical side of the technological debt – software development, was further extended by Mohan Babu K by adding a complex view of the application portfolio and enterprise architecture (Table).

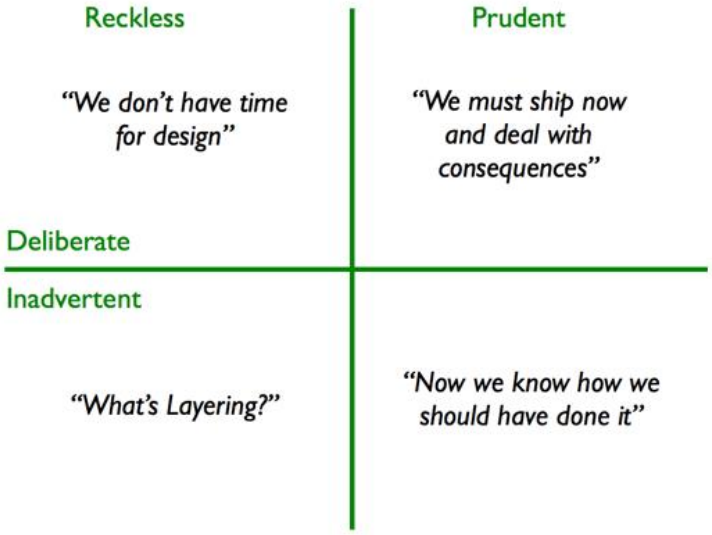

Figure 2 Technical Debt Quadrant (Fowler, 2009)

Table 2 Description of technical debt types

Type of technical debt	Description
Reckless/deliberate	Sometimes projects teams succumb to time-to-market tensions from the business or market side without necessary analysis and foresight. For instance, a financial business unit may require a different version of the accounting system to be implemented without waiting for the new ERP system to be set up across the organization. This kind of debt should be paid off, when the new global ERP system would be ready for implementation. (Babu K, 2016)
Prudent/deliberate	Sometimes project teams could deliberately take over a short-term technical debt with the explicit plans to repay it in the future. Such a decision could be a reaction for the change in the external environment. (Babu K, 2016)

Reckless/inadvertent	Such technical debt occurs in the poorly managed companies, where project teams do not know the consequences of taking technical debt or recklessly disregard the guidelines. (Babu K, 2016)
Prudent/inadvertent	Technical debt of this kind occurs when project teams might take a reasonable decision that meet the functional needs at a certain time, but which might also unexpectedly miss other situations and requirements in the organization. (Babu K, 2016)

It was proved by many researchers, that technical debt is not just a metaphor, but a serious issue that need to be addressed in a specific way. Technical debt decreases company's productivity (Letouzey, 2016) and could be a symptom of a more serious weakness in the companies organization, especially in the communication process (Declan, 2016)

Technical Debt is used as a metaphoric definition of technical compromises with which a company may cope or even may benefit from in short-run, but which may be threatening in the long run. Firstly, this metaphor related to code level issues and was introduced by Ward Cunningham about twenty years ago to clear up the need of code refactoring for nontechnical stakeholders. Since that time, the concept of technical debt started to evolve and was expanded from narrow coding perspective to more broad view including software architecture, design, requirements, testing and documentation (Kruchten, Nord and Ozkaya, 2012).

As for more formalized notion of Technical debt, it could be described as the costs that are needed to be spent to increase the technical quality level to a point where it could be considered as ideal. Additionally, technical debt has its own interest, that is the extra costs needed to maintain and unsure the reliability of the software with a poor technical quality. (Marinescu, 2012). However, this definition is not complete, because more recent studies characterize technical debt from a bit different perspective. They consider technical debt as invisible results that appeared because of the past decisions about software and that could influence the whole system in the future. Moreover, the technical debt that is managed in a company in a logical and accurate manner could bring valuable benefits and somehow can be considered as investments opportunities (Falessi et al., 2014).

Some researches see technical debt as a core invisible part of software that lies between visible parts of new features and additional functionality on the one hand and defects and bugs on the other hand (see Figure 3).

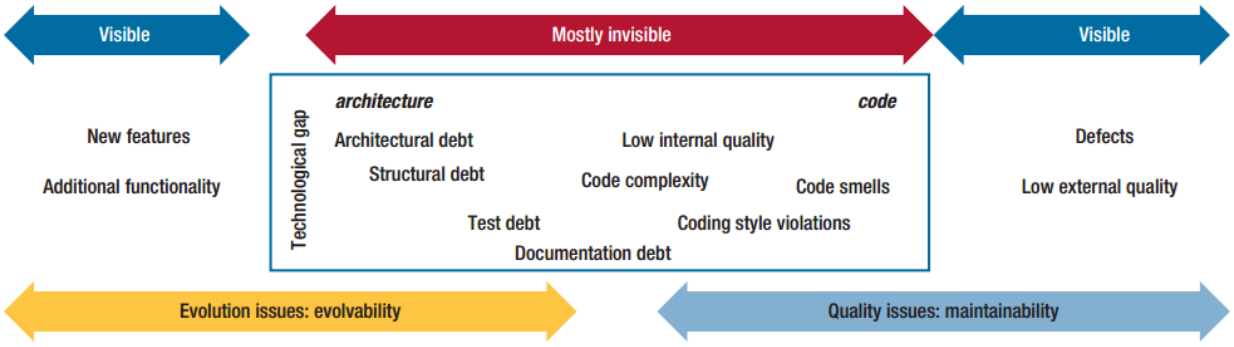

Figure 3. The technical debt landscape. On the left, evolution or its challenges; on the right, quality issues, both internal and external. (Source: Kruchten, Nord and Ozkaya, 2012)

Technical debt was proved to be not just a metaphor, but a complex concept that could be valuable for practitioners. It was proved by asking 544 participants (coding and software architects professionals) and 65% of respondents disagree that "Technical debt is just a metaphore", furthermore, 79% of respondents of the same group agreed on a statement that "Lack of awareness of Technical debt is a problem" (Ernst et al., 2015)

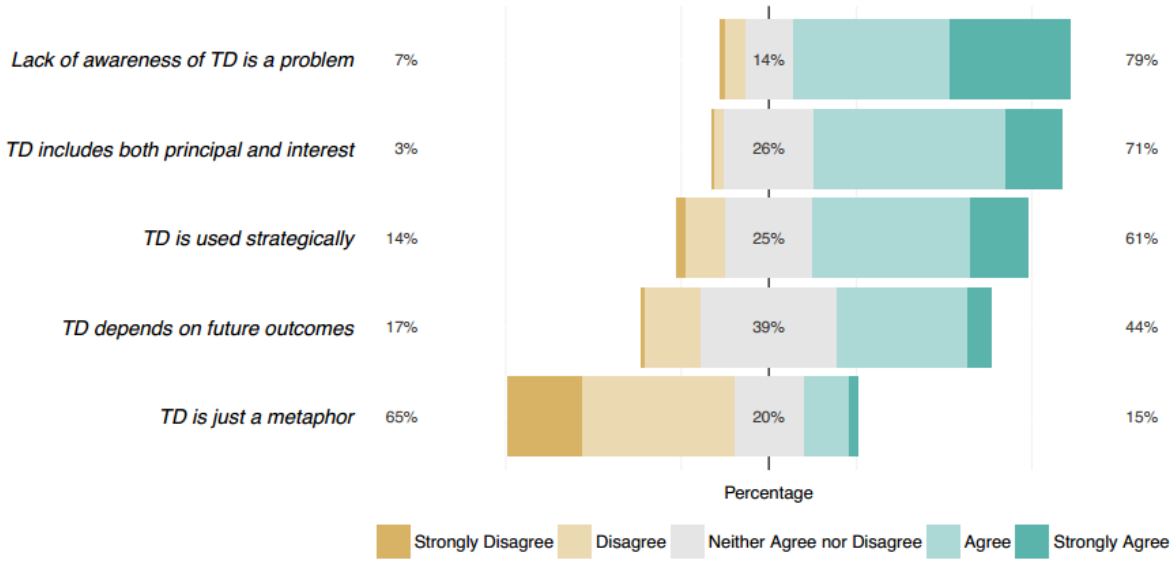

Figure 4 High-level definitions of technical debt. (Source: Ernst et al., 2015)

The concept of technical debt is not mature yet, it is still evolving, however, there are already number of studies dedicated to it. These studies cover wide range of topics, related to technical debt, such as:

1. Overall research on the topic, a systematic literature review or a systematic mapping study.
2. Investigation of causes of technical debt and its classification.

3. Core activities of technical debt management:
    a. Identification of technical debt.
    b. Methods of technical debt measurement.
    c. Practices and tools for technical debt management.
4. Research on particular type of technical debt.

1.2.Technical debt in agile software development

Technical debt is often used in a context along with Agile software development. One of the most popular feature of agile methods is to deliver working functionality quickly in resource constrains and constantly changing requirements. Indeed, this short time period may lead to the insufficient quality in software design, test coverage and non-optimized code. And all these may cause the appearance and accumulation of technical debt. Hence, it is needed to clearly identify such points of technical debt emergence and find the ways of manage it properly in agile software development (Behutiye et al., 2017).

Though first attempts to change the approach to software development in more flexible form started in the middle of 1980s, a kind of official date of agile methodology birth as a new and separate methodology is the 2001 year. In 2001, the main leaders of different agile software development approaches such as Kanban, Extreme Programming and Scrum together created the Manifesto of Agile Software development. In this Manifesto were included core principles and values that were aimed at optimization and facilitation of the software development processes. The main proncioles from this Manifesto are listed below:

1. Software that is working is more important than comprehensive documentation.
2. Focus on collaboration with customers, not on the negotiations about the contract.
3. Individual minds and people interaction is more important than formal processes and tools.
4. Ability to react quickly on changes is more important than strict plan following. (Beck et al. 2001)

To sum up the main idea of agile approach it should be said that it focuses mainly on the product features that could be delivered with existing resources (comparing to plan-focused approach that cares more about pricing and budgeting modules). The second core idea of agile approach is concentration on the people's needs, their values and their positive experience using the software. Hence, it is much more qualitative rather than quantitative approach (Schön, Thomaschewski, Escalona 2017).

The move from plan-driven approach to agile one is shown on Figure 5.

Agile approach has several advantages, such as flexibility of process of software development, moreover, it allows to avoid bureaucracy of traditional software development approach. Agile development processes are based mostly on informal interaction between software development team rather than on time-consuming planning and design. All of these allows companies deliver ready-to-use solutions in shorter periods. However, Agile approach also has some drawbacks that are tight closely with knowledge management. Through all period of system development proper maintenance of project documentation is not the highest priority of the agile team, because they rely mostly on informal collaboration among team members. This approach may lead to the loss and of important knowledge during and after system development (Ru-Zhi et al. 2005).

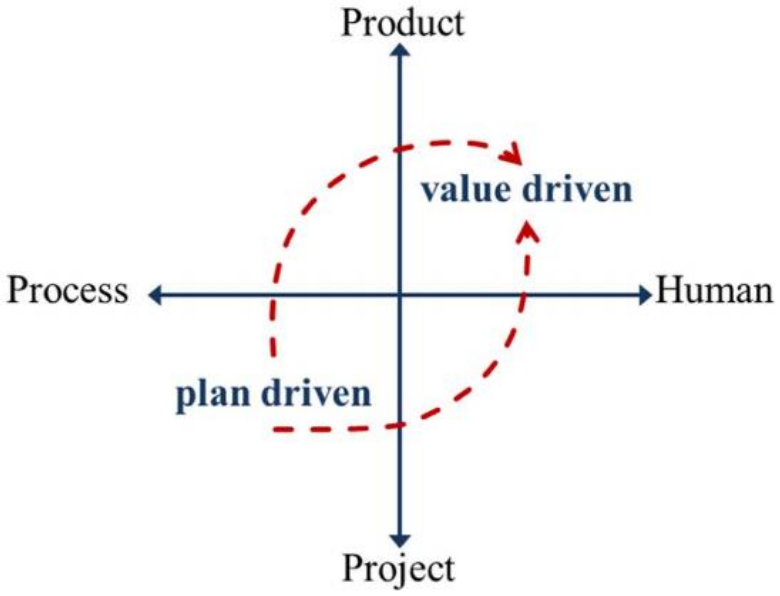

Figure 5Move from plan-driven to value-driven approach (Source: (Schön et al. 2017).

Moreover, recent years more crucial need came to the forefront. It is the need of constant updating of stored knowledge and its maintenance to remain the stored knowledge actual. The core issue there is that companies switched from Waterfall model of system development to Agile approach. As the incredible pace of changes in modern world, companies should be flexible, they should be able to adjust their strategy and their plan within changing environment conditions (these conditions could be either changes in people preference or appearance of new technologies). The main difference between Waterfall and Agile approaches is that Waterfall is reluctant to any changes in the schedule and any kind of changes should be avoided. In contrast, Agile approaches are aimed at getting the best value in frames of certain time period (Davis et al. 2014).

1.3.The context of software development process

Technical debt is highly related to the context (Fernandez-Sanchez et al. 2015), as well as software development process through which technical debt appears is also highly context-dependent (Kruchten 2011). Therefore, in order to define the context of technical debt it is necessary to identify the context of software development processes.

There several studies which identify main factors that are needed to consider in order to define software development context. Despite some of the studies define the context in order to later determine, whether the company would be able to absorb agile development methodology, their approach is also applicable for defining overall software development context. The comparison of proposed factors is shown in Table 3.

Table 3 Comparison of factors needed to determine the context of software development

	Boehm-Turner, 2003	Cockburn & Crystal, 2005	Ambler, 2009	Kruchten, 2011
Factors	• Size, • Criticality, • Personnel (their skill, know-how), • Dynamism (rate of change) and • Culture of the team: thriving on chaos or on order	• Size, • Criticality • Skills.	• Team Size • Geographical Distribution • Compliance • Organization & Culture • Organization distribution • Application complexity • Enterprise discipline • Governance	• Size • Team distribution • Criticality • Business model • Governance • Age of system • Rate of change

The model of Krutchen is described below as more recent one:

Size.

By this part the overall size of the system is implied. It is considered as one of the greatest factor, because it act as a driver for the size of the team, the number of teams, the needs for communication and coordination between teams, the impact of changes, etc.

Business model

This part relates to the money flow, and what is the main product of the company - internal system, a commercial product, contract system for a customer, or not an independenty product but instead a component of a large system involving many other parties? Is it commercial or free and open-source software?

Team distribution

This aspect is linked to the size of the project. If the team is widely distributed, a lot of attention should be put into communications and coordination of decisions. Moreover, stable interfaces between teams, and between the software components could be needed.

Rate of change

This rate implies the position of the system in modern changing environment, including business environment, business stability, unknown risks and the role of the system in this environment.

Age of system

This aspect relates to the amount of legacy code in the system as well as its architecture that could be strongly affected by the historical decisions about the system development. If considered system is quite young, it could contain less legacy code.

Are we looking at the evolution of a large legacy system, bringing in turn many hidden assumptions regarding the architecture, or the creation of a new system with fewer constraints?

Criticality

This part of the context covers the questions that relate to the consequences of the system fails and documentation that is needed to support this system.

Governance

This aspect relates to software development processes (how do they start and finish), to the person (group of people) who makes critical decisions about the system and its development in questionable or highly important moments and to the person (group of people) who manages project managers.

There are also studies that define overall context of software development, not only for agile practices. The context could be defined by these factors: Business, Architecture, Process, Organization and by the interconnection of these factors (Betz, Wohlin, 2012).

1.4. Causes of technical debt and its classification

Technical debt can be classified based on the types of the causes of this debt (Li et al., 2015). Classification, presented in Table 4.

Table 4. Types of technical debt (source: Li et al., 2015)

Technical Debt type	Explanation	Examples
1. Requirements	The difference between the real processes in the existing system and the optimal requirements that couldn't be met due to system constrains.	• Over-engineering
2. Architectural	Is caused by the decisions on system architecture level to agree on some compromises that could be crucial in the future.	• Architecture smells • Architectural anti-patterns • Violating of good architectural practices • Architectural compliance issues
3. Design	Refers to technical shortcuts in detailed design.	• Code smells • Incomplete design specifications • Grime
4. Code	Refers to the poor quality of the code (code that goes againt the coding rules od coding best practices).	• Low-quality code • Duplicate code • Code violations
5. Test	Is caused by shortcuts while testing.	• Lack of test • Lack of test automation • Residual defects not found in tests • Expensive tests
6. Build	Is about drawbacks in the system or about too complex processes in built system.	• Bad dependences • Manual build processes • Flawed automatic building
7. Documentation	Is caused by incomplete or outdated documentation in system description (when the current state of the system could be found only in code)	• Outdated documentation • Insufficient documentation • Lack of code comments
8. Infrastructure	Refers to negative impact of infrastructure on the team (when processes, technologies and supporting tools are not optimal).	• Lack of continuous integration • Old technology in use

		• Lack of automated deployment
9. Versioning	Is caused by inaccurate code versioning.	• Multi-versioning support • Code forks
10. Defect	Is found in system bugs and failures.	• Bugs • Defects

Martini et al. in their work "Architecture Technical Debt: Understanding Causes and a Qualitative Model" in 2014 investigate and classify the most frequent causes for accumulation of architecture technical debt, however, their classification is highly compatible with the overall causes of technical debt, not only architecture one, see Figure 6.

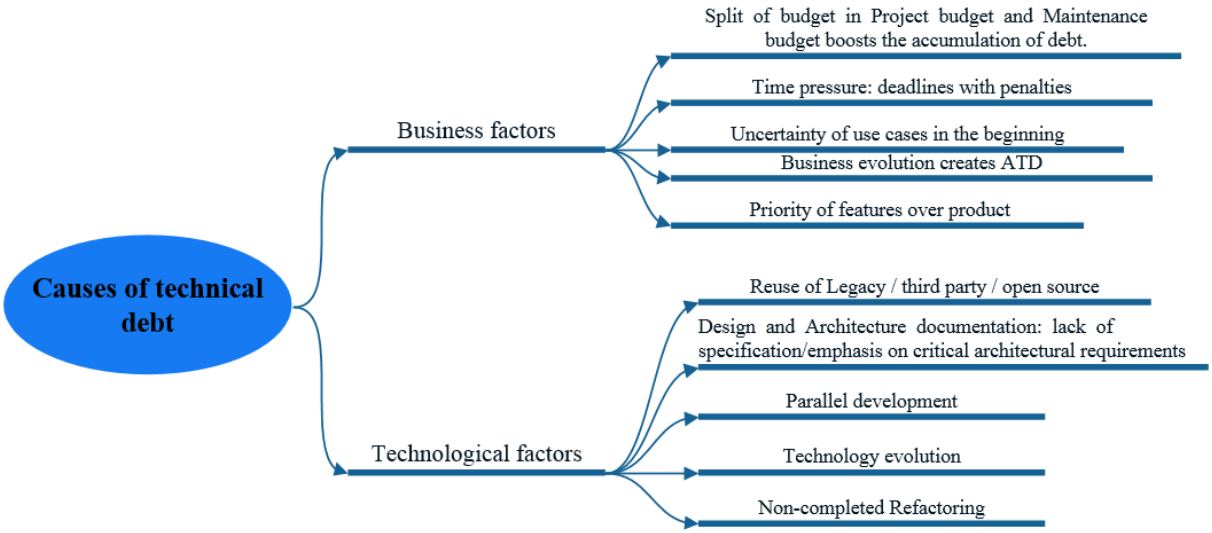

Figure 6 Causes of ATD accumulation (source: Martini et al., 2014)

Li et al. based types of technical debt on the causes of these types, but the causes in their classification shows only by the technical side of the question. Martini et al. have more broad causes: which included not only technical constrains but also business factors and human factor. The point is that several causes may influence particular type of technical debt and one cause may influence several types of technical debt and the proportion of the influence may vary. Hence, it may be needed to build more clear interconnections between the causes of technical debt and the types of debt that may appear.

As for causes of technical debt in agile software development, eight main causes can be pointed out, see Figure 7 (Behutiye et al. 2017)

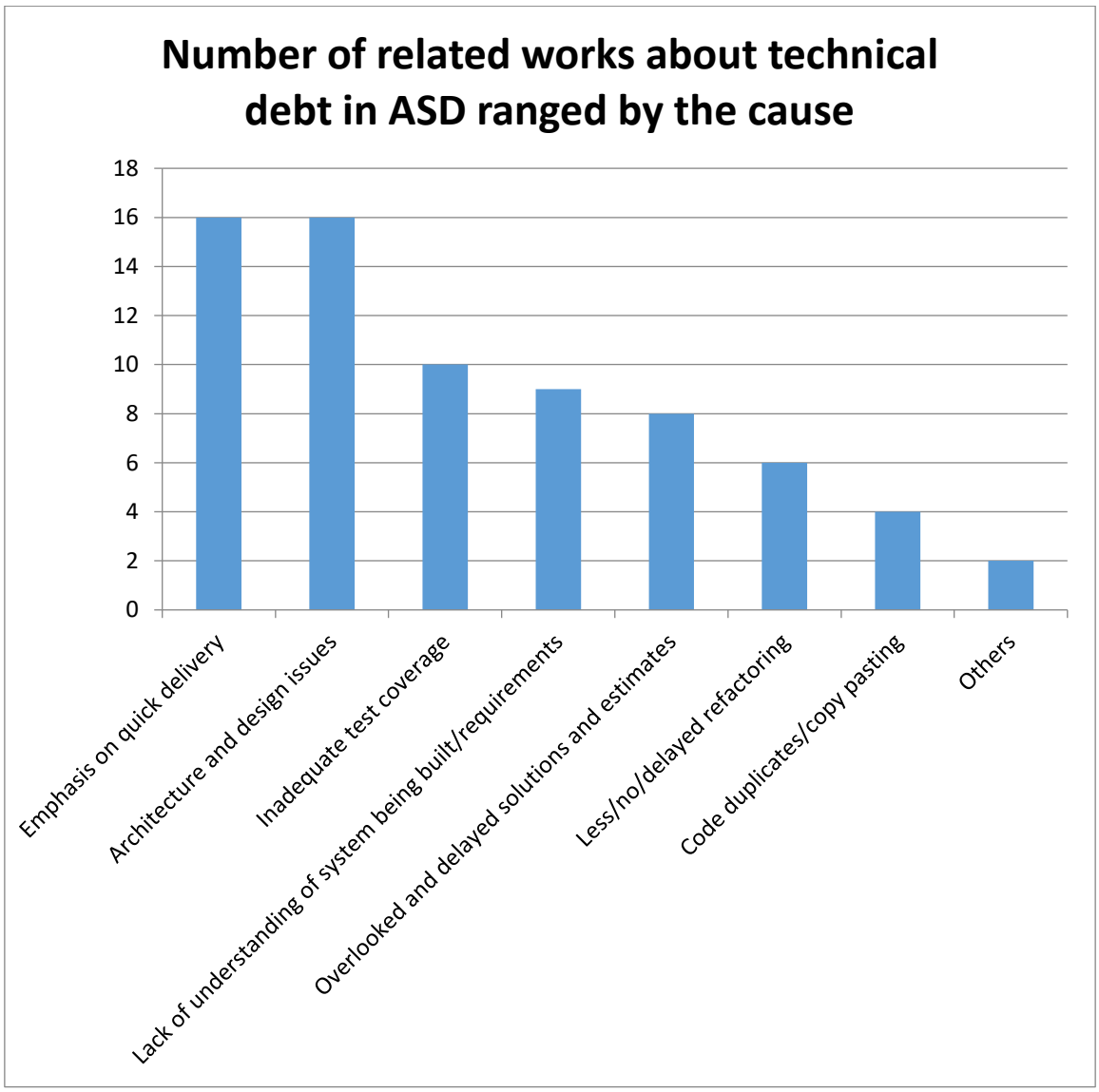

Figure 7. Number of related works about technical debt in ASD ranged by the cause (source: Behutiye et al. 2017)

The most common consequences of incurring technical debt in agile software development were also identified. The consequences are the following:

- Reduced productivity (in 17 papers)
- System quality degradation (in 17 papers)
- Increased cost of maintenance (in 15 papers)
- Complete redesign or rework of system (in 3 papers)
- Market loss/ hurt business relationships (in 3 papers) (Behutiye et al. 2017)

Although the classification of technical debt consequences is useful, it is needed to be linked with the causes and types of technical debt as well as with the management practices to avoid them.

1.5.Technical debt management

Technical debt studies claim that technical debt could be taken on purpose to have a quick win in a short-term. For example, release of new product feature prior to competitors may help the company beat the competitor. However, existing and occurring technical debt should be identified, measured and managed in a proper way. Technical debt is needed to be tracked and kept visible because without proper management, technical debt accumulates and may create a lot of challenges and problems in system maintenance and further development (Li et al., 2015).

One of the key issues in technical debt management is the difference in indicating and measuring different types of technical debts. Modern tools and techniques are mostly concentrated on code quality analysis, this code evaluating methods are technical and can be measured with quantity. In contrast, the existence of architecture debt, requirements debt, etc. challenges the way of technical debt measure (Ernst et al. 2014). Hence, technical debt couldn't be considered only in frame of the code. It is a multidimensional problem, that could be solved with complex approach, that includes and requires analysis of software evolution, qualitative research on a context program analysis, software metrics and risk management (Shull et al., 2013).

Moreover, as researches states, very often technical debt is managed in implicit way – by the project manager's previous experience or even driven only by his or her instinct. In such cases, critical information about technical debt, such as its location, amount, possible risks is hidden for other stakeholders and, therefore, there is a high possibility, especially for large software projects to lose controle over the project and over the system as a whole (Seaman et al., 2011).

Costs of managing technical debt

Systematic literature review conducted by Li et al. discovered eight different activities of technical debt management, for each activity several approaches were found. Indicated activities are presented on

Technical debt repayment helps to diminish and ease known technical debt. The most popular and frequently used repayment approach is refactoring – a process by which internal code quality or system architecture could be improved without changing external system behavior. Such approaches as rewriting – rewriting the code with technical debt, automation – make automatic previously manual work (deployment, tests, etc.) and reengineering – change not only code, but also external features or operational quality of the system. The last three approaches are rarely presented in the academic studies, they are repackaging – group connected modules with dependencies that are convenient to manage in order to make the codebase simpler, bug fixing – solve existing bugs in the system and fault tolerance – set runtime exceptions on purpose.

For identification of technical debt, source code analysis approach could be used, where emphasis should be put on such issues as coding rules violation, flaws in design or architecture and lack of tests. Another approach is to analyze dependences between modules or components of the software. Approaches that are listed as approaches with minimum mentioning in research studies are: check list of scenarios that were predefined and comparison of actual solution with an optimal solutions in some dimensions.

Technical debt measurement activity implies quantification of costs and benefits caused by technical debt through special estimation techniques, by measurement, the overall level of technical debt in a system also could be estimated. The most frequently used approaches for measurement technical debt are calculation model which uses mathematical models and formulas, code metrics that also uses sources of code and human estimation which refers to experts in the field of programming who based on their experience and knowledge are able to give quantitative measure for technical debt.

Technical debt monitoring watches the changes of the cost and benefit of unresolved TD over time.

Technical debt prioritization ranks identified TD according to certain predefined rules to support deciding which TD items should be repaid first and which TD items can be tolerated until later releases

Technical debt communication makes identified TD visible to stakeholders so that it can be discussed and further managed by different stakeholders in the company as well as outside of the company.

Technical debt prevention aims to prevent potential TD from being incurred. Prevention methods include such methods as development processes improvement, architecture decision-making support, lifecycle cost planning, and human factors analysis.

Technical debt representation/documentation provides a way to represent and codify TD in a uniform manner addressing the concerns of particular stakeholders. The research conducted by Li et al. points out that technical debt representation methods still do not have common understanding by in research areas.

The concept map of technical debt management activities is shown on Figure 8.

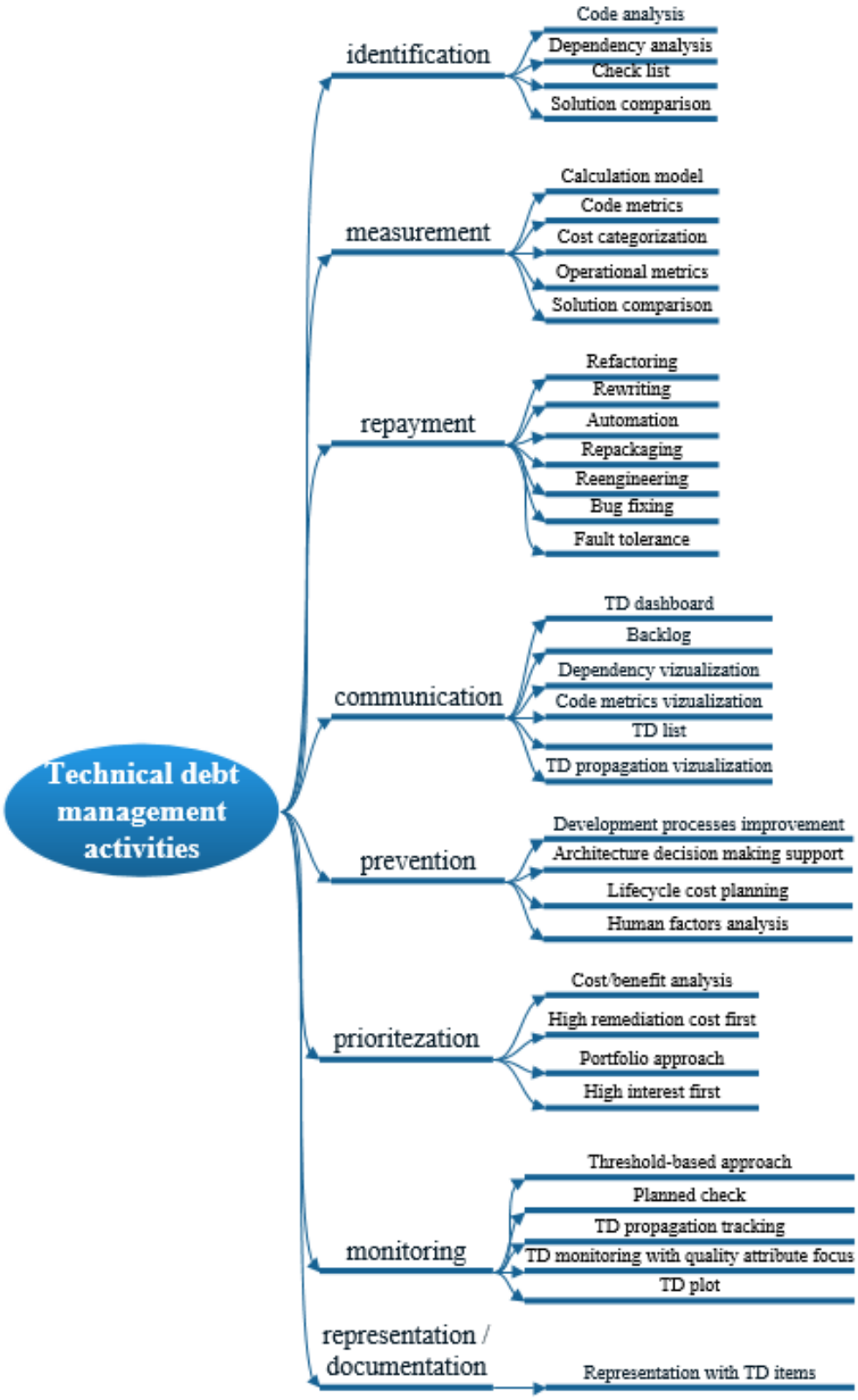

Figure 8 Technical debt management activities (compound from Li et al. 2015)

Technical Debt Management Framework

Technical debt could be grouped by different elements that include core elements, implementation elements and management elements, this grouping was obtained by conduction of

systematic mapping study (Fernandez-Sanchez et al. 2015). Figure 9 shows Framework for the Elements for Technical Debt Management.

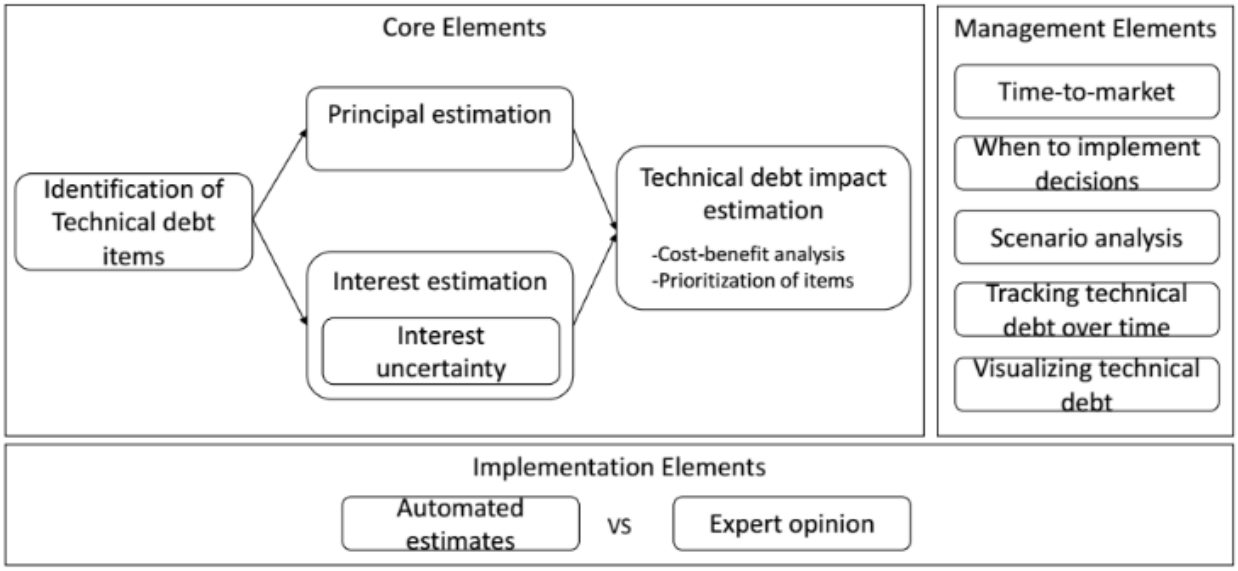

Figure 9. Framework for the Elements for Technical Debt Management (source: Fernandez-Sanchez et al. 2015)

The detailed description of the Framework is presented in Table 5.

Table 5. Description of the Elements of the Framework (source: Fernandez-Sanchez et al. 2015)

Core elements:	
Identification of technical debt items.	Technical debt identification focuses on two main types of technical debt: code and architecture. To identify code debt different methods based on lines of code and dynamic and static analysis of code deficits are used. As for architectural debt, such methods as modularity violation detection and rare class analysis are used.
Principal estimation	There were detected two main ways to estimate technical debt principal. The first way is based on repository of previous projects, where similar ones may help to estimate the principal. The second way is to estimate items of technical debt and then apply typical estimation of the organization.
Interest estimation	For the interest estimation, it is possible to use information from previous projects with the same technology. Another way is to estimate the difference between cost-per-change and cost-per-defect. Interest Uncertainty Estimation:

	There were found several propositions to estimate the uncertainty of the interest by the probability assignment, however, concrete methods of estimation were not provided.
Technical Debt Impact Estimation	This element is concentrated on analysis of economic consequences caused by technical debt. However, proposed methods for this estimation have not considered technical debt accumulation in concrete modules or components in the system, but rather describe the consequences for the system as a whole. Other studies provided methods based on cost-benefit analysis, comparing effects from incurring technical debt or developing new feature. Furthermore, several studies include time dimension into analysis and propose to evaluate technical debt evolution over the time.
Implementation elements:	
Automated Estimates	For this type of estimation, there are also two different approaches. The first is based on the historical repository of the previous approaches. The second one is based on such resources as code base or control version system.
Expert opinion	The studies point out the need of expert opinion in case of estimations which cannot be estimated in another way
Management elements	
Scenario analysis	There are several different types of scenarios that could be used: technical debt goals analysis and estimation of the efforts to achieve these goals, release analysis to find the most profitable release from the point of architectural debt view.
Time-to-market	Studies are very limited in provision of explicit methods for time-to-market decisions about technical debt
When-to-implement decisions	Several studies report portfolio method or real option method for evaluating when to implement decision in release. When to implement secession refers to the decision whether it is necessary refactor now or it is needed to release new feature.
Tracking technical debt over time.	A lot of articles propose to look at the historical data in order to estimate the interest of technical debt. However, the studies are highly limited when it comes to tracking technical debt evolution over time.

Visualizing technical debt	There were found several methods that are used for visualize technical debt. One way is to create charts that show relationships among interest, principle and time. Another way is to show different type of relations among software modules or components. However, these studies are limited.

1.6.Technical debt management strategies

Alves et al. in the research point out the technical debt management strategies that were found more than in two papers, those strategies are:

- Portfolio Approach.

  The central concept of this strategy is to list TD items. This list contains debt items identified for the project. Each TD item in the list should contain the registration information, such as the location of the debt, the time at which it is identified, the responsible person, the reason why it is considered TD, an estimation of the principal, as well as estimation of the interest and also the estimation of the correlations of this item with other TD items. After conducting the list the analysis should be done in order to identify, which items should be paid off first and for which items the repayment could wait.
- Cost-Benefit Analysis.

  This type of analysis is used to evaluate whether the repayment oof technical debt is justified by the high cost of the interest. It should be pointed out that the interest rate is composed of two parts: the probability of interest and its value. The first part refers to the probability that the debt, if not paid, will result in extra cost to the project. The second part is an estimated amount of additional work that will be required if this item is not paid.
- Analytic Hierarchy Process.

  In AHP, the problem is structured by running a comparison of alternatives that are compared with the help of specific criteria. For each alternative the overall ranking is determined. The usage of AHP in technical debt management implies the identification of technical debt and the outcome of this method is a prioritized list of technical debt items with identification of the most crucial technical debt items for paying off..
- Calculation of technical debt Principal.

  The strategy is focused on the estimation of the principal. The principal is estimated and associated with quality attributes, which helps the managers to “feel” these technical debt items better and with this feeling to make better decisions.
- Marking of dependencies and Code Issues

This strategy is used to manage problems and dependencies in the project source code. by conducting these dependences, the special tags in the code are inserted in order to ease for the developers the visibility of technical debt items and to support their decisions about when and how to pat off technical debt.

Behutiye et al. in their research provides a different view on technical debt strategies classification. The strategies are:

1. Specific approaches, tools and models to manage TD in ASD
2. Refactoring
3. Enhanced visibility of TD
4. Test automation
5. Common (agreed) DoD
6. Planning in advance for TD
7. Code analysis
8. Agile practices such as pair programming, TDD (test driven development) and CI (continuous integration)
9. Prioritizing
10. Improving estimation techniques
11. Transparent communication as to the level of TD with business stakeholders
12. Establishing an acceptable level of TD

1.7.Examples of empirical studies of technical debt

Table 6 Literature review of empirical studies (compound by the author)

Author	Type of study	Conclusions	Comment
Zazworka et al, 2013	Single case-study. Brazilian company	The tools used are especially useful for identifying defect debt but cannot help in identifying many other types of debt, so involving humans in the identification process is necessary.	Single company case, Concentrated on identification methods and tools, lacks of context about software development process.
Klinger et al., 2011	Single company, 4	Decisions related to TD issues were often informal and ad hoc,	A case of single company, the study can be quite outdated,

	interviews at IBM (USA)	Which led to a lack of tracking and quantifying the decisions and issues. The study also identified that there was a large communication gap between technical and business people as regards discussion about TD.	lack of view from organizational perspective.
Guo et al., 2016	Single case-study, Brazil	Goal of this study was to uncover the costs of explicit TD management. Through data analysis, were identified three major themes regarding TD management – costs of, obstacles to applying explicit TD management to the project, and deviation of the actual TD management process from the proposed one.	Describes only one project from the very beginning to the end.
Yli-Huumo et al., 2016	Single company, several teams, Finland	The goal was to identify technical debt management activities in different teams and generalize them by the level of maturity.	Generalization of the results, lack of organizational view perspective.
Falessi, Voegele, 2015	Single case-study, quantitative-qualitative analysis	The aim is to explore the interest associated with violating quality rules.	Technical paper, lack of the context of system development and organizational view.
Yli-Huumo et al., 2017	Single company case, Finland	The aim was to find and identify processes for technical debt identification, documentation and prioritization in order to increase its manageability and visibility.	Covers only several technical debt management activities, lack of organizational view.

The framework of technical debt activities maturity levels developed by by Yli-Huumo et al. is shown in Table 7.

Table 7 Technical debt management framework (source: Yli-Huumo et al., 2016)

TDM activity/ TDM levels	TD repayment	TD prevention	TD representation/ documentation	TD identification	TD measurement	TD monitoring	TD communication	TD prioritization
Organized (Level 3)	Continuous repayment with monthly assigned percentage of the development tasks.	Mandatory prevention practices used by the team. Continuous practice during development.	Documentation is a mandatory practice in development. Issues are documented in a separate TD backlog.	Continuous identification conducted manually and/or with tools during development.	Continuous measurement during development. Data analysis (various data used (e.g. quality. performance)). Assisted with trials	Continuous monitoring during development with various data (e.g. quality, performance). Tools used to support.	Continuous discus-sions/meetings about TD issues with all the necessary stakeholders involved.	Prioritization conducted continuously during development. Prioritization follows a specific method or model.
Received (Level 2)	Repayment during normal development tasks and previously identified repayment tasks. Repayment conducted based on current needs.	Optional prevention practices. Not mandatory to use, but recommended. Conducted based on current time constraints.	Documentation an optional practice, but recommended. Issues documented in a general development backlog without TD id.	Identification optional during normal development. Conducted based on current time constraints.	Measurement an optional practice. Measurement done with simple data (number of TD issues) from development. and the data not necessarily used for other activities.	Monitoring based on simple data (number of TD issues). Conducted occasionally.	Discussions/ meetings organized only with some stakeholders.	Prioritization based on hunches and rough estimations based on previous experiences. Prioritization done in a simple way without any specific model.

Unorganized (Level 1)	Repayment not conducted at all or only when it is not possible to avoid the issue any longer.	Prevention not assigned as part of the development practices. Conducted only occasionally.	Documentation not part of development. Issues are left in developers' own minds and notes.	Identification practices not assigned as part of development. Conducted only when issues occur.	Measurement not part of development practices.	Monitoring not part of development practices.	TD not a topic in discus-sions/meetings and often handled only in coffee table discussions.	Prioritization not conducted, and decisions done without reasoning or discussions.
Responsibility for activity	Development team, software architect(s)	Development team, software architect(s)	Development team, software architect(s)	Development team, software architect(s)	Software architect(s), team manager	Software architect(s), team manager	Development team, software architect(s), team manager	Software architect(s), team manager
Practices / tools for activity	Refactoring. redesigning, rewriting	Coding standards, code reviews. Definition of Done.	Technical debt backlog/list, Documentation practice, project management tool (/IRA. Wiki)	Time reservation for manual code inspection. Use of code analysis tools (SonarQube.	Data from measurement tools (SonarQube) and data from project	Monitoring tools (SonarQube). Project management tools (jIRA. Wiki)	Specific TD meetings, TD included in discussion topics.	Cost/Benefit model. Issue rating

1.8.Conclusions and research gap identification

The conclusion about current situation on technical debt research can be formulated in a such way:

- Despite the description of different types of technical debt, the strategies and management practices in majority does not linked with these types, this link is needed to find effective technical debt tracking activities. Moreover, while investigating a technical debt, it is always needed to look at the big picture and avoid focusing only on details.
- There are limited studies on influencing of the system software visualization on technical debt and its management.
- Technical debt management strategies are not fully investigated and understood. Many of the proposed strategies need further and deeper investigation as well as more clear classification
- The studies in TD are quite recent, and the subject is not mature (Martini et al., 2015)
- In current technical debt research the focus on particular types of technical debt is noticeable (architecture, design, code and defect). However, the concept of technical debt implies the importance of other types of technical debt and their further investigation. (Ernst et al., 2015, Alves et al., 2016).
- Most of the empirical studies of TDM take in consideration only few aspects of the eight TDM activities (Li et al., 2015).

The concept of technical debt has wide range of research areas that are to research opened from academic perspective. The following areas may introduce the possible direction of further research on the topic of technical debt:

1. Investigation of the ways for technical debt management.
2. Tools for tracking technical debt.
3. Models for technical debt evaluation.
4. Examination of relationships between the causes and the consequences of technical debt.
5. Strategies of repaying the technical debt. (Li et al., 2015, Alves et al., 2016, Behutiye et al. 2017).

1.9.Theoretical model of the study

The concept of technical debt was studied by the number of scholars from different perspectives. Previous studies have reported technical debt classification in terms of causes, types, identification tools, measurement techniques, consequences and management strategies. Visualized concept of technical debt is presented on the Figure 6.

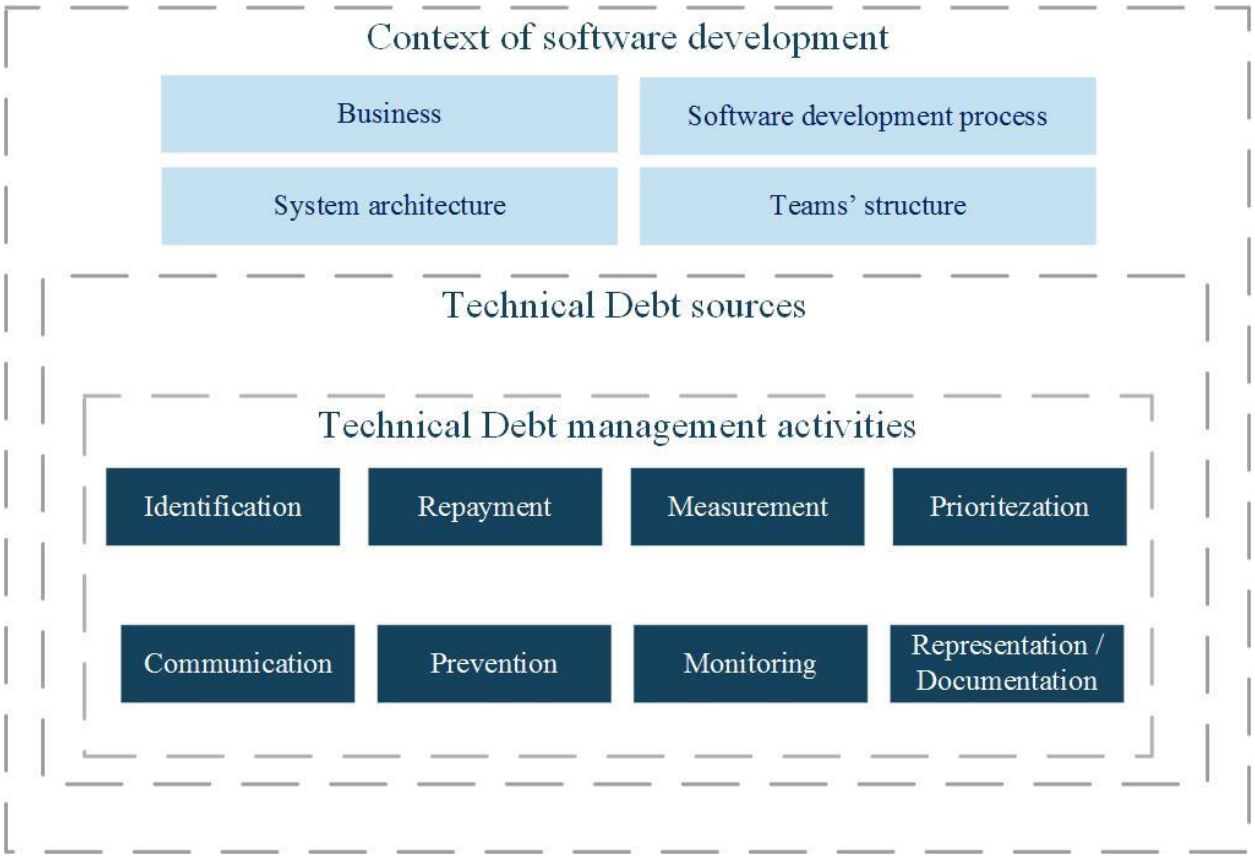

Figure 10 Theoretical model of the study (compound by the author using sources: Kruchten, 2011, Betz, Wohlin, 2012, Li et al., 2015)

2. CHAPTER 2: RESEARCH METHODOLOGY

2.1. Research questions

In this chapter the research methodology will be introduced, it includes the research design, approach of the study, methods of the data collection and, finally, possible limitations.

The literature review in the previous chapter clearly shown a research gap in the field of technical debt studies. This gap occurs when technical debt management practices meet the complex of system architecture, organizational design and development methodology. Finally, the research on technical debt in Russian software development companies is also highly limited. Therefore, in order to investigate the topic more deeply the following research questions were asked:

RQ1. How do software development process influence the sources of technical debt in Russian software development companies?

RQ2. How does the context influence technical debt management in Russian software development companies?

RQ3. What technical debt management activities could be considered as mature in Russian software development companies? What methods are used to support these activities?

RQ4 What factors should be considered during decision-making processes about managing technical debt?

2.2.Research methodoogy

This research consists of several parts: The first one is theoretical and is represented by literature review. This theoretical background is necessary to provide a strong fundamental basis for further research. The literature review helped to identify core causes and types of technical debt as well as modern methods and tools to manage technical debt. By conducting the literature review the research gap was found and research questions were determined.

There are two main types of research that is recognized by researches: quantitative analysis and qualitative analysis. The difference between these two types lies on the type of data used for the research. Quantitative research underlines quantification in the analysis of data, while qualitative research emphasizes words. Moreover, in the base of qualitative research is an inductive approach that analyze the relationships between theory and research (Bryman and Bell, 2003).

For the analysis of technical debt management practices in Russian software companies using agile the qualitative research was chosen. The main reason of that lies in the findings of the first chapter. Technical debt management is a complex concept that includes a lot of data many of which is very hard to evaluate quantitatively.

This study is qualitative, and it uses case study as the research methodology. A case study is an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident. (Yin, 2003). A case study that is used as a research strategy could contribute to the knowledge of individual or group. .Despite being highly useful for economic research, case study is becoming more and more popular approach to make a research in the field of software development. And taking into account the fact the software is developed by individuals, groups and organizations and impose social context, a case study could be considered as a relevant approach (Runeson and Hest. 2008).

Technical debt could be studied by analysis of code sources, and further special analysis of code quality. Hovewer, theis study has the aim ton investigate technical debt from the organizational point of view and should be performed with qualitative methods.

The empirical study stands for the second part of this research. The second part would be practical and would be aimed at investigate the technical debt management practices in Russian software development companies.

The following methods for the second part of the research were chosen:

- Interviews.
- Documentation analysis and participant observation for one of the company.

Both of them are targeted on getting a deep understanding of current technical debt management situation in companies as well as their attitude towards this topic form inside. By using these methods it is planned to run an exploratory qualitative research and as a result to present a multiple case study of technical debt management practices of Russian software development companies that use agile. This multiple case study would have a comparison of two types of such companies: the ones, who focus on B2B and the others, whose focus is B2C.

In order to meet the reliability requirements, the one operating market was chosen – the Russian market. Moreover, due to the fact that technical debt is closely tight with software development processes, the companies chosen for the research should use agile development methodology.

2.3.Data Collection and research design

To run high quality qualitative research, it is needed to have a deep investigation about the market and chosen companies. The analysis of current situation of technical debt of Russian software development companies should be conducted with the help of data from news and IT-journals. After understanding the market, it is needed to understand business models of the chosen companies and what kind of system lies in their core business. These types of analysis are called the desk research. While doing the desk research it is also needed to pay attention to the identified in the first chapter models and tools which might be helpful in further, field research.

After completion of the desk research, the field research should be started. This research would include indepth interviews with the companies' representatives. It would be needed to interview a number of different people with different positions from each company, for example: developers, projects managers, IT and infrastructure architects. It is possible that in some companies the level of awareness about technical debt would be higher and in other companies this level might be very law. Hence, it is needed to be prepared to adjust conducting interview with these different levels of awareness.

The research design is shown on Figure 11

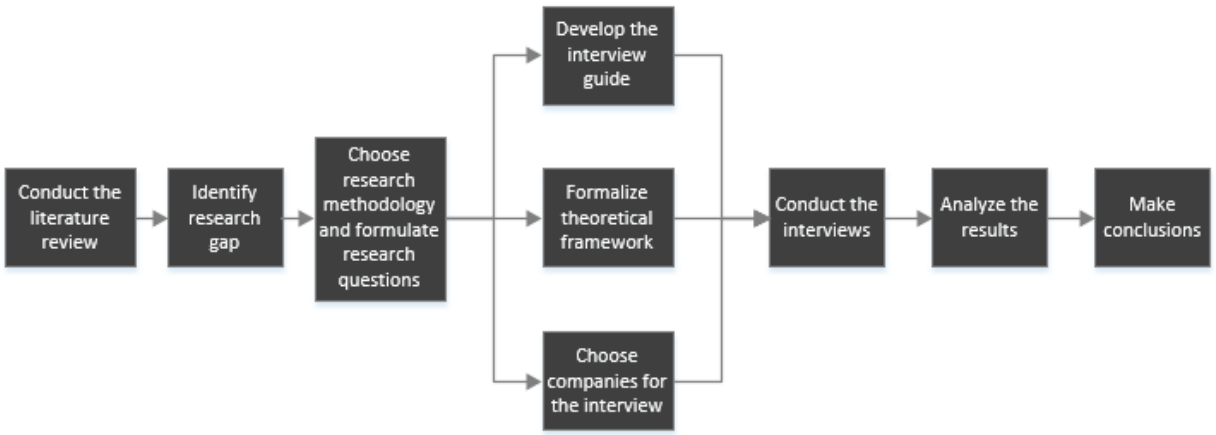

Figure 11 Research design

2.4.Choice of the companies for the research

The choice of the companies for the analysis was built on the several criteria:

1. The industry.

Despite the variety of companies, which could be considered as IT-companies, it was important to define precisely software development companies and not the software implementation companies or hardware producers.

2. The age of the company.

Technical debt management practices may vary greatly for young start-up companies and for mature companies which were operating on the market for several years. For the research were chosen that companies that were operating on the market at least 10 years.

3. The market.

As it was mentioned in the previous chapter, one of the research questions is to investigate the differences between technical debt management practices for companies with B2C and B2B (or b2g) market in order to identify, how external business-client may influence technical debt management in a company.

All three companies that were chosen for the analysis have expressed their willingness to remain undisclosed, so in this research they would be named as “Company A”, “Company B” and “Company C”. This fact could have indirect positive influence on the interview results, because companies’ representatives, being sure to remain unclosed, could be more honest answering questions related to obstacles and difficulties in technical debt management inside the company.

For the research five interviews with different people were conducted. Each interview lasted from 1,5 hours to 2 hours and took 9 hours in total. Interviewees and their positions in each company are presented in Table 8.

Table 8 Interviewees and their experience

Position	Company	Years in IT	Years in the company
System architect	A	14	9
Project Manager	A	5	2
Head of channel solutions	B	20	2
Architect/ team lead	B	13	12
Project Manager	C	5	2

3. CHAPTER 3: CASE STUDY ANALYSIS

In this chapter the description of the companies for case study would be presented as well as the choice of the companies would be justified. For each case, detailed case description would be provided as well as cross-case study analysis would be given with further conclusions, implications and limitations.

3.1.Company A case study

Company A is a fin-tech company which operates on the market about 15 years. Despite having both – individual users and companies as clients, the company has internal product owners, who facilitate and drive product development according to the main company strategy. The main products of the company are aimed at satisfying desires on both B2B and B2C markets. Due to the fact that company A provides services, instead of final product as well as its B2B clients are "mass market" – small and medium enterprises which do not require customize solution and pay for services, company A could be considered as a company with B2C market. The company is quite big, it has more than 600 employees with several offices in different Russian cities. It operates primarily on the Russian market but has a pool of foreign clients.

System architecture

The system of company A has a service-oriented architecture (SOA). This type of architecture implies several components that could act as clients as well as services for other components (modules) in the system. The components are linked through a communication protocol over a network. SOA architecture has the main basic principle which lies on the independence of products, vendors and technologies. In SOA architecture service is a functional unit with independent update and remote access.

For SOA there are four main points about the service:

- It has defined outcome and particular business logic.
- It is closed element.
- For consumers, it should be a black box.
- It may group other services. (Welke et al. 2010).

Several years ago the strategic decision by the top-management was made and a course on microservices architecture was taken. That meant important changes in particular services in the system in terms of separation of system modules (components) into several, more independent components. Moreover, in order to support these planned changes, the company also revised current system components in order to identify those, which would require separation.

Organizational design and teams' structure

There are fourteen different teams in company A, and each team is responsible for particular product of the company (particular service development). Each team has project manager, one or more product owner (if team is responsible for several services), one or more front-end developer, one or more backend developer, one or more quality assurance engineer. Some teams have an analyst as a team member, but more often one analyst could be assigned for a project of different teams. Looking only at departments, which have direct impact on the system (not including commercial, accounting, marketing and other departments), the company has different departments for such positions as front-end developers, back-end developers, quality assurance engineers, projects managers and analytics. Moreover, the company has positions of business architect and system architect who are responsible for approval of solutions.

Software development process

For the purpose of investigation technical debt management practices in the company it is needed to study the processes of product development. Each team has several projects in quarter plan which should be formally approved, however, plans could be reconciled. There are planned short sprints inside each project and also development process is regulated by agreement processes in all stages of development. The company uses agile development methodology called SCRUM but with several adjustments in accordance with accumulated natural processes in the company. The steps of development process are presented below:

1. Formulation of the idea / request.
2. Verification of the ides / request.
3. Formulation of upper-level requirements.
4. Analysis, preparation of detailed technical solution.
5. Solution agreement with architects.
6. Product / new feature development.
7. Code review.
8. Testing.
9. Bugs correction.
10. Release.

It should be mentioned that as the company uses agile methodology, for each step from 5 to 9 can be repeated for each project, moreover, it is possible that product owner decides to add new requirements and therefore some changes would appear.

Technical debt causes

As it was mentioned above, technical debt causes could be operational and strategic, these types of causes could occur because of business or technological factors. The causes of technical debt for company A are presented in

Table 9.

Table 9 Causes of technical debt in company A

Operational	Strategic
• the pressure of time limits for the development of a new functional; • insufficient coverage of the code by tests (due to lack of time or money resources); • insufficient competence of some developers; • changing the requirements by the product owner in the course of the project implementation - insufficient funds for a full analysis and testing.	• needed changes in the architecture; • a way of deliver new feature more quickly; • the "legacy" of an existing system - it's hard to write beautiful code quickly, because everything is strongly tied to the current working processes; • technology evolution and retirement of particular technologies.

The overall development process with possible appearance of technical debt and its causes along with stakeholders communication during the process are presented on Figure 12.

Technical debt management activities

Identification

Identification of technical debt could appear in several processes. First, when new feature is developing, system analyst, discussing together with the developer future process may come across a technical debt. Another way to identify technical debt is to look through the code manually or with the help of special tools to identify code violations. However, it is necessary to point out that in these cases, developers usually know, where to look for this technical debt, because they feel and remember the parts of code where "*it was painful to develop new feature*". Moreover, due to historically development of the system, there are several components (modules) in the system, which are the core components and have the largest number code lines, hence, it is common that these components contain technical debt.

Measurement

When technical debt was identified, developers estimate, how much time it may be needed to pay off this debt. There is no automated estimations, developers give their evaluation based on expertise and previous experience. The cost of paying off technical debt is estimated in human-

weeks (human-days) and in order to translate this value into the money, it is needed to multiply it by the price of developer work.

Company A

	Development process	Communication with	Type of technical debt possible apperance	Causes of possible technical debt
Product Owner / Business	Formulation of request / idea	-	-	-
Business-architect	Verification of the idea / request	Product owner Business	Architectural debt	New requirements that are hard to develop with the current system conditions
Product owner	Formulation of the upper-level requirements	-	Architectural debt	New requirements that are hard to develop with the current system conditions
System Analyst	Technical solution creation	Developers Architects	Architectural debt Design debt Infrastructure debt	Improper solution decision Improper input data
Architect comity	Technical solution approval	System analyst	Architectural debt Design debt	Approval of improper decision
Developers	Development	System analyst Product owner	Code debt Defect debt	Development with code rules violation
Assigned group of developers	Code review	Developers	Code debt Defect debt	Approval of non-optimal code
QA engineers	Testing	Developers	Test debt	Lack of automated tests Not fully coverage of the code by tests
Developers	Bugs fixing	QA engineers Product owner	Code debt Defect debt	Not all bugs were fixed
Operation department	Release	Product owner Developers	Infrastructure debt Versioning debt Documentation debt	Unforeseen load on the system Deployment problems The difference between technical solution and realization

Figure 12 Company A possible technical debt appearance through development process.

Technical debt management activities

Identification

Identification of technical debt could appear in several processes. First, when new feature is developing, system analyst, discussing together with the developer future process may come across a technical debt. Another way to identify technical debt is to look through the code manually or with the help of special tools to identify code violations. However, it is necessary to point out that in these cases, developers usually know, where to look for this technical debt, because they feel and remember the parts of code where "*it was painful to develop new feature*". Moreover, due to historically development of the system, there are several components (modules) in the system, which are the core components and have the largest number code lines, hence, it is common that these components contain technical debt.

Measurement

When technical debt was identified, developers estimate, how much time it may be needed to pay off this debt. There is no automated estimations, developers give their evaluation based on expertise and previous experience. The cost of paying off technical debt is estimated in human-weeks (human-days) and in order to translate this value into the money, it is needed to multiply it by the price of developer work.

Repayment

The process of repayment of technical debt is divided into two main directions: strategic and operational. Strategic repayment is related to overall vision of the system by CIO, and these strategic tasks are done by special "refactoring". This team consists of 4 front-end and 5 back-end developers with one project manager and one product owner who has the position of system architect and more than 10 years of development experience. This strategic team doesn't develop new features, instead, they refactor the code, to make it more flexible and convenient for future development.

Operational tasks appear when some minor tasks appear during new feature development processes. These tasks could be done by particular team itself and these tasks are put into teams; backlog tasks. In company A backlog task is defined by the task that could be done in less than a week.

Communication

Is supported by company meetings in order to ensure the common understanding of current technical debt situation and its further management activities. For operational level it is needed to build a common idea with product owner in order to explain him/her what consequences for the business could be.

Prevention

By approving by architectural comity of new solutions, by test coverage, require code review, by setting the culture of high-standards programming (along with seniors development in refactoring team there are several junior developers, who accumulating best coding practices).

Prioritization

By running cost-benefit analysis, by expert opinion, by communication with product owner

Monitoring

By checking the readiness of set tasks, by covering code by tests.

Representation / documentation

Detailed description of the components in a system (including visualized processes), description of the desired functions of the components, written plan of actions (what should be changed and where). Also by other teams' backlog tasks.

3.2.Company B case study

Company C was launched in 1996 and more than twenty years shows stable positive results. The company works on B2B market and has banks as business clients (external product owners). Company C represents innovative technological solutions for automating payment services based on cards. Currently the company has more than 500 employees. The core company business is built on payment services provision, which includes the following:

- a wide range of operations on payment cards, from the issuance of bank cards to the provision of banking services at all stages,
- the processes of routing monetary transactions,
- operations related to mobile wallets, prepaid and fuel cards,
- management of remote banking services (RB) channels,
- management of loyalty programs, electronic and mobile commerce platforms,

Besides core business of services for card payments provision, Company C also has a direction of channel solutions - internet and mobile banking. This direction is tight closely with the core payments solution, despite having separate department and separate clients. This channel department was launched in 2004 and since that time had several evolution steps. Talking about the development of channel solutions it is needed to say that the core component of this system was developed that time and had minor changes. Channel solutions department has two main teams with different processes in technical debt management.

System architecture

Because of two teams with different products, there are two separate systems with SOA in company B. One system is independent and another one is tight closely with the core system of the company which enables payments services

Teams' structure

Despite the common idea of the final product (for both teams it is internet banking and mobile banking) the teams itself and their processes are vary significantly.

Team 1 is responsible for the first solution of internet banking which was developing since the creation time of this solution. The problem of this solution is extremely high cohesion of the internet-banking logic with core payment services logic, the reason of this is the idea, that internet-banking would be the part of the whole payment services, but not independent and alienable solution. By the time when the understanding of the role of this solution as a separate one came, a huge volume of system logic and code lines were already developed and it was too hard and risky to try to set apart both of these system objects. As a head of channel solutions said during the interview: "*This logic could be separated only by surgical methods*". Team 1 is responsible for front-end and back-end components as well as integration of back-end with payment services and also for the integration with other external systems.

Team 2 is responsible for the relatively new solution (was introduced four years ago). This solution was partially based on the external ready-to-use back-end solution and front-end solution was developed by the team itself. This approach helped to avoid past problems with connectivity of payment services and internet-banking.

Software development process

Team composition also matters for software development process. Team 1 consists of 10 developers that are separated by front-end and back-end, 2 quality assurance engineers, and also architect of e-channels. In processes of team 1 people from implementation department also plays significant role, despite not having direct contribution to software development process, they communicate directly with clients in order to go through several steps, which are necessary to deliver ready solution for the client:

Team 1	Team 2
• Formulation of request / idea • Requirements gathering • Requirements formulation (in user stories) • Agreement on solution • Development	• Formulation of request / idea • Requirements gathering and formulation (in user stories) • Agreement on solution • Development • Code review

• Code review • Testing • Bugs fixing • Release • Implementation • Testing and Commercial operation • Formulation of additional requirements • Additional development	• Testing • Bugs fixing • Release • Implementation • Testing and Commercial operation • Formulation of additional requirements • Additional development

Despite development processes steps are very similar for team 1 and team 2, inside they have a dramatic difference – team 1 has implementation engineers as intermediaries, and team 2 interacts with client directly, with the help of product owner. Detailed processes with stakeholders for team 1 and 2 are presented in Appendixes on Figure 13 and Figure 14.

It was said that employees from implementation department could act as integration engineers, system analysts, business analysts, project managers besides the main role of implementation engineer. The problem that is hide there is that because of the gap implementation engineers could not know the realization in precise details, which leads to the work through usual, gained implementation scenarios. This non-optimal implementation solutions may lead to increasing costs of maintenance and also increase the time of development of new features.

Team 2 consists only of 4 people, two of them are full-stack developers, one is front-end developer and one is back-end. The team also has product owner, who is responsible for communication with clients, requirements gathering and final solution delivery. The team along with product owner, gather clients' requirements and implements it on a client side. Also team 2 teats developed solutions by itself.

Technical debt management activities

Identification

While developing new feature or manually by architect or developer, special tools are not used as for this moment integrated development environment is enough. Recently, the project of test coverage was launched. Also, periodically there is a technical debt inventory.

Measurement

By expert estimation or by blind votes of developers and after discussions of the results.

Repayment

Mostly, when experienced team member feels that the critical moment of the system reliability is close, by initiating refactoring task. Sometimes, when there is a vacant development forces, by doing refactoring during this time. Sometimes in cases when system falls.

Communication

Could be divided into internal and external communication. Among developers communication is working well, but if consider communication between developers and implementation department or business development side, sometimes communication may be difficult because of contradictory goals.

Communication with clients also vary: for majority of the clients technical debt constrains would be shown as delays, but with some clients (who have their own development, technical debt is discussed)

Prevention

By informal agreement on particular solutions in some cases, by formal approve from architect, by required code reviews.

Prioritization

By the feeling of developers, by requirements from business.

Monitoring

By checking the readiness of inventoried tasks, but them are rarely checked and some tasks could even expire.

Representation / documentation

By technical debt inventory and backlog tasks.

3.3.Company C case study

The company C is an IT company that specializes in the development of software for medical institutions and also provides various services such as consulting, supplying, implementing and maintaining this specialized software. Automation of medical institutions and introduction of medical information technologies are the main specialization of the company.

The company operates in the market for 10 years, and since that time it is hard to say exactly, when the company started to feel the burden of technical debt. But it could be noted, when the company became more involved modifications of the system to the requirements of current customers than when it was time "to capture the market" and numerous implementations of model functionality. The market right now is divided, the system meets the basic requirements of the federal legislation, so customers began to develop their existing system to fit their specific requirements – "*And here were revealed system imperfections and drawbacks*".

For example, recently there was a case: Customers began to complain about the poor performance of one module after another new version. Climbed into the code - a bunch of code all made once for a specific customer. The company decided to remove pieces of "outdated" code and the code of the customer, who is no longer on the system (moved to another system, or use some old version and have not in the tech support).

The system was originally written on commercial American platform. Convenient document management system, which has turned out very quickly build up the necessary functionality for the medical information system (MIS). At first everything was great, but after several months it became clear that technical limitations of the platform has bad influence on the system development. However, lots of code was already written and system functionality works with this code and in was too difficult to change the platform However, for the other part f the business (regional solutions) was decided to switch to the open-source platforms, but the old solution still was “living” on the old platform. Finally, when the government have forbidden the usage of commercial foreign software for public companies, company C had no choice except from moving its old solution to the not-forbidden open-source platform. to free software and all new products are developing with free software.

Software development process

- Formulation of request / idea
- Requirements gathering
- Requirements formulation (in user stories)
- Agreement on solution
- Development
- Code review
- Testing
- Bugs fixing
- Release
- Implementation
- Testing and Commercial operation
- Formulation of additional requirements
- Additional development

Detailed processes with stakeholders for company C is presented in Appendixes on Figure 15..

Technical debt causes

In company C were identified the following causes of technical debt:

- the pressure of deadlines for the development of new functionality;
- insufficient code coverage by tests;
- lack of competence of some developers;
- is too complex to implement new process into existing the system, partly because of customization for different clients
- "legacy" of the existing system;
- changing customer requirements during the project;

Identification

Mostly by accident (when it is difficult to develop new feature or when the system couldn't cope with overload). When there is some kind of global critical situation with the system (critical speed is reduced, falls stupidly system) - begins a massive refactoring.

Measurement

By expert estimations.

Repayment

Mostly in cases when the system falls. Repayment is made by refactoring of the code and this process could be time-consuming. Example - last summer fell Electronic Registry and a month and a half the company was doing refactoring.

Communication

For the external clients technical debts is not shown, all drawbacks of the systems are presented as temporary issue. Inside the company the topic of technical debt is discussed widely on different levels of organizational structure.

Prevention

By required code review procedure before release. By required approve of the solution from a particular number of people in the company.

Prioritization

By developers' opinion, by clients needs. When making decisions is taken into account, the demand for functional at the customer. For example, there was a unit "Medical institution web-site" where everything was working poorly, and only 3-4 client used it, and this module interfered the other modules. The solution for the problem was: "*Well, we just removed it from the new version*".

Monitoring

By checking the readiness of set tasks.

Representation / documentation

By expert estimations. Also with All the tasks on rework - a programmer in the internal system indicates a separate task that goes to the analyst.

3.4.Cross-case study comparison

Software development processes

One of the main differences between Company A which provides services and Companies B and C which creates products for particular clients is that for delivering final solution to the clients, companies B and C need implementation engineers as an intermediary between clients and development. This fact creates additional complexity in directions below:

- Requirements gathering.
- Feedback receiving.
- Implementation processes.

Implementation engineers may become sources of technical debt in several ways. First of all, implementation engineer remembers, how he acted in previous projects, he usually considers implementation for new client as an implementation of the same product he delivered before. However, very often, solution for a new client was changed by the development team and now implementation may be done in a different way, but implementation engineer doesn't aware of this changes and will act as of old. Another possible source of debt caused by implementation engineer is their mentality of enduring inconvenience: tight deadlines, clients' requirements – all of these cause the attitude which is based on clear goal: deliver solution to the client at time and with budget frames. It means that implementation engineer may find non-optimal, rough ways of implementation using existing system capabilities, instead of say out about the problem and find better solution with development team.

To reduce the negative impact on the system caused by implementation processes, the team 1 of Company B is trying to engage developers in implementation activities – starting from requirements gathering and going to implementation. These procedures help not only straighten communication with the client but also establish better mutual relations of implementation team and development team.

Company A has no implementation, however, the gap between initial requirements and development may appear when system analysts develop technical solution. It should be pointed out, that there is a difference between technical specification which answers the question "what?" and technical solution, which answers the question "how?". Therefore, if system analyst has lack

of communication with development team, it may lead to the choice of wrong direction from the very beginning or to the usage of incorrect input data. Outdated documentation may be the source of incorrect input data if system analyst develops the solution for the running process to add the new feature. Currently, the formal approval of technical solution is done by architects who read the final text, prepared by the analyst. However, there is a plan of changing development and approval of technical solutions processes by make it more communicative among all stakeholders.

The comparison of the companies' context is shown in Table 10. Cross-cases context description

The comparison of companies' technical debt management activities are presents in Table 11.

Table 10. Cross-cases context description

	Company A	Company B	Company C
Market	B2C	B2B	B2G
Product	Payment services	Internet and mobile banking	Medical information systems
Software development process	Quarter planning, several projects in quarter, inside project planned short sprints, regulated agreement processes in all stages of development.	After receiving requirements from client, the process from building definitions of done to the final implementation.	After signing the contract with client, standard process from requirements gathering to implementation with formal controls on each stage.
Development methodology	Agile, SCRUM-like	Agile, Scrum and Kanban-like	Waterfall-like
Teams' structure	14 teams, each has project manager, one or more product owner, one or more front-end developer, one or more backend developer, one or more quality assurance engineer. Some teams has analyst as a team member.	4 different teams, which has no required roles (several teams consist only of developers, one team consists of developers, quality assurance engineers and implementation engineers who are not the formal members of the team, but may play role of project managers)	There is no team-like organizational structure instead, company is divided in departments (web-applications development, development based on foreign commercial software platform, quality assurance, implementation).
Architecture of the system	SOA, strategic goal to make it more micro services-like.	Two separate systems with SOA. One system is independent and another one is tight closely with the core system of the company which enables payments services	Two separated SOA systems (one is based on free software and the other is based on foreign commercial software platform)
Product owners	Internal, each team has product owner	External, banks. In one company there is internal product owner, who closely communicate with client	External (medical institutions, 95% from public sector)

Table 11. Cross-cases analysis of technical debt management activities

	Company A	**Company B**	**Company C**
Identification	Sometimes when new feature is development; manually, by software architect or by special tools by developers (ex. Jenkins).	While developing new feature or manually by architect or developer, special tools are not used as for this moment integrated development environment is enough. Recently, the project of test coverage was launched. Also, periodically there is a technical debt inventory.	Mostly by accident (when it is difficult to develop new feature or when the system couldn't cope with overload).
Measurement	By expert estimation in human-weeks (human-days).	By expert estimation or by blind votes of developers and after discussions of the results.	By expert estimations.
Repayment	Is divided into two main directions: strategic – one team of 4 front-end and 5 back-end developers was created only for conducting refactoring/ rewriting tasks with system architect as a product owner. Operational – by assigning particular time for backlog refactoring tasks related to the team.	Sometimes in cases when system falls, mostly, when experienced team member feels that the critical moment of the system reliability is close, by initiating refactoring task. Sometimes, when there is a vacant development forces, by doing refactoring during this time.	Mostly in cases when the system falls. Repayment is made by refactoring of the code and this process could be time-consuming.
Communication	Is supported by company meetings in order to ensure the common understanding of current technical debt situation and its further management activities. For operational level it is needed to build a common idea with	Could be divided into internal and external communication. Among developers communication is working well, but if consider communication between developers and implementation department or business	For the external clients technical debts id not shown, all drawbacks of the systems are presented as temporary issue. Inside the company the topic of technical debt is discussed widely on

	product owner in order to explain him/her what consequences for the business could be.	development side, sometimes communication may be difficult because of contradictory goals. Communication with clients also vary: for majority of the clients technical debt constrains would be shown as delays, but with some clients (who have their own development, technical debt is discussed)	different levels of organizational structure.
Prevention	By approving by architectural comity of new solutions, by test coverage, require code review, by setting the culture of high-standards programming (along with seniors development in refactoring team there are several junior developers, who accumulating best coding practices).	By informal agreement on particular solutions in some cases, by formal approve from architect, by required code reviews.	By required code review procedure before release. By required approve of the solution from a particular number of people in the company.
Prioritization	By running cost-benefit analysis, by expert opinion, by communication with product owner	By the feeling of developers, by requirements from business.	By developers' opinion, by clients needs.
Monitoring	By checking the readiness of set tasks, by covering code by tests.	The readiness of inventoried tasks is rarely checked	By checking the readiness of set tasks.
Representation / documentation	Detailed description of the components in a system (including visualized processes), description of the desired functions of the components, written plan of actions (what should be changed and where). Also by other teams' backlog tasks	By technical debt inventory and backlog tasks	By upper-level description of the drawbacks in the system.

3.5.Discussion

The research aim was to investigate how technical debt is managed across Russian software development companies. The answers on research questions are presented below.

RQ1. What sources of technical debt appear through software development process in Russian software development companies?

It was found out during the interviews, that all three firms are exposed to both types of technical debt: short-term and long-term, and the sources of each type differs crucially.

Sources of short-term technical debt.

Short-term debt was defined as an operational or tactical one, that appears during the development process in the form of small bugs and other code imperfections. The following sources of short-term technical debt sources were identified. The sources are shown in Table 12.

Table 12 The sources of technical debt

Sources of short-term TD	**Description (companies)**
Communication issues	• Lack of communication in the project team (A, B, C) • Lack of communication with business client (B, C) • Indirect communication between business client and programmers (B, C) • *'Mentality of patience'* inside the implementation team (C)
Requirements issues	Change of the business client's or internal requirements for the system (A, B, C)
Testing issues	(A, B, C)
Infrastructure issues	Hardware does not keep up with the software; performance issues (A, B, C)
Time issues	Software should be developed in very tight time frames. (A, B, C)
Developers competences issues	Developers with lower competences tend to make more mistakes and shortcomings in the code design and structure which leads to the emergence of technical debt (A, B, C)

Four out of five interviewed experts stated that communication flaw in the project team and with the business client is the primary source of the *'bad technical debt'*.

The system architect of the company A mentioned that *'there is no much communication and interaction between analyst teams (those who prepare the requirements for the system changes) and the development team'*. This results in the increasing the timing and inconvenience of the development process and growing number of system imperfections. Furthermore, *'the process of technical solution alignment is not perfect as well'.* At this moment the process is the following. Analyst team prepares technical solutions and upload it in the internet portal for review and approval of system architects. System architects read the solution and discuss it in the architect commission with our project team members. Such a process according to the interviewed experts causes a long debate, and make the project team concentrate on the small details, but not the whole picture.

In the company B, as it was mentioned previously, there are two teams. In the one team there is a complex indirect communication between project team and business client resulting in the emergence of high amount of operational technical debt. In the other team, the communication process is much smoother, because of the *'developer-in-the-field'*, working on the client side and gathering the requirements. Furthermore, in that team client is fully involved in the process of software development. There are even common practices of managing and prioritizing of technical debt interest payments. Now the company is thinking about transferring these communication practices to both teams.

Company C project manager admitted that *'client and project team communication, being the largest source of operational technical debt, is a stumbling block for the company'*. Implementation team which is responsible for designing system device as well as gathering system requirements. In company C implementation team tend to *'go on about the business client'*, without proper advising with programmers. As a result, a lot of *'crutches and bugs'* appear that would need to be paid off sometime.

Overall, it was confirmed from the interviews, that sources of short-term technical debt falling into five different categories are quite the same for all three companies with the communication issues being the most serious source of short-term technical debt.

Sources of long-term technical debt.

Long term technical debt is a strategic one aimed at fulfilling a strategic goal not only of the software development unit, but also of the whole enterprise.

In the company A (B2C), the source of the long-term technical debt come from the internal environment. Recently a new CIO was hired. Having his own vision, he had changed the priorities

for system development to the side of the agile microservices system, which required a lot of changes (refactoring) in the current system.

In the company B (B2B), historically, internet banking and card processing systems were inextricably linked. There was no intention in the past to separate those system, and now because of that limitation company B is struggling at growing its customer base. Some clients may need only internet banking without processing, but technically it is not possible to provide such an option. Furthermore, it is becoming harder to develop additional program feature in that unified complex system.

Company C (B2G and B2B) had its major product built on the American commercial platform, and because of the recent Russian law that forbid the usage of foreign software in public companies, company C is switching to the open (free) platform, having a lot code to be refactored.

The long-term technical source common for all three companies is the shifts in the external environment, like change in customer preferences, competitors moves or emergence of a new technologies, which could make the companies to recognize technical debt and make them to start code refactoring in order to remain competitive with their product.

Overall, long-term technical debt sources are very context oriented and depend on many factors like company's business model, internal vision and changes in the external environment.

RQ2. What context-related technical debt management practices could be identified in Russian software development companies?

In all the researched companies after the technical debt has been identified, there is a dilemma: to pay it off right away, to delay the payment of the technical debt interest or to forget about the technical debt at all. At first, the technical debt is being analyzed by the programmer who has identified it, whether it is a critical one, which should be tackled right away, or not a critical one, which could be delayed. Three out of five experts said this evaluation is usually done intuitively with the appliance of some sort of the cost-benefit analysis where the programmer together with people from business side compare cost (or consequences) and benefits of holding technical debt to the benefits and cost of paying it off. If total benefit of paying it off outweigh, then refactoring is done, otherwise, refactoring is being deferred.

In the company A, if a programmer identifies flaw in the code logic, the special task (ticket) should be created in the special bug tracking task management system. That flaw is added into the system in accordance with the defect priority matrix developed by the company, and is tackled respectively. The time to pay off the technical debt in this case is set in the ticket according to the priority matrix and usually is solved on time.

In the company B, the technical debt inventory is held every six months. That inventory is aimed at revising the system architecture, and all the found code inconsistencies are being added

into the task pool for the execution. *'Despite the existence of the task pool of the technical debt (defects, bugs, code revisions, etc.), the executing of these tasks are not tracked by anyone, and after six months there could be still a lot of tasks in the task pool. Some of them could be outdated and would not require to be paid off anymore'.*

In the company C there is no special procedure to cope with technical debt. Usually it is paid off, only on the demand of the business client, or when the incident occurs affecting the reliability and vital functions of the system. *'We change something only when there is a vital need for this'* - the company's project manager said. The main reason for this is the lack of time and resources for prevention methods.

Based on the company's business model, it was found out that that B2C companies (company A) are more willing to pay off technical debt than B2B and B2G software development companies (companies B and C). The system architect of the company A states that this is fact, because '*in B2C software development business the risk and the level of responsiveness of making the mistake is lower, whereas in B2B (or B2G) there is a very high level of responsiveness to the business client, with whom usually you have a strict service level agreement (SLA). That SLA usually includes strict fines for the system malfunctioning, therefore these companies are very cautious about changes in the system code structure.'* Moreover, due to the market conditions, clients are perceived by B2B (B2G) companies like this: *''They are few and each of them is higly important for us''.* Therefore, when it comes to the decision of paying of technical debt or implement new feature, very often the decision is taken in a favor of second options, in order to correspond clients' needs.

RQ3. What technical debt management activities could be considered as mature in Russian software development companies? What methods are used to support these activities?

The conducted research has shown that there are two groups of prevention method used by Russian software development companies: industry common methods and companies specific methods.

Industry common method are the ones used across all the software development company to prevent the appearance of technical debt. According to the company's B head of channel solution, such methods are like a *'rules of good taste*, and every IT company should adopt them in order to *'keep themselves afloat'*. These methods include code review, testing, automatic deployment, alignment of technical solutions.

Companies specific methods are the ones that only adopted by the certain companies, and which are not commonly spread across the industry.

In the company A such practice are ‘Junior-senior refactoring’ and ‘Analyst-architect communication’. ‘Junior-senior refactoring’ is the practice of involving junior developers in the process of refactoring together with senior colleagues. As result, junior developers would acquire best practices from the more senior colleagues, and the quality of the code would increase, consequently leading to the prevention of technical debt emergence. ‘Analyst - architect direct communication’ would lead to better communication and as a result to higher quality technical solutions, which would allow developers to code easily without inventing any ‘crutches’.
In the company B the culture of a beautiful code is widely promoted. Beautiful code is the one that has a perfect structure and would be easy to edit in the future. Furthermore, in order to increase the quality of communication, the developers are involved in the process of gathering requirements and designing the device of the system together with the business client. The head of company’s technical solutions called this procedure as *‘Developer- in-the-field’*

Company C does not have any specific practices devoted to prevention of technical debt, except industry common methods. The attitude to the technical debt prevention is quite immature in that company. The overall comparison is shown in Table 13 Technical debt prevention methods

Table 13 Technical debt prevention methods

Methods	**Company A**	**Company B**	**Company C**
Code review	+	+	+
Testing	+	+	+
Automatic deployment	+	+	+
Alignment of technical solution	+ (formal)	+ (informal)	+ (sometimes formal)
‘Developer-in-the-field’	not applicable	+	-
‘Junior-senior refactoring’	+	-	-
‘Analyst-architect direct communication’	+	-	-
‘Beautiful code culture’	+	+	-

RQ4 What factors should be considered during decision-making processes about managing technical debt?

Based on the conducted interviews, five key factors affecting technical debt management were identified. They are shown in the Table 14.

Table 14 Factor that affect decisions on technical debt management

Factor	Description
Time	Could refactoring of the component be postponed without affecting company's performance?
Team size and structure	Are there enough resources to make the refactoring? Is the team aware of the importance of dealing with technical debt?
Top management attitude	Does top management understand the importance of technical debt and have it's own vision towards managing it?
Type of client	How demanded is our client in terms of technical debt management (B2B or B2C client)? How the system of our client may be affected by our changes?
Importance of the module (component)	How important is the component for the system development ? (Prioritization of the component refactoring based on it's importance) Is refactoring done to the business needs or to the needs of code beauty?

3.6.Conclusion and implications

In this study the practices of technical debt management in Russian software companies were investigated. The purpose of this research was to study the reasons of the emergence of technical debt, to investigate the ways to manage technical debt in Russian software development companies, and also to identify factors that affect the decision-making on technical debt management. Three Russian software development companies were analyzed. An important aim in the study of technical debt in these companies was to understand the context of software development, which includes the market in which the company operates the development process,

the structure and size of the development team, and the age and the history of the system development in the company. As results of this study, the reasons for the emergence of technical debt, the common ways of managing it in all studied companies were found. Furthermore, there were identified common factors that influence the decision-making on the management of technical debt. In addition, the main differences in the methods of managing technical debt in companies operating in different markets were shown as well as some recommendations were given.

Managerial implication

The results of this research could be applied into business practices in several directions. Nowadays, software development companies are seeking ways to manage technical debt, to find the ways to prevent avoidable technical debt. This study by mapping software development process steps with active participants in each step helped to identify the steps on which technical debt could occur and to classify the possible type and the cause technical debt appearance. In each company common practices for improving the quality of final solutions were revealed, they are: solutions agreement (formal or informal), code review, testing, bugs fixing. Furthermore, for some companies could be useful informal practices, such as 'Junior-senior refactoring' and 'Beautiful code culture'.

However, by conducting this analysis, the communication gap was also revealed for all companies. This gap relates to interpretation of business requirements by different participants and lack of communication between them on each step. Moreover, communication gap affects B2B companies during implementation stage, because of mentality and goals of implementation engineers. Therefore, in order to prevent avoidable technical debt, it is necessary to apply practices which allow striating the communication between business people, analysts and developers for B2C companies and clients, implementation engineers and developers for B2B (B2G) companies. These methods could be formal and could require direct interactions of all needed participants.

From the research it was revealed that B2B (B2G) companies are highly client-dependent. In shows off in two ways. The first cause is that they are beware of changing something in the system until the high or critical need for it comes out. The second cause is that due to the business environment, when market is a kind of already divided, the major source of finance for companies is provision of improvements or customization for existing clients. Therefore, very often, companies decide to implement new feature for the client instead of paying off technical debt.

A possible way to overcome this problem is to include the risk of technical debt payment during development phase and to define longer time frame for a particular project. Another possible way is to set up a process of technical debt communication with the client. It is not applicable for all clients (for example, it could not work with B2G client), however, if the client

has his own development team, it is possible to communicate on technical debt topic and together, with client development team provide more smooth solution.

Considering B2C companies, they are more about to change the system, because they do not have limitations from the clients' side and also they are interested in more flexible and convenient development process. Some problems could occur, when business need meets the obstacles from development side – the impossibility of developing new feature in short-term, because of system limitations (the need of paying off technical debt before developing). And at this moment communication process between product owner (business people of the company) and development appear. Through communication it is needed to answer several questions:

What does new functionality give for the business?

What would happen if we do not pay off technical debt?

By answering these questions though communication, it is possible to reach an agreement based on facts and logic come both from business side and development side.

Research implication

This research was conducted in order to contribute to empirical studies of technical debt management in Russian software development companies. Another contribution of this study is that the research investigated deeply the context of technical debt management in studied companies. The context includes companies' markets (B2C, B2B and B2G), the age of the system, software development processes, active participants of development processes, and historical overview of companies' systems development with emphasis on some important points, critical in decision-making process. The sources of technical debt was also investigated the context with the sources were linked with technical debt management activities. It was identified that the context, including past decisions, made at the dawn of the company, have significant influence on current decisions regarding technical debt management.

The research also revealed high importance of communication process though development process for all companies in order to prevent technical debt and therefore, opens directions for further research in investigation of the impact of the quality of communication during development process on the amount unconscious technical debt.

The study has also identified the importance of clients' needs for B2B and B2G companies during decision-making process about whether to pay off technical debt. And external client could be considered as additional limitation factor in prioritizing technical debt pay offs.

Research limitations

It also should be noted that technical debt management, being emerging concept, do not have yet commonly accepted "best practices". As for studied companies, technical debt management practices have different level of maturity for different activities. For example,

prioritizing process of technical debt for all companies is more ad hoc, without applying special models or frameworks. Therefore, is hard to say, whether the practice of one company is definitely more efficient than the practice of the other one. Furthermore, in order to compare the practices financial data about projects and costs of technical debt payment is needed and this information could be closed for the research (for example, one of the companies agreed to give an interview, only if there is no revealing of financial data).

Time frame could also be considered as limitation, some approaches of technical debt management was implemented in company not long time ago, and, therefore, the long-term effect of implemented strategy has not shown up yet.

The number of companies for the research are also can be considered as a limitation, however, as the aim of the study was to investigate technical debt management practices in a context of the company, the deepness of the research was more important.

Respondents' bias could also be considered as a limitation, however, for companies A and B it was partly mitigated by conduction interviews with two representatives of these companies separately.

List of references

Alves, Nicolli S.r., Thiago S. Mendes, Manoel G. De Mendonça, Rodrigo O. Spínola, Forrest Shull, and Carolyn Seaman. "Identification and management of technical debt: A systematic mapping study." *Information and Software Technology* 70, 2016

"Ward Explains Debt Metaphor". 2017. *Wiki.c2.Com*. Accessed May 05. http://wiki.c2.com/?WardExplainsDebtMetaphor.

Ambler, S.W.: Agility at Scale: Become as agile as you can be. IBM, Toronto (2009). eBook at ftp://ftp.software.ibm.com/software/au/201106/Agility_at_scale.pdf

Beck et al. (2001), Manifesto for Agile Software Development. http://www.agilemanifesto.org/ (accessed November 24, 2016).

Behutiye, Woubshet Nema, Pilar Rodríguez, Markku Oivo, and Ayşe Tosun. "Analyzing the concept of technical debt in the context of agile software development: A systematic literature review." *Information and Software Technology* 82, 2017, 139-58.

Boehm, B.W., Turner, R.: Balancing Agility and Discipline--A guide for the perplexed. Addison-Wesley, Boston, MA (2003) p. 20

Bryman, A. and Bell, E. Business research methods. Oxford University Press, 2003

Cockburn, Alistair. 2005. *Crystal Clear*. 1st ed. Boston: Addison-Wesley. p. 45-48

Ernst, Neil A., Stephany Bellomo, Ipek Ozkaya, Robert L. Nord, and Ian Gorton. "Measure it? Manage it? Ignore it? Software practitioners and technical debt." *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015*, 2015.

Falessi D., Voegele A. Validating and prioritizing quality rules for managing technical debt: An industrial case study. Conference Pape October 2015. *Conference: 2015 IEEE 7th International Workshop on Managing Technical Debt (MTD)*

Falessi, Davide, Philippe Kruchten, Robert L. Nord, and Ipek Ozkaya. "Technical debt at the crossroads of research and practice." *ACM SIGSOFT Software Engineering Notes 39*, 2014

Fernandez-Sanchez, Carlos, Juan Garbajosa, Carlos Vidal, and Agustin Yague. "An Analysis of Techniques and Methods for Technical Debt Management: A Reflection from the Architecture

Perspective." *2015 IEEE/ACM 2nd International Workshop on Software Architecture and Metrics*, 2015.

Fowler, Martin. 2003. "Technicaldebt". *Martinfowler.Com*. https://martinfowler.com/bliki/TechnicalDebt.html.

Fowler, Martin. 2009. "Technical Debt Quadrant". *Martinfowler.Com*. https://martinfowler.com/bliki/TechnicalDebtQuadrant.html.

Fowler, Martin. 2013. *Refactoring: Improving The Design Of Existing Code*. 2nd ed. Boston: Addison-Wesley.

Guo Yuepu, Carolyn Seaman, and Fabio Q.B. da Silva. 2016. "Costs And Obstacles Encountered In Technical Debt Management – A Case Study". *Journal Of Systems And Software* 120: 156-169.

Kruchten, Philippe, Robert L. Nord, and Ipek Ozkaya. "Technical Debt: From Metaphor to Theory and Practice." *IEEE Software* 29, no. 6, 2012: 18-21.

Kruchten, Philippe. 2011. "Contextualizing Agile Software Development". *Journal Of Software: Evolution And Process* 25 (4): 351-361. doi:10.1002/smr.572.

Letouzey, Jean-Louis, and Declan Whelan. 2016. "Introduction To The Technical Debt Concept". https://www.agilealliance.org/wp-content/uploads/2016/05/IntroductiontotheTechnicalDebtConcept-V-02.pdf.

Letouzey, Jean-Louis. 2016. "Introduction To The Technical Debt Concept". Agilealliance.Org. https://www.agilealliance.org/wp-content/uploads/2016/05/IntroductiontotheTechnicalDebtConcept-V-02.pdf.

Li, Zengyang, Paris Avgeriou, and Peng Liang. "A systematic mapping study on technical debt and its management." *Journal of Systems and Software* 101, 2015: 193-220.

Marinescu, R. Assessing technical debt by identifying design flaws in software systems, *IBM Journal of Research and Development*, 56(5), p. 9:1–9:13, 2012

Martini, Antonio, Jan Bosch, and Michel Chaudron. "Architecture Technical Debt: Understanding Causes and a Qualitative Model." *40th EUROMICRO Conference on Software Engineering and Advanced Applications*, 2014.

McConnell, Steve. 2007. "Technical Debt Taxonomy". *Construx.Com*. http://www.construx.com/10x_Software_Development/Technical_Debt/.

Mohan, Babu K. " Vendor-Driven Technical Debt: Why It Matters and What to Do About It." *Cutter IT Journal* 29, no. 3, 2016: 33-37.

Mohan, Babu K. "Vendor-Driven Technical Debt: Why It Matters and What to Do About It" *Cutter IT Journal* 29, no. 3, 2016: 33-37.

Nord, Robert L., Ipek Ozkaya, Philippe Kruchten, and Marco Gonzalez-Rojas. "In Search of a Metric for Managing Architectural Technical Debt." *Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, 2012.

Radford, Barbara Davis & Darren. Going Beyond The Waterfall: Managing Scope Effectively Across the Project Life Cycle. J. Ross Publishing, 2014.

Runeson, P., Host, M., 2008. Guidelines for conducting and reporting case study research in software engineering. *Empir. Softw. Eng.* 14, 131-164.

Ru-Zhi, X., Tao, H., Dong-Sheng, C., Yun-Jiao, X., & Le-Qiu, Q. Reuse-oriented process component representation and retrieval. *The Fifth International Conference on Computer and Information Technology (CIT'05)*, 2005

Schön, E., Thomaschewski, J., & Escalona, M. J. (2017). Agile Requirements Engineering: A systematic literature review. *Computer Standards & Interfaces, 49*, 79-91.

Seaman, C.B., 1999. Qualitative methods in empirical studies of software engineer-ing. *IEEE Trans. Softw. Eng*. 25, 557-572.

Seaman, Carolyn, and Yuepu Guo. "Measuring and Monitoring Technical Debt." *Advances in Computers*, 2011, 25-46.

Shull, D. Falessi, C. Seaman, M. Diep, and L. Layman. Technical debt: Showing the way for better transfer of empirical results. *In Perspectives on the Future of Software Engineering*, pages 179-190, 2014

Stefanie Betz, Claes Wohlin, 2012. Alignment of business, architecture, process, and organisation in a software development context. ESEM 2012: 239-242

Tim Klinger , Peri Tarr , Patrick Wagstrom , Clay Williams, 2011. An enterprise perspective on technical debt, *2nd Workshop on Managing Technical Debt*, May 23-23, 2011, Waikiki, Honolulu, HI, USA

Welke Richard, Rudy Hirschheim, and Andrew Schwarz. 2010. "Service Oriented Architecture Maturity". *Computer*.

Yli-Huumo J., Maglyas A., Smolander K., Haller J., Törnroos H. (2016) Developing Processes to Increase Technical Debt Visibility and Manageability – An Action Research Study in Industry. In: Abrahamsson P., Jedlitschka A., Nguyen Duc A., Felderer M., Amasaki S., Mikkonen T. (eds) Product-Focused Software Process Improvement. PROFES 2016. Lecture Notes in Computer Science, vol 10027. Springer, Cham

Yli-Huumo Jesse, Andrey Maglyas, and Kari Smolander. 2014. The Sources and Approaches to Management of Technical Debt: A Case Study of Two Product Lines in a Middle-Size Finnish Software Company. Conference Paper, December 2014 *Conference: 15th International Conference, PROFES 2014, Helsinki, Finland, December 10-12, 2014*, At Helsinki, Finland, Volume: 8892

Yli-Huumo, Jesse, Andrey Maglyas, and Kari Smolander. 2016. "How Do Software Development Teams Manage Technical Debt? – An Empirical Study". *Journal Of Systems And Software* 120: 195-218.

Yin, R.K., 2003. Case Study Research: Design and Methods. Sage Publications, *Thousand Oaks, Calif.*

Zazworka N., Spínola R., Vetro' A., Shull F. , Seaman C. (2013). A Case Study on Effectively Identifying Technical Debt. In: *17th International Conference on Evaluation and Assessment in Software Engineering*, Porto de Galinhas, Brazil, April 14th - 16th , 2013. pp. 42-47

Appendix 1. Interview questions

1. **General questions about the experience and positions of interviewee**

a) How many years have you been working in the industry? How old is the company?

b) What is your role in the company?

2. **General questions about the architecture and system:**

a) Please describe the system architecture.

b) What do you remember the transitional moments in understanding architecture in the system?

3. **The process of developing new functionality:**

a) What development methodology used by your company?

b) How is the process of adding new functionality to the system?

- The idea, the formulation of requirements
- Analysis, writing the technical solution
- Development
- Code Review
- Testing
- Bug fixes
- Release functionality

c) What methods of control still exist?

d) If in the process of writing code the programmer knows that the resulting solution is not optimal, if he makes some notes in code or on a separate page?

4. **Organizational structure and composition of teams**

a) What positions in the company are directly related to the process of creating new features?

b) What are the size and structure of the teams which are responsible for software development?

5. **Technical debt**

a) At what point about it was clear that the system contains a technical debt, which must be fought?

b) Have there been any major changes in the understanding of those. debt for the company?

c) How did the attitude of the technical debt on the org structure in the company, in the development process?

d) What has influenced a change in attitude to the technical debt?

6. The causes of technical debt:

a) What are the main causes of the technical can be distinguished?

- timing pressure;
- insufficient code coverage (due to lack of time or financial resources);
- lack of competence of some developers;
- "Legacy" of the existing system - it is difficult to write code quickly and beautifully for a new functionality, because all tied strongly that the current running process, so you have to "crutches";
- changing customer requirements during the project - not enough money for a full analysis and testing;
- changing the system architecture;
- technological obsolescence;
- anything else;

7. Identification of technical debt:

a) What methods from a strategic point of view are used for the detection of technical debt? It examines whether the separate components of the system is particularly important, which contains the basic logic?

b) Allocated if such components, in which a large technical debt is valid and is not critical to the functioning of the system?

c) What methods are used to identify the technical debt from an operational point of view? (special programs for the detection of code coverage, code duplication detection, etc.)

8. Technical debt Measurement

a) how to measure the amount of technical debt? (in man-hours?)

9. Technical debt repayment:

b) How do you conduct the repayment technical debt?

c) During the development of new functionality simultaneously refactor code separate project or a separate team for refactoring Provided?

10. Other processes that relate to technical debt:

- Prioritization;
- Monitoring;
- Prevention;
- Document;
- Communication (. to make the debt visible for all stakeholders).

11. What factors should be considered when the decision about technical debt is being made?

12. Optional:

a) How does the management of technical debt that you are working in B2C / B2B / B2G market? What limitations do you see for managing technical debt?

b) What is the general attitude in the management of the technical debt? Do managers understand that you need to refactor the code or perceived as a clean waste of resources to nowhere?

c) Usually programmers do not like to read someone else's code, but love to write something new from scratch. But there are some programmers who like "clean code". Do you pay attention to the personal qualities of the programmer, giving him the task?

Appendix 2. Company B team 1 development process (Figure 13)

Figure 13 Company B team 1 possible technical debt appearance through development process.

Company B (Team 1)				
	Development process	Communication with	Type of technical debt possible apperance	Causes of possible technical debt
Client	Formulation of request / idea	Developers Architects	-	-
Implementation engineers	Requirements gathering	Client	-	-
Implementation engineers	Requirements formulation (in user stories)	Client	Architectural debt Design debt Infrastructure debt	Clients' needs oppose system architechture
Architect and development team	Agreement on solution	Developers	Architectural debt Design debt Infrastructure debt	Improper solution decision
Developers	Development	Implementation engineers	Code debt Defect debt	Development with code rules violation
Assigned group of developers	Code review	Developers	Code debt Defect debt	Approval of non-optimal code
QA engineers	Testing	Developers	Test debt	Lack of automated tests Not fully coverage of the code by tests
Developers	Bugs fixing	Implementation engineers QA engineers	Code debt Defect debt	Not all bugs were fixed
Operation department	Release	Developers	Infrastructure debt Versioning debt	Unforeseen load on the system Deployment problems
Implementation engineers	Implementation	Client	Design debt	Implementation engineer works with "old scheme"
Implementation engineers	Testing and Commercial operation	Client Developers	Design debt	Non-optimal implementation decision
Cliens	Formulation of additional equirements	Implementation engineers	Architectural debt Design debt	New requirements are hard to develop with current system processes
Operation department	Additional development	Developers	Architectural debt Design debt Code debt	New requirements may need "crutches" Time pressure

Appendix 3. Company B team 2 development process (Figure 14)

Figure 14 Company B team 2 possible technical debt appearance through development process

Company B (Team 2)				
	Development process	Communication with	Type of technical debt possible apperance	Causes of possible technical debt
Client	Formulation of request / idea	-	-	-
Product owner and developers	Requirements gathering and formulation	Client	-	-
Product owner and developers	Agreement on solution	-	Architectural debt Design debt Infrastructure debt	Improper solution decision
Developers	Development	Product owner	Code debt Defect debt	Development with code rules violation
Developers	Code review	-	Code debt Defect debt	Approval of non-optimal code
Developers	Testing	-	Test debt	Lack of automated tests Not fully coverage of the code by tests
Developers	Bugs fixing	-	Code debt Defect debt	Not all bugs were fixed
Developers	Release	Operation department	Infrastructure debt Versioning debt	Unforeseen load on the system Deployment problems
Product owner and developers	Implementation	Client	Design debt	Non-optimal implementation decision
Product owner and developers	Testing and Commercial operation	Client	Infrastructure debt Versioning debt	Unforeseen load on the system Deployment problems
Cliens	Formulation of additional equirements	Product owner and developers	Architectural debt Design debt	New requirements are hard to develop with current system processes
Developers	Additional development	Product owner	Code debt Defect debt	Not all bugs were fixed

Appendix 4. Company C development process (Figure 15Figure 14)

Figure 15 Company C possible technical debt appearance through development process