ОБЗОРЫ

УДК 004.053

ОБЗОР АВТОМАТИЗИРОВАННОГО ИЗМЕРЕНИЯ ТЕХНИЧЕСКОГО ДОЛГА

Хомяков И.А.

АНО Университет Иннополис, Иннополис, e-mail: i.khomyakov@innopolis.ru

Неясно, какие меры и методы Технического Долга (ТД) следует использовать в каких обстоятельствах, и за последние несколько лет было предложено множество определений и подходов. Он почти никогда не основывается на существующих подходах, и их обоснованность учитывается в очень небольшом количестве проектов. В результате практикующие находят такие подходы запутанными и поэтому затрудняются их использовать. Цели: чтобы помочь практикам и исследователям понять доступные альтернативы и правильно их применять, в этой статье исследуются способы оценки ТД с использованием автоматизированных инструментов. Методы: систематический обзор был проведен по 1063 статьям из трех крупных электронных библиотек, которые были собраны из наиболее актуальных цифровых баз данных. Остальные 46 работ детально анализируются после применения всех этапов фильтрации. Полученные результаты: найденные статьи редко разрабатывают или проверяют существующие методы, и в основном они предлагают новые подходы к измерению ТД. Выводы: не существует независимых оценок моделей, предлагаемых в этой области, и эта область еще не развита. Отсутствие консолидации очевидно в литературе, поскольку авторы сосредотачиваются на отстаивании новых подходов. Кроме того, почти все подходы автоматизированы лишь до некоторой степени в тех статьях, в которых они предлагаются, и редко поддерживаются. Более того, большинство предлагаемых инструментов представляют собой прототипы, которые не обслуживаются и могут использоваться для поддержки исследований, проанализированных в статье. Из-за этих фактов практикующим специалистам сложно использовать такие методы.

Ключевые слова: технический долг, измерение, обзор литературы

OVERVIEW OF THE AUTOMATED MEASUREMENT OF TECHNICAL DEBT

Khomyakov I.A.

Innopolis University, Innopolis, e-mail: i.khomyakov@innopolis.ru

It is unclear what Technical Debt (TD) measures and methods should be used in what circumstances, and many definitions and approaches have been proposed over the past few years. It is almost never based on existing approaches, and their validity is considered in very few projects. As a result, practitioners find such approaches confusing and therefore have difficulty using them. Aims: To help practitioners and researchers understand the available alternatives and apply them correctly, this article explores ways to evaluate TD using automated tools. METHODS: A systematic review was conducted of 1,063 articles from three major digital libraries that were collected from the most relevant digital databases. The remaining 46 papers were analyzed in detail after applying all filtering steps. The results: the articles found rarely develop or test existing methods, and they mostly propose new approaches to measuring TD. Conclusions: There are no independent evaluations of the models proposed in this area, and the field is still underdeveloped. The lack of consolidation is evident in the literature as authors focus on advocating new approaches. Furthermore, almost all approaches are automated only to some degree in the articles in which they are proposed and are rarely supported. Moreover, most of the proposed tools are prototypes that are not maintained and can be used to support the research analyzed in the article. Because of these facts, it is difficult for practitioners to use such methods.

Keywords: technical debt, measurement, literature review

Технический Долг (ТД) является одной из новейших концепций, которая была введена в разработку программного обеспечения. Она признает компромисс между качеством кода и необходимостью удовлетворять ожидания участников рынка (например, низкая стоимость, короткое время выхода на рынок и т.д.). Это типичная ситуация для компаний-стартапов, которые имеют строгие требования к тому, чтобы создать минимально жизнеспособный продукт (MVP), чтобы протестировать рынок и получить финансирование для дальнейшего существования. В таких условиях принятие неоптимальных решений, снижающих качество системы, ведущих к созданию стратегических ТД [1], способно стать ключевой стратегией для достижения успеха. В любом случае компании должны понимать, что такие неоптимальные решения требуют дополнительных усилий для исправления продукта в долгосрочной перспективе [2; 3]. Тем не менее создание ТД может быть ценной стратегией, позволяющей вывести продукты на рынок, зная, что этот долг должен быть выплачен (с процентами) в будущем. Это феномен был первоначально описан Уордом Каннингемом в 1992 году [4], введшим понятие ТД. Существует гораздо больше источников ТД, которые были изучены в недавнем времени, включающих в себя коммуникацию, сотрудничество между членами команды, документацией и индивидуальным отно-

шением [5; 6]. Поскольку ТД – это способ измерения усилий, необходимых для достижения наилучшего качества программной системы по сравнению с текущим состоянием, крайне важно иметь возможность измерить (или оценить) его. Важность такой работы доказывается тем простым фактом, что большинство программных проектов имеют некоторый ТД [7]. Возможность оценки ТД позволяет командам разработчиков и руководителям планировать работу должным образом. Может также случиться, что ТД слишком высок для оплаты [8], что требует различных подходов для его разрешения (например, переписывание системы). Однако знание о том, каким образом система достигла этого состояния, может помочь в выявлении ошибок и улучшить процесс разработки. Частые изменения программных артефактов (в основном исходного кода) без соответствующих мер по обеспечению качества быстро приводят к снижению качества программного обеспечения с увеличением затрат на дальнейшую разработку и эволюцию программного продукта в связи с увеличением ТД [9]. Более того, оценка ТД должна выполняться автоматически, чтобы не увеличивать нагрузку на разработчиков и иметь возможность постоянно отслеживать показатель оценки на любом этапе разработки. Это особенно полезно в сочетании с использованием Agile-подходов, поскольку их ориентированный на доставку характер и непрерывная адаптация к потребностям клиента могут быть более предрасположены к возникновению ТД по сравнению с традиционной разработкой программного обеспечения. Однако они также наиболее склонны платить ТД за счет правильной реализации рефакторинга. По всем вышеперечисленным причинам возможность автоматического измерения ТД имеет первостепенное значение для поддержки ежедневной работы разработчиков. В литературе существует множество различных подходов к ТД, и в данной работе представлен обширный анализ, показывающий текущее состояние исследований, расширяя работу тех же авторов в [10]. В данной работе мы расширили анализ, включив в него обширный ряд первичных исследований.

В данной работе исследуются доступные методы оценки ТД с помощью автоматизированных инструментов с целью помочь практикам и исследователям в понимании доступных вариантов и их правильном применении.

Материалы и методы исследования

Протокол, принятый для данного систематического обзора литературы, является протоколом, который был предложен Китченхэмом и Чартерсом [11] для проведения подобных обзоров в области программной инженерии. Основной целью данной работы является обзор существующих исследований и выделение аспектов, связанных с измерением ТД, поэтому мы определили следующие исследовательские вопросы. Вопрос 1: какие существуют методы измерения ТД? Вопрос 2: какие инструменты поддерживают автоматизацию измерения ТД? Вопрос 3: существуют ли эмпирические исследования, способные продемонстрировать полезность выявленных методов? Вопрос 4: существуют ли эмпирические исследования, способные продемонстрировать полезность выявленных инструментов? Чтобы ответить на вопросы исследования, мы провели поиск статей с использованием трех крупнейших электронных библиотек: Цифровая библиотека Ассоциации вычислительной техники (ACM), Полнотекстовая база данных IEEE (The Institute of Electrical and Electronics Engineers,Inc.), и цифровая библиотека Гугл Сколар (Google Scholar). Поскольку для нашей цели интересны только исследования, посвященные ТД как основной теме, мы предполагаем, что их название или аннотация включают ключевые слова «технический долг». Следовательно, мы использовали соответствующие запросы для каждой библиотеки. Мы нашли 1 063 статьи, распределенные следующим образом: Цифровая библиотека Ассоциации вычислительной техники (211), Полнотекстовая база данных IEEE (317), Гугл Академия (535). Как и ожидалось, статьи, найденные в разных библиотеках, значительно пересекались. Поэтому первым шагом было объединение результатов и удаление дубликатов. К концу процесса мы отобрали 46 статей.

Шаг 1: объединение всех документов из источников данных. Первоначальный список включал 1063 статьи, но в нем было много дубликатов. Идентификация дубликатов проводилась вручную, чтобы избежать проблем, связанных с незначительными различиями в символах в названиях и именах авторов. В итоге мы получили список из 835 уникальных статей.

Шаг 2: применение критериев исключения. На данном этапе мы применили критерии исключения, в результате чего было отобрано 524 работы. Здесь мы все еще сохраняли в списке вторичные исследования.

Шаг 3: исключение не первичных исследований. На данном этапе мы определили вторичные исследования (например, систематические обзоры, систематические отображения и т.д.), и список сократился до 452 работ.

Шаг 4: рассмотрение исследований, связанных с измерением ТД. Во время чтения названия и аннотаций 452 статей мы определили работы, касающиеся вопроса измерения ТД. Мы выявили 77 работ, опубликованных в период с 2011 по 2021 год.

Шаг 5: оценка качества. Мы внимательно ознакомились с 77 отобранными работами и исключили 31 из них, поскольку они не касались измерения технического долга, даже если по названию или аннотации они казались подходящими для нашего исследования.

Результаты исследования и их обсуждение

Вопрос 1: какие существуют методы измерения ТД? Выявленные исследования были проанализированы с точки зрения предложенных методик, их требований к исходным данным, необходимым для расчета ТД, получаемой в результате методики информации, преимуществ и недостатков подхода. В табл. 1 суммируются все обнаруженные методы, их входные данные, выходные и расчёт.

Таблица 1

Обнаруженные методы для измерения ТД

Метод. Ссылка	Входные данные	Расчёт	Выходные данные
SQALE [12; 13]	1. Целевой уровень качества (список нефункциональных требований, определяющих правильный код). 2. Модель оценки долга (связь каждого требования функцией исправления, превращает количество несоответствий затраты неисправленные)	Запустить анализ кода с помощью инструментов анализа и использовать функции исправления, чтобы рассчитать затраты на исправление для каждого элемента. ТД – сумма затрат на исправление всех несоответствий. Этот долг называется индексом качества SQALE (SQI)	Симптомы дизайна ТД (пирамида-индикатор, представляющий конкретное распределение ТД по восьми характеристикам)
CAST [14]	1. Количество нарушений, которые следует исправить в приложении. 2. Часы исправления каждого нарушения. 3. Стоимость труда	((∑крайне тяжелые нарушения) x x (процент для исправления) x (среднее количество часов, необходимых для исправления) x ($ в час)) + + ((∑средние нарушения) x (процент для исправления) x (среднее количество часов, необходимых для исправления) x ($ в час)) + ((∑легкие нарушения) x (процент для исправления) x (среднее количество часов, необходимых для исправления) x ($ в час))	Стоимость исправления
SIG [15]	1. Исходный код. 2. Целевой уровень качества. 3. Стоимость труда	Для извлечения значений измерений из исходного кода используется SAT от SIG. RE = SS* RA* RF * TF $ME = \frac{MF*((1+r)^t*SS*TF)}{2^{QL-3}/2}$	1. Стоимость исправления. 2. Стоимость невмешательства
Модель на основе сравнительного анализа [16]	1. Выходные данные статических анализаторов кода (референсные проекты). 2. Исходный код. 3. Целевой уровень качества. 4. Стоимость рабочей силы	Поддержка инструмента доступна [17], что облегчает запуск инструментов анализа кода, а также создание базы данных тестов и набора тестов 1. указан целевой уровень качества 2. # максимально допустимых нарушений рассчитывается 3. # исправляемых нарушений рассчитывается 4. # нарушений, подлежащих устранению * расчетное усилие для фиксации* почасовая ставка	Стоимость устранения

Продолжение табл. 1

Метод. Ссылка	Входные данные	Расчёт	Выходные данные
Подход к моделированию, основанный на колебаниях [18]	Облачный мобильный сервис-кандидат	Количественная оценка ТД в течение первого года $TD_1 = 12*[ppm*(U_{max} - U_{curr}) - C_{u/m}* *(U_{max} - U_{curr})] = = 12*(U_{max} - U_{curr})*(ppm - C_{u/m})$ Со второго и далее $TD_i = 12*[K_{i-2}*[U_{max} - L_{i-2}] - M_{i-2}* *[U_{max} - L_{i-2}]] = 12*(U_{max} - L_{i-2})* *(K_{i-2} - M_{i-2}),\ i > 1$	Относительное количество ТД
Критическая точка для ТД [8]	1. Количество нарушений, которые следует исправить в приложении. 2. Часы исправления каждого нарушения. 3. Стоимость труда. 4. Прошлые изменения в истории системы (LOC)	$\text{Interest} = addedLoc* \left(1 - \frac{\text{FitnessValue}(\text{optimum})}{\text{FitnessValue}(\text{actual})}\right)$ $\text{versions} = \frac{\text{Principal}(\ )}{\text{Interest}(\ )}$	1. Стоимость устранения. 2. Стоимость не устранения. 3. Критическая точка ТД
LOC и Fan-In для количественной оценки САТД [19]	Исходный код	1. Извлечение комментариев и сопоставление их с соответствующими методами. 2. Определение изменения во времени в этих методах САТД. 3. Определение показателей, измеряющих долю. 4. Расчёт процентов для каждой доли САТД	Стоимость не устранения
Фреймворк ТД для дизайна [20]	Исходный код	1. Выбрать набор актуальных недостатков дизайна. 2. Определить правила для обнаружения каждого дефекта дизайна. 3. Измерить негативное влияние каждого обнаруженного экземпляра дефекта $FlawImpactScore(FIS)_{flaw\_instance} = = I_{flaw\_type}*G_{flaw\_type}*S_{flaw\_instance}$ 4. Подсчет общего балла $\text{DebtSymptomsIndex} = \frac{\sum FIS_{flaw\_instance}}{KLOC}$	Дизайн симптомы ТД
Схема оценки процента ТД [21]	Данные активности разработчиков	1. Запуск собраний. 2. Расчет показателей, связанных с усилием понимания в рамках собраний 3. $Interest(I) = I_{current} - I_{ideal}$ Статические метрики показывают наличие ТД в классах, метрики позволяют количественно оценить усилия по пониманию классов	Стоимость не устранения

Окончание табл. 1

Метод. Ссылка	Входные данные	Расчёт	Выходные данные
Метрики модульности для АТД и прошлые изменения в истории системы (записи фиксации) [22]	Исходный код	1. Анализ записи фиксации, чтобы извлечь необходимые элементы данных для расчета ANMCC. 2. Фильтрация данных в записях коммитов $\mathrm{ANMCC} = \left(\sum_{j=1}^{h} \mathrm{NMC}(k+j) \right) / h$ Более высокий ANMCC влечет за собой потенциальное увеличение АТД. 1. Создание карты кода (XML). 2. Парсинг карты кода. 3. Расчет показателей модульности более высокий IPCI или IPGF указывает на меньшее АТД и относительное количество ТД	Относительное количество ТД
Обнаружение и количественная оценка САТД [23]	Исходный код	1. Извлечение данных проекта (используемая версия, количество классов, общее количество строк исходного кода, общее количество извлечённых комментариев и количество участников). 2. Разбор исходного кода и извлечение комментариев к коду. 3. Фильтрация комментариев. 4. Ручная классификация на пять различных типов САТД и количество комментариев (количество отдельных строк, блоков и комментариев Javadoc)	Относительное количество ТД

Таблица 2

Вопрос 2: какие инструменты поддерживают автоматизацию процесса измерения ТД?

Метод [Источник]	Инструмент	Ссылка на инструмент	Открытый доступ
SQALE [12; 13]	SonarQube	sonarqube.org	+
	MIND	sourceforge.net/projects/mindyourdebt	+
	FindBugs	findbugs.sourceforge.net	+
Точка невозврата для ТД [8]	JCaliper	http://se.uom.gr/index.php/projects/jcaliper	+
Фреймворк ТД для дизайна [20]	inFusion	chocolatey.org/packages/infusion/	-
Схема оценки доли ТД [21]	Blaze monitoring tool	https://sites.google.com/site/blazedemosite/home/abou	-
Приоритизация ТД [24]	Tracy	-	-
Автоидентификация, мониторинг и контроль ТД [25]	VisminerTD	https://visminer.github.io	-
Инструмент для управления ТД [26]	DeepSourse	https://deepsource.io	+
Инструмент для предотвращения в основном невидимого технического долга [27]	Debtgrep	-	-
Инструмент для автоматического определения архитектурных запахов для C / C ++ [28]	Arcan too	essere.disco.unimib.it/wiki/arcan	+
Инструмент вычисляет наличие набора кода запахов и вычисляет индекс интенсивности [29]	JCodeOdor	essere.disco.unimib.it/jcodeodor	+

Окончание табл. 2

Метод [Источник]	Инструмент	Ссылка на инструмент	Открытый доступ
Инструмент для обнаружения запахов кода из кода Java и определения приоритетов технического долга на основе запахов [29]	JSpiRIT	-	-
Инструмент статического анализа кода для конкретной предметной области [30]	PLC software		-
Инструмент для тактического планирования ТД при использовании гибких методологий [9]	ProDebt	-	-
Среда программирования [31]	EXA2PRO		-
Метрики модульности для АТД [22]	TortoiseSVN	tortoisesvn.net/	+
LOC и Fan-In для количественной оценки СПАТД [19]	Understand JDeodoran	scitools.com/github.com/tsantalis/ JDeodoran	- +
Обнаружение и кол. измерение СПАТД [23]	SLOCCoun	dwheeler.com/sloccount/sloccount.html	+

Вопрос 3: существуют ли эмпирические исследования, способные продемонстрировать полезность идентифицированных методов? В [32] оценили три метода ([12; 14; 20]), чтобы выяснить, эффективно ли они описывают взаимосвязь между качеством системы и уровнем ТД. Изуриета и др. [33] используют Нугрохо и др. [15] для примера методологии. Модель на основе бенчмаркинга Майра и других [16] тесно связана с их более ранней работой по оценке качества, ориентированной на бенчмаркинг. Также она рассчитывает стоимость устранения недостатков способом, аналогичным подходу CAST [14]. Релевантные метрики структуры кода в структуре для оценки интереса к ТД [21] были выбраны таким образом, чтобы связать сопровождаемость и ТД в [15]. Как и в предыдущей работе, используются статические метрики кода. [34] провели эмпирическое исследование на 21 известном и развитом проекте с открытым исходным кодом, чтобы подтвердить гипотезу об ошибочности применения SonarQube. [35] выбрали четыре различных метода идентификации ТД (запахи кода, проблемы автоматического статического анализа (ASA), накопление кода и нарушения модульности) и применили их к 13 версиям программного обеспечения с открытым кодом Apache Hadoop проекта. Результаты показали, что различные методы ТД слабо связаны между собой и поэтому указывают на проблемы в разных местах исходного кода. Более того, их индикаторы заинтересованности (изменения и дефектность) коррелируют только с небольшим подмножеством индикаторов ТД. В [36] был проведен обзор эмпирических исследований по возникшей теме SATD после 2014 г. и до составления данного обзора в июле 2018 года. Они собрали инструменты и наборы данных, которые могут быть использованы в качестве основы для привлечения и содействия представлению новых и усовершенствованных подходов для управления и в конечном счете погашения САТД. Одновременно авторы отметили отсутствие исследований, посвященных погашению и управлению САТД, что чрезвычайно важно.

Вопрос 4: существуют ли эмпирические исследования, способные продемонстрировать полезность выявленных инструментов? В [37] ТД измерялся с помощью двух инструментов статического анализа кода (Findbugs и SonarQube). Целью было оценить, имеет ли код, созданный с помощью подхода Test Driven Development, более низкий ТД, чем код, созданный с использованием других методов. Эти два инструмента широко используются в сообществе для измерения ТД. Другие исследования тестировали SonarQube: [38] использовали его для измерения ТД в системе слежения за частицами; [39] использовал его для нескольких расчетов ТД в цепочке поставок программного обеспечения; [40] описывает тематическое исследование Ericsson, где они наблюдали за ТД, чтобы использовать их для создания системы оценки на основе стандарта ISO 15939:2007.

Заключение

Технический Долг – это широко используемое популярное выражение. Однако довольно сложно иметь четкое представление о доступных подходах и инструментах

из-за большого количества материала. Цель данной работы – предоставить исследователям и практикам обзор состояния дел в области ТД с упором на автоматизированные подходы. Согласно обзору, эта область исследований является новой и активно развивающейся, но все еще незрелой. Постоянно появляются новые подходы и инструменты, которые не основаны на результатах предыдущих исследований, а исследователи сосредоточены на проверке собственных подходов без каких-либо независимых оценок. Более того, такие проверки часто невоспроизводимы из-за использования собственных наборов данных. Поэтому необходимы дополнительные усилия для определения подходов с перекрестной проверкой и четкие указания на их применимость. Это очень важно, особенно для практиков, поскольку им трудно определить, какие модели следует применять в их конкретном контексте. В исследовании также отмечается, что в тех случаях, когда имеются инструменты для поддержки некоторых специфических подходов, они часто тяжелы в использовании, требуют сложной настройки и обеспечивают ограниченную поддержку широкого спектра языков программирования, используемых в реальных проектах. Более того, большинство доступных инструментов не способны измерить или оценить общий ТД. Они обычно сосредоточены на затратах на устранение последствий и не принимают во внимание смежные интересы (часто называемые не восстановительными затратами), которые часто очень важны для планирования процесса разработки и для контролирования долга на протяжении всего жизненного цикла продукта.

Список литературы

1. Barry Boehm, Paul Grünbacher, and Robert O. Briggs. Developing Groupware for Requirements Negotiation: Lessons Learned. IEEE Computer Society Press. 2001. Vol. 18. No. 3. P. 46–55.

2. Coman I. D., Sillitti A., Succi G. Investigating the usefulness of pair-programming in a mature agile team. 9th International Conference on extreme Programming and Agile Processes in Software Engineering (XP2008). 2008. P. 127–136.

3. Corral L., Sillitti A., Succi G. Software development processes for mobile systems: Is agile really taking over the business? 1st International Workshop on Mobile-Enabled Systems (MOBS 2013) at ICSE. 2013. P. 19–24.

4. Cunningham W. The wydahs portfolio management system, addendum to the proceedings on object-oriented programming systems, languages, and applications (addendum). OOPSLA. 1992.

5. Tom E., Aurum A., Vidgen R. An exploration of technical debt. Journal of Systems and Software. 2013. Vol. 86. No. 3. P. 1498–1516.

6. Lenarduzzi V., Sillitti A., Taibi D. Analyzing forty years of software maintenance models. 39th International Conference on Software Engineering. 2017. P. 146–148.

7. Falessi D., Shaw M., Shull F., Mullen K., Stein M.K. Practical considerations, challenges, and requirements of tool-support for managing technical debt. 2013 4th International Workshop on Managing Technical Debt. 2013. P. 16–19.

8. Chatzigeorgiou A., Ampatzoglou A., Ampatzoglou A., Amanatidis T. Estimating the breaking point for technical debt. 7th International Workshop on Managing Technical Debt (MTD). 2015. P. 53–56.

9. Ciolkowski M., Guzman L., Trendowicz A., Salfner F. Lessons learned from the prodebt research project on planning technical debt strategically. PROFES. 2017. P. 523–534.

10. Khomyakov I., Makhmutov Z., Mirgalimova R., Sillitti A. Automated measurement of technical debt: A systematic literature review. 21st International Conference on Enterprise Information Systems. 2019.

11. Kitchenham B., Charters S. Guidelines for performing systematic literature reviews in software engineering (version 2.3). Technical report, Keele University and University of Durham. 2007.

12. Letouzey.J. The SQALE method for evaluating technical debt. In Managing Technical Debt (MTD). 2012 Third International Workshop on Managing Technical Debt (MTD). 2012. P. 31–36.

13. Letouzey. J., Ilkiewicz M. Managing technical debt with the SQALE method. IEEE software. 2012. P. 44–51.

14. Curtis B., Sappidi J., Szynkarski A. Estimating the size, cost, and types of technical debt. Proceedings of the Third International Workshop on Managing Technical Debt. 2012. P. 49–53.

15. Nugroho A., Visser J., Kuipers T. An empirical model of technical debt and interest. Proceedings of the 2nd Workshop on Managing Technical Debt (ACM). 2011. P. 1–8.

16. Mayr A., Pl ¨osch R., K ¨orner C. A benchmarking-based model for technical debt calculation. In Quality Software (QSIC). 2014. P. 305–314.

17. Ploesch R., Gruber H, Pomberger G., Saft M., Schiffer S. Tool support for expert -centered code assessments. 1st International Conference on Software Testing, Verification, and Validation. 2008. P. 258–267.

18. Skourletopoulos G., Mavromoustakis C.X., Mastorakis G., Rodrigues J.J., Chatzimisios P., Batalla J.M. A fluctuation-based modelling approach to quantification of the technical debt on mobile cloud-based service level IEEE Globecom Workshops (GC Wkshps). 2015. P. 1–6.

19. Kamei Y., Maldonado E.D., Shihab E., Ubayashi N. Using analytics to quantify interest of self-admitted technical debt. QuASoQ/TDA@ APSEC. 2016. P. 68–71.

20. Marinescu R. Assessing technical debt by identifying design flaws in software systems. IBM Journal of Research and Development. 2012. Vol. 56. P. 9.

21. Singh V., Snipes W., Kraft N. A framework for estimating interest on technical debt by monitoring developer activity related to code comprehension. Sixth International Workshop on Managing Technical Debt (MTD). 2014. P. 27–30.

22. Li Z., Liang P., Avgeriou P., Guelfi N., Ampatzoglou A. An empirical investigation of modularity metrics for indicating architectural technical debt. Proceedings of the 10th international ACM Sigsoft conference on Quality of software architectures QoSA. 2014. P. 119–128.

23. Maldonado E., Shihab E. Detecting and quantifying different types of self-admitted technical debt. IEEE 7th International Workshop on Managing Technical Debt (MTD). 2015. P. 9–15.

24. Almeida R.R., Treude C., Kulesza. U. A business-driven technical debt prioritization framework. 35th International Conference on Software Maintenance and Evolution (ICSME'19). 2019. P. 181–185.

25. Mendes T.S., Gomes F.G.S., Goncalves D., Mendonca M.G., Novais R.L., Spinola. R.O. VisminerTD: a tool for automatic identification and interactive monitoring of the evolution of technical debt items. Journal of the Brazilian Computer Society. 2018. Vol. 25. P. 1–28.

26. Parthiban D.G. Examination of tools for managing different dimensions of technical debt. CoRR, abs/1904.11062. 2019. 20 p.

27. Arvedahl S. Introducing debt grep, a tool for fighting technical debt in base station software. International Conference on Technical Debt. 2018. P. 51–52.

28. Biaggi A., Fontana A., Roveda R. An architectural smells detection tool for c and c++ projects. 44th Euro micro-Conference on Software Engineering and Advanced Applications (SEAA). 2018. P. 417–420.

29. Lenarduzzi V., Martini A., Taibi D., Tamburri D.A. Towards surgically-precise technical debt estimation: Early results and research roadmap. In Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation, MaLTeSQuE2019. 2019. P. 37–42.

30. Bougouffa S., Dong Q.H., Diehm S., Gemein F., Vogel-Heuser B. Technical debt indication in plc code for automated production systems: Introducing a domain specific static code analysis tool. IFAC-PapersOnLine. 2018. Vol. 51 P. 70–75.

31. Soudris D., Papadopoulos L., Kessler C.W., Kehagias D.D., Papadopoulos A.I., Seferlis P., Chatzigeorgiou A., Ampatzoglou A., Thibault S., Namyst R., Pleiter D., Gaydadjiev G., Becker T., Haefele M. Exa2pro programming environment: Architecture and applications Proceedings of the 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation. 2018. P. 202–209.

32. Griffith I., Reimanis D., Izurieta C., Codabux Z., Deo A., Williams B. The correspondence between software quality models and technical debt estimation approaches. Sixth International Workshop on Managing Technical Debt (MTD). 2014. P. 19–26.

33. Izurieta C., Griffith I., Reimanis D., Luhr R. On the uncertainty of technical debt measurements. Information Science and Applications (ICISA). 2013. P. 1–4.

34. Lenarduzzi V., Saarim ̈aki N., Taibi D. The technical debt dataset. International Conference on Predictive Models and Data Analytics in Software Engineering, PROMISE'19. 2019. P. 2–11.

35. Zazworka N., Vetro' A., Izurieta C., Wong S., Cai Y., Seaman C., Shull F. Comparing four approaches for technical debt identification. Software Quality Journal.2014. Vol. 22. P. 403–426.

36. Sierra G., Shihab E., Kamei Y. A survey of self-admitted technical debt. Journal of Systems and Software. 2019. Vol. 152. P. 70–82.

37. Parodi E., Matalonga S., Macchi D., ınSolari M. Comparing technical debt in student exercises using test driven development, test last and ad hoc programming. Computing Conference (CLEI). 2016. P. 1–10.

38. Luhr L.R. The application of technical debt mitigation techniques to a multidisciplinary software project. PhD thesis, Montana State University-Bozeman, College of Engineering. 2015. 98 p.

39. Monteith J., McGregor J. Exploring software supply chains from a technical debt perspective. 4th International Workshop on Managing Technical Debt. 2013. P. 32–38.

40. Britsman E.,¨Tanriverdi O. Identifying technical debt impact on maintenance effort-an industrial case study. 2015. 70 p.