UX Debt in an Agile Development Process: Evidence and Characterization

Andres Rodriguez (arodrig@lifia.info.unlp.edu.ar)
LIFIA, Univ. Nac. de La Plata
Juan Cruz Gardey
CONICET
Julián Grigera
CICPBA
Gustavo Rossi
LIFIA, Univ. Nac. de La Plata
Alejandra Garrido
CONICET

Research Article

Keywords: technical debt, user experience, UX smells, refactoring, user testing, A/B testing

Posted Date: October 27th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-2190539/v1

Abstract

The metaphor of Technical Debt (TD) has generated a conceptual framework on factors that weaken the quality of software and accumulate a repair cost. However, user-related aspects like user experience (UX) receive little consideration among TD types, for reasons like the substantial focus on code TD, some dynamics inherent to agile processes, and an apparent lack of cumulative cost over time. This article has two main goals: first, to present evidence of the existence of UXDebt as a type of TD, with a cumulative cost for the development team as well as stakeholders; second, to propose a definition and characterization of UXDebt that may serve as a frame for further research on methods and tools for continuous management within agile processes. For the first goal, we have compiled evidence on the current state of UXDebt from three sources: a literature review, a survey among software engineering professionals in agile teams, and the analysis of UX issues in GitHub. All sources have evidenced some form of UXDebt; surveyed practitioners have recognized its poor management with a cost for the entire team that accumulates over time. Moreover, issue-tracking systems allow to visualize and measure a technical form of UXDebt. For the second goal, we have defined a conceptual model that characterizes UXDebt in terms of both technical and non-technical aspects. On the technical side, we propose the notion of UX smells which allows us to discuss concrete management activities.

1. Introduction

The metaphor of Technical Debt (TD) in software was proposed by Ward Cunningham in 1992 (Cunningham, 1992) as a way to visualize and communicate the harmful consequences of quality compromises in software development, and the need for continuous refactoring. Since then, it has been a popular topic in research studies, which have mainly focused on code-related TD and different ways to calculate it (Ciolkowski et al., 2021; Tsoukalas et al., 2018). Agile development is particularly drawn to accumulate TD as short development cycles often call for design shortcuts in favor of immediate delivery (Behutiye et al., 2017). Either if TD is accrued deliberately or inadvertently, it is known to create a cost estimated at around 23% extra working time (Besker et al., 2019). This cost affects the development team in its ability to reach quality standards in a timely fashion (Li et al., 2015), as well as the team's morale, productivity (Besker et al., 2019) and project risk (Tom et al., 2013). Moreover, the cost of TD includes both principal and interest, where the *principal* is defined as the effective cost of TD remediation, while the *interest* is the additional effort needed in the future to maintain software with accrued TD (Tsoukalas et al., 2018; Curtis et al., 2012).

The concept of TD has grown throughout these decades and gave rise to a conceptual framework focused on several factors that influence the quality of software (Li et al., 2015). These factors include not only internal quality problems with code, but also other types of software flaws related to "requirements TD", "architectural TD", "design TD", "documentation TD", etc. (Li et al., 2015). Researchers have extended the metaphor to higher-level design issues (domain level technical debt) (Störrle and Ciolkowski, 2019), the impact of social relationships in the work team (Social Debt) (Tamburri et al., 2013), or the proper use of methodological approaches (Process Debt) (Martini et al., 2020).

Noticeably, there are only a few research works that recognize user experience (UX) in relation to TD. UX has been defined as “user’s perceptions and responses that result from the use and/or anticipated use of a system, product or service” (ISO, 2019). This ISO definition additionally expresses that UX is a consequence of both instrumental aspects in relation to functionality, interactive behavior, and usability, and hedonic aspects like user’s emotions, attitude, and comfort. The few studies that mention the concept of “usability debt”, “UX debt” or some stakeholder perspective ((Theodoropoulos et al., 2011; Zazworka et al., 2013; da Fonseca Lage et al., 2019; Baltes and Dashuber, 2021)) consider it as debt with end users, but with no apparent cost for the development team, at least in terms of interest. This narrow perspective eludes the responsibility of the whole team towards the designed system or product (Gothelf and Seiden, 2021). UX is an essential aspect of a product, determining its quality as well as its success and durability in the future, as much as a solid architecture or decent maintainability level (Djamasbi et al., 2014; Erdös, 2019). As such, poor UX quality, like poor code quality, creates a cost that negatively affects the whole team and its productivity, since it hinders its ability to deliver high-quality customer value without time waste (Besker et al., 2019). This is in line with current development methods like Lean UX (Gothelf and Seiden, 2021), which make all team members aware of UX issues and decisions.

Currently, both white and gray literature indicate that developers pay little attention to UX issues after the initial design stage (Kuusinen, 2016; Chan,2017). This implies that, in a context where agile methodologies are becoming an industry standard, the TD related to UX (UXDebt) is not being dealt with as a topic that requires constant and systematic monitoring during development and maintenance. We believe that incorporating UXDebt as a type of TD will serve in making the agile team and stakeholders aware of the importance of detecting, measuring, and reducing its cost.

Besides reviewing current literature, our intention in this work is to collect empirical evidence of the existence of UXDebt as a type of TD and understand the degree to which agile teams recognize it and deal with it. Moreover, we are interested not only in the processes involved but also in the fitness of current tools for code management to monitor UXDebt, calculate its cost, and prioritize it. For these purposes, we have performed two experiments: a survey with practitioners and the mining of open-source project repositories in GitHub as an issue-tracking system (ITS). With these experiments, we intend to answer the following research questions, respectively:

RQ 1.0: How is UXDebt identified and managed by agile development teams?

RQ 2.0: What information about UXDebt can current ITSs provide?

Our findings provide evidence of the existence of UXDebt as a type of TD that has a cost with both principal and interest components, not only for end users but also for the development team. We also found that UXDebt may occur at any stage of software development and may have technical and non-technical effects. Our contributions include both a conceptual model that characterizes the UXDebt, including symptoms and factors that affect technical constructs and stakeholders and a definition of UXDebt. On the technical side, we found that ITSs may be used to track the UXDebt that appears after the production stage, in the form of issues that weaken the quality of the user interaction and may impact the

future development of user interface (UI) code. To identify this type of UXDebt we propose the concept of UX smells, which are hints of poor interface and interaction design that may be solved by applying UX refactoring (Gardey et al., 2020; Grigera et al., 2017). In turn, UX refactorings have been defined as changes to the navigation, presentation, or interaction of a web application that preserve functionality (Garrido et al., 2017, 2011). Having concrete solutions in terms of UX refactorings makes UX smells an operational concept that may serve to identify, measure, and prioritize a technical form of UXDebt that appears in ITSs. Thus, like code smells are considered as relevant sources of design TD (Li et al., 2015; Fontana et al., 2012), we propose UX smells to hint sources of UXDebt.

The rest of the article is structured as follows: Section 2 presents the background and related work. The following two sections present empirical evidence about UXDebt as referred to and managed by practitioners from industry (Section 3) and UXDebt in open-source projects (Section 4). Section 5 presents a Conceptual Model of UXDebt as emerged from the previous Sections. In Section 6 we discuss the model and what it implies for UXDebt management. Finally, Section 7 includes conclusions and future work.

2. Background And Related Work

This section provides background and discusses related research on some topics that will be covered in the article: TD and its components, the relationship between UX, User-Centered Design (UCD), agile development and TD, and UX-related debt.

2.1. Technical Debt

The accepted definition of TD is as follows: “a collection of design or implementation constructs that are expedient in the short term but set up a technical context that can make future changes more costly or impossible” (Avgeriou et al., 2016). That is, there is a cost associated with TD that is important to measure, so decision-makers know the system’s maintainability status in order to take action in a timely manner before it becomes unmanageable.

Cunningham introduced the TD metaphor to motivate refactoring as a key feature in any iterative development process (Cunningham, 1992). The steady growing body of research has focused on low-level TD (e.g. code, low-level design) over more abstract aspects (requirements, design, architecture) that were nevertheless identified in several studies (Ciolkowski et al., 2021; Li et al., 2015; Störrle and Ciolkowski, 2019; Alves et al., 2016; Rios et al., 2018). For example, domain debt (DD) denotes the disparity between an application and a domain, like an incorrect assumption about it (Störrle and Ciolkowski, 2019). It cannot be detected by analyzing code-level or architecture artifacts, but by domain knowledge data. Moreover, higher level artifacts are often incomplete, obsolete, or even missing. Therefore, higher-level TD can be more damaging than implementation debt, harder to detect and track, and highly iterative approaches further exacerbate the problem (Ciolkowski et al., 2021).

In some cases, TD may be accrued deliberately. This type of TD is also called “self-admitted TD” (SATD) (Potdar and Shihab, 2014), and it is introduced to obtain short-term business value. In other cases, TD may be introduced inadvertently, when applying poor design decisions usually under pressure.

Like the case of financial debt, the cost of TD has two components: principal and interest (Curtis et al., 2012). The principal is the amount of owed capital, which in the case of TD, is the cost of resolving TD items, for example, in man-hours. The interest in financial debt basically depends on time and interest rate, but in the case of TD, it is the future cost associated with the additional effort required to maintain unnecessarily complex code (Curtis et al., 2012). Thus, the interest in TD is one of the most difficult things to measure, because it depends on the expected future development, and it is not something that purely depends on time. Moreover, it has been argued that interest should include not only the cost of maintainability, but other costs such as opportunity costs, UX issues, and product value (Ciolkowski et al., 2021).

Identifying the presence of TD and its components, and managing this debt effectively is essential to maintain the desired level of software quality (Li et al., 2015). TD management activities include TD identification, measurement, repayment, monitoring, and prioritization. These activities require a reification of the metaphorical concept of TD, so as to get to concrete, actionable level (Ramirez Lahti et al., 2021). For this reason, several studies have analyzed the relation between software metrics and TD, and one of the most applied metrics is that of code smells (Fontana et al., 2012; Ramirez Lahti et al., 2021). Code smells represent possible sources of design problems that may lead to code decay, and as such represent an important source of TD (Fontana et al., 2012). Since several tools may calculate the number of smells in a code base (e.g., SonarQube[1]), and each smell has identifiable solutions in terms of refactorings, code smells offer an effective mechanism for practical management activities. In contrast, bugs or defects are generally not considered TD, since most of the quality flaws of TD raise from poor design or strategic decisions, and not from test failures (Curtis et al., 2012). We follow this same view in this article.

Other (non-technical) debts that also cause a negative impact on software have been identified in the literature, such as social and process debt. Social debt (SD) it is referred to as “the presence of sub-optimality in the development community, which causes a negative effect” (Tamburri et al., 2013). An example of SD may be having UX and Development teams without the necessary connection to develop mutual trust and work together. Process debt (PD) has been initially defined as “a sub-optimal activity or process that may have short-term benefits but generates a negative impact in the medium-long term” (Martini et al., 2020). For example, when UX designers only participate in daily meetings when the backlog includes UI user stories. In the short term, they can get more time for working on other projects. However, knowledge sharing and dependency resolution, which are more valuable in the medium to long term, are not achieved (Störrle and Ciolkowski, 2019).

2.2. UX, UCD, and agile methods in the context of TD

The notion of UX emerged several years ago to account for the multiple dimensions involved in the use of interactive systems (Hassenzahl and Tractinsky, 2006). This construct considers not only the pragmatic aspects of interaction (functionality, interactive behavior, user skills, context of use) but also the hedonic (brand image, presentation, internal state of the user resulting from previous experiences) (see (ISO, 2019), Note 2). Other definitions in the literature share this characterization for UX (Law et al., 2009; Hassenzahl et al., 2021).

UX design work has traditionally followed the UCD process, which includes iterations over a basic four-moment cycle: understanding and specifying the context of use, specifying interaction requirements, producing design solutions, and evaluating them against requirements (ISO, 2019). Three roles of UX experts can be identified throughout the entire cycle: UX designer (responsible for understanding users), interaction designer in charge of designing and evaluating user interaction with products or services (both at the prototype and product level), and UI developer, who generates the guidelines and details of the UIs and design guides (Silva et al., 2013). Often the UI developer is the link between the UX team and the software engineers. These UX experts contribute to different tasks and produce technical artifacts in a wide range of abstraction and complexity. There are early artifacts like Persona descriptions, storyboards, wireframes, and cognitive walkthrough reports (Cooper et al., 2015). As the build progresses, design guides appear, wireframes become high-fidelity prototypes, and systematic usability indicators are added to expert inspections. After each delivery, previous artifacts return to the scene for new functionality, and more accurate interaction data emerge from user tests and A/B tests (Kohavi et al., 2020). User tests, in which representative users follow a sequence of typical tasks, are costly but provide many measures of UX problems, like task completion rates, task time, satisfaction ratings and UX questionnaire results (Sauro and Lewis, 2016). Meanwhile, A/B tests have become a popular method for comparing alternative designs in terms of conversion rates, but do not provide a good measure of UX (Nielsen, 2005).

While both UCD and agile methods are driven by customer satisfaction, there are several mismatches between UCD and agile processes, which have been the focus of several studies (Da Silva et al., 2011; Brhel et al., 2015). Moreover, the role and responsibility of UX professionals within an agile team are not clearly defined, and neither is the timing of their responsibilities within an agile cycle (Bruun et al., 2018). Typical settings show a relevant role for UX experts in the early phases of a project and poor attention to their tasks afterward.

To help synchronize UX practices with agile development, lightweight methods are needed to evaluate UX as a part of iterative development (Da Silva et al., 2018). In previous work, we have proposed an integration method composed of two main mechanisms: A/B testing and refactoring for UX (Firmenich et al., 2019). A/B testing in the context of UX means that the winning version is decided based on higher values in UX metrics. Meanwhile, UX refactoring is defined as changes applied with the purpose of improving UX quality while preserving functionality (Garrido et al., 2017).

In turn, a UX smell hints at a problem with the navigation, presentation, interaction or any UX aspect that may be solved by applying UX refactoring (Grigera et al., 2017). An example of a UX smell is a free text

input that only accepts a small set of possible values (which has been cataloged as “Free input for limited values” (Grigera et al., 2017)); it may be solved by applying alternative UX refactorings like “Add Autocomplete” or “Turn Input into Select”. Another type of UX smells could be issues related to the style or aesthetics of a UI such as low color contrast, misaligned elements, and lack of responsiveness, among others. Note that UX smells are different from bugs in the UI since smells do not prevent users from accomplishing their goal but just made it cumbersome or uncomfortable. Moreover, while there are UX smells that signal concrete problems on a very specific UI element, there may be others that affect the whole UI and their solution may require multiple refactorings (like a UI redesign). Regardless of their magnitude, the key aspect of UX smells is that they are fixable by applying one or more UX refactorings.

2.3. UX Debt in the Literature

Only a few works in the academic literature specifically approach the user-related aspects of software as a type of TD. Also, being UX a recent construct, this research mostly speaks about usability or the “stakeholders’ perspective”. Gray literature had addressed in different opportunities the topic of UXDebt from the UX practitioner’s point of view, in general as a form of debt similar to TD that mainly affects end users (e.g. (Wright, 2013; Kaley,2018)). Recently, some of these approaches seem to be included for the analysis in academic research (e.g. (Baltes and Dashuber, 2021)).

Theodoropoulos et al propose including in the development team “a stakeholder perspective” that integrates the vision of internal and external stakeholders (business executives, technology leaders, risk managers, and end users; regulators, shareholders, and customers). They further explain that in order to incorporate the stakeholders’ goals, it is necessary to take a “value-based view” of the TD that considers quality to be whatever the customer is willing to pay for, and one of the attributes of this definition of quality is usability. Thus, poorly implemented features affecting usability force users to create workarounds that generate TD (Theodoropoulos et al., 2011).

Zazworka et al. use the term “usability debt” in a case study comparing TD as identified by developers vs. the one automatically identified by a tool (Zazworka et al., 2013). A developer of that case study mentioned the term to refer to the lack of a common interface template. This only emphasizes that developers need to be able to characterize and measure this type of debt.

Kuusinen talks about “allowing technical debt in UX design” so that a cross-functional agile team may work incrementally on the UX design in each iteration. She further defines UX design debt as “not quite right design which we postpone making it right” until further information becomes available (Kuusinen, 2016). However, the author does not provide any further characterization, and as it is, this definition does not seem to conform to the agreed definition of TD as the constructs that “can turn future changes more costly” (Avgeriou et al., 2016), i.e., it does not seem to carry any debt interest.

The work of Fonseca et al. is the only one that explicitly addresses some management activities for usability TD as the identification and estimation of its remediation cost (da Fonseca Lage et al., 2019). However, the paper does not define or fully characterize it but adheres to the definition sketched in a

tertiary study as “inappropriate usability decisions that will need to be adjusted later”, which suffers from the same problems as the definition in the previous paragraph. With a survey of open issues in the backlog of 5 projects, the paper identifies those issues that, according to the Project Manager, can be compatible with usability problems and estimates the cost of remedying each one in man-hours. To facilitate prioritization in the TD management strategy, it also associates each issue identified as a TD item with one of the 10 heuristics proposed by Nielsen for expert usability assessment (Nielsen, 2020).

Some secondary and tertiary studies on TD (Li et al., 2015; Rios et al.,

2018) criticize that the quality attributes (QA) compromised by incurring TD did not comply with a uniform quality model. All the extracted QAs are about software product quality, and none is about quality in use (ISO, 2011).

Xavier et al. mined issue tracker systems to find SATD (Xavier et al., 2020). They found that with a 10.1% of occurrence, UI debt is the second most common type of SATD-related issue. In this case, developers implemented shortcuts that result in usability flaws (Xavier et al., 2020).

[1] https://www.sonarqube.org/

3. A Practitioners Survey

This section explores how development teams understand UXDebt. We conducted an online survey with practitioners from companies that deliver software with strong UX requirements. The sample includes companies developing and delivering for third parties, for internal use, or software as a product/service.

This qualitative study has three sub-goals: *(i)* to learn about the familiarity of agile development teams with the existence of UXDebt; *(ii)* to analyze how they approach the management of what they understand by UXDebt (what are the activities or products most likely to generate it, how much time is spent on its analysis and repayment, what processes and tools are used to trace it); and *(iii)* to know if there are significant differences between the way UXDebt is understood and managed by UX experts as compared to software engineers.

These goals provide the rationale for the survey research question and related sub-questions:

RQ 1.0 How is UXDebt identified and managed by agile development teams?

This main RQ is broken down into the following sub-questions, which represent areas of concern:

- **RQ 1.1**: To what extent are practitioners familiar with the concept of UXDebt?
- **RQ 1.2**: What kind of UX tasks and products generate the most negative impact on UXDebt?
- **RQ 1.3**: How much work time is wasted because of UXDebt?
- **RQ 1.4**: How much is the team aware of the acquired UXDebt?

- **RQ 1.5**: What is the difference between the collective management of UXDebt and individual (UX-related subgroup) management?
- **RQ 1.6**: In what way does the age of the software system affect the precedent questions?
- **RQ 1.7**: In what way are the different roles in the team affected by the precedent questions?
- **RQ 1.8**: How is UXDebt tracked? What tools do practitioners use?

In the remainder of the section, we report our methodology, the collected results, and finally a discussion on contributions.

3.1. Methodology

3.1.1. Data collection

The survey was designed and hosted with Google Forms and included 27 questions. We ran a first draft with 2 practitioners (a developer, and a UX researcher) and 2 authors, and checked for unusual terms or expressions (Blair et al., 2013). The survey was written in Spanish, as it was targeted at Argentinian-based software teams.

We used convenience sampling (Wohlin et al., 2012) to select the participants. We invited three groups of people: practitioners in our networks who work in different-sized companies, members of Argentina's IXDA Chapter (Interaction Design Association), and companies in the regional IT Hub (which brings together more than 100 IT companies of different sizes, domains, and levels of internationalization).

The form remained open from mid-September to end-October 2021. Participation was anonymous and voluntary (no compensation was given to respondents). The survey was structured in 5 sections: the project and the company, the methodology, the definition of UXD, UXDebt examples, and UXDebt management. The full questionnaire is described in Table 1, translated from the original in Spanish.

For a particular question in Section 3, we inquired about knowledge of UXDebt and its definition (question 13). This required a definition of UXDebt, so we used the following one, taken from the Nielsen/Norman Group website, which is well-known among UX practitioners:

"UX Debt is similar to its technical counterpart, known as Technical Debt (TD). TD refers to the additional costs of time and effort that result from launching faster or easier technical solutions, rather than launching the best approach. It implies that the cost of having to go back and solve problems after launch is always higher than launching ideal solutions in the first place (the debt is paid with usurious interest)" (Kaley,2018).

3.1.2. Data analysis

The survey received a total of 56 responses (and 3 more that were incomplete). The answers to open questions were analyzed with the prior consensus of the authors.

We aimed at having a balanced proportion of both, UX-related roles (UX Researcher, UX/UI Designer, UX Manager or Consultor) and software engineer roles (Manager, Software architect, Tester, Analyst). The final sample satisfies this by including 37% of UX-roles, and 63% of non-UX-roles. Approximately 38% of the UX roles and 74% of the software engineering roles had more than 5 years of experience. Participants work for large (43%) and medium-sized organizations (32%).

After validation, we performed a quantitative and qualitative analysis of the data collected. In order to apply frequency analysis, we transformed categorical data to a Likert scale (1-5), where “Never” or “Fully disagree” were mapped to 1, and “Very frequently” or “Fully agree” to 5.

To address each RQ, quantitative analysis was performed with different data cuts. First, we considered the set of responses to draw conclusions common to all roles, team sizes, and systems age. Then, we filtered general results by system age (grouping the responses in three ranges: <2 years, 2-5 years, and >5 years), by roles (into two subsets: roles directly related to UX -UX Researcher, UX Designer, UX Manager- and other software engineering roles in the team -Analyst, Tester, Developer, Technical Leader). Whenever we needed to assess the independence between the crossover series, we performed cross-tabulation chi-square tests.

Table 1: Complete Survey for practitioners

Section	N	Question	Type
1. Project	1	What is the size of your organization	closed
&	2	Where is your organization based?	Closed
Organization	3	What is the size of your current project?	Closed
	4	How many people work on your project?	Closed
	5	What is the age of your current project?	Closed
	6	What is your role in the project?	Closed
2. Methodology	7	Which methodology better describes your project?	Closed
	8	What is your experience with such methodology?	Closed
	9	What is your software project's category? (App kind)	Closed
	10	What is your background? (i.e. studies)	Closed
3. UX Debt	11	Do you know UXDebt to be a kind of technical debt?	Closed
	12	How would you define UXDebt?	Open
	13	How much do you agree with this UXDebt definition?	Closed
	14	If you disagree, which part do you disagree with the most?	Closed
4. UXDebt examples	15	Provide a UXDebt example that impacted your project	Open
	16	Why is this a UXDebt instead of another kind of issue?	Open
	17	How representative do you think this example is?	Closed
	18	Which issues generate UXDebt in your project?	Closed
5. UX management	19	Are you aware of the identified UXDebt in the system?	Closed
	20	All team members are aware of UXDebt in the system	Agree
	21	You monitor UXDebt in the system	Agree
	22	Who participates in UXDebt monitoring within the team?	Closed
	23	What measures are taken to counter UXDebt?	Closed
	24	What tools do you use to keep track of UXDebt	Open
	25	What % of your time is dedicated to UXDebt?	Range
	26	What % of your time is dedicated to identifying UXDebt?	Range
	27	What % of your time is dedicated to solving UXDebt?	Range

For the qualitative analysis of the open questions in sections 3, 4, and 5, we coded responses with the following schema:

• Definitions and examples of UXD. To validate the understanding of UXDebt as a type of TD, we used the following components taken from the agreed definition of TD in the literature (Avgeriou et al., 2016)):

- Whether it is a case of self-admitted debt or not
- Whether it is caused by decisions based on schedule, process, convenience
- Which development artifact it affects (code, test, documents)
- Whether it has an impact on cost, value, schedule, or quality
- If it particularly affects the UX characteristic of the system

• Tracking tools: among those answers that explicitly reported tools (38 out of 56), we translated them to English, eliminating those that lacked interest in terms of tracking tools. Then we manually applied the codification scheme in 4 categories ((Avgeriou et al., 2016)):

- documentation: tools for recording the UXDebt items (includes prototyping tools as case documentation)
- issues: using the same tools for any other issue tracking
- backlog: registering the UXDebt items as part of a general, or specific purpose backlog
- other: we open this category for teams using specific techniques or ceremonies as a tool for identifying and tracking UXDebt

3.2. Results

Many of the research questions were answered with quantitative analyses of the survey's questions. Figure 1 illustrates these results, discussed in Section 3.2.1. The remaining RQs are discussed in qualitative analysis in Section 3.2.2.

3.2.1. Quantitative Analysis

Regarding **RQ 1.1**, 62% of the participants declared to have some knowledge of the concept of UXDebt as a type of TD (See Figure 1.a). We complement with qualitative analysis on RQ 1.1 in Section 3.2.2. Regarding **RQ 1.2**, the problems that are most associated with an impact on UXDebt are the lack or low quality of requirements, testing, and interaction design. Figure 1.b shows the percentages, along with the answers discriminated by role, i.e., UX/non-UX.

When evaluating the work time allocated to UXDebt management (**RQ 1.3**), the participants indicated, on average, 32.79% of their work time, as shown in Figure 1.c with an activity breakdown. We did a study of frequencies in 10% intervals, and we found that although some participants indicated up to 60% of their time in some UXDebt management activity, the bulk is between 0 and 20% for any of the three activities (identification, quantification, remediation).

Research question **RQ 1.4** was answered by two questions: the first was about the individual awareness of UXDebt, where 18% responded that they are unaware of the UXDebt of their project (Fig. 1.d). The second question asked about team awareness, where 57% answered that only some members are aware (Fig. 1.e).

In terms of UXDebt management (**RQ 1.5**), we inquired about the periodicity of UXDebt monitoring carried out by each participant. Similarly to RQ 1.4, we questioned to what extent each professional considers that the rest of the team participates in the monitoring activities. Results are summarized in 1.f: only 12% admitted that they do it regularly or often, and most think that some other members do this task (55%).

3.2.2. Qualitative Analysis

Familiarity with UXDebt (RQ 1.1): When presented with a general definition of UXDebt, 4% disagreed with it, achieving almost 30% of total agreement. When asked about which part of the definition they disagreed with the most, only 5% indicated that they did not agree with it requiring a greater effort or cost caused by the deferral of optimal solutions (interest earned by the UXDebt). We also coded 37 definitions provided by the participants and found that more than 70% recognized that UXDebt usually originated from explicit decisions. In addition, 32% consider that UXDebt imposes an extra cost on development tasks that are deferred or performed suboptimally.

The examples of UXDebt items that participants gave reinforce this familiarity with the core concepts and components of UXDebt as a type of TD. A few of these examples were:

"... we developed a modal notice of promotions with so much cognitive load that led users to access the promotion (usually users DO NOT READ long texts). (After that, users) called Support to cancel (the promotion). This implied a tremendous cost in staff to attend to those requests in addition to doing the rollback in the activation of the promos..." (UX expert)

"... we made a button to upload documents without considering batch loading (...) the UI did not contemplate the problems that could be presented with multiple uploads). The solution only considered solving the persistence of the files in the remote DB, but it did not matter if that made the load slower or buggy for the user. (After delivery) problems emerged and reported loading errors (3 files were uploaded, 1 failed and everything was canceled, and the user did not realize what the problem was)" (Software engineer)

"(in the mobile version) the size of the legal notices text box covered the call to action (...). Many conversions were lost because of that". (UX expert

"... a financial data listing with many columns was required. We had two options: export the data to an Excel file or a scrollable table on the screen.

To speed up delivery, the table was chosen. The selection and order of the columns were decided from previous listings, without user validation. After delivery, users required for choosing the columns to

display. So, we added a form to filter by all possible fields. Users had to re-select filters over and over again to repeat queries, so they asked to be able for saving the selection. We fixed that. Finally, the users demanded the export to Excel files for working with the data besides consulting them...” (Software engineer)

System age and UXDebt (RQ 1.6): Taking into account the tension usually reported between the approach of an initial design in the field of UX and the dynamics of short and incremental deliveries in agile methodologies (Brhel et al., 2015), we wondered if the development time or the age of the system can have any impact on the previous answers, in particular on the causes that generate UXDebt and the time devoted to its management activities.

Regarding the causes of UXDebt (RQ 1.2), survey results are clear in that systems with less development suffer more from issues related to Requirements (60%) and Testing (64%) than any other factor. However, there are two aspects here that stand out. First, awkward social relationships maintain a high value as generators of UXDebt (36% of those who respond with recent systems and 32% of systems older than 5 years). Also, the lack or poor quality of refactoring of UI code reaches up to 40% in projects older than 5 years (well above the 26% assigned in younger projects).

The time spent on UXDebt management activities (RQ 1.3) appears to decrease with the age of the system in the three categories that we asked for: identify, quantify and remedy. The time spent identifying, and quantifying UXDebt goes from 13% in new systems to less than 4% in older ones. UXDebt remediation goes from 20% in systems less than two years old to less than 7% in systems older than 5 years. Contrarily, Besker et al. showed that, in general, TD forces to invest more time in management activities as the system ages (Besker et al., 2017). This different behavior between UXDebt and TD in relation to the system age could be related to the UX life cycle in which uncertainty decreases with the development of the system (due to in-depth knowledge of users, consolidation of design guidelines, etc.). It could also be related to other aspects of development, but further research is needed to verify any other hypotheses.

Individual awareness of UXDebt shows uneven behavior according to the age of the system. For instance, numbers indicate that little or no awareness grows with system age, while “total” or “much” awareness decreases. In terms of perceived team awareness, most participants feel they are not alone in perceiving the accrued UXDebt (with percentages much higher than the sum of the remaining categories).

If awareness of UXDebt decreases with the age of the system, the most obvious reason would be that monitoring activities also decrease, which survey results assert (up to 56% report no monitoring for systems older than 5 years). Even more, the proportion of individual monitoring for those involved decreases considerably (from 32% to a 11%).

Team roles and UXDebt (RQ 1.7): Regarding roles, i.e. “In what way are the different subgroups in the team (UX roles, non-UX roles) affected by the previous questions?”, results show that UX roles have a different view on the causes of UXDebt than non-UX roles. In particular, the difference in association with

the design of interactions, UX refactoring, and social interactions between both subgroups is notable. In some ways, it could be assumed that each subgroup assigns greater responsibility for UXDebt accrual to the other one (see Figure 2). For instance, UI design is a cause of UXDebt for around one-third of UX roles, but for almost half of the non-UX roles. Conversely, only 20% of non-UX roles believe that awkward social interactions are an inconvenience that generates UXD, against almost 60% of UX roles, which could reflect the tension that literature reflects between both groups (Maudet et al., 2017).

When asked about whether respondents disagreed with the provided UXDebt definition, 57% UX roles agreed, while 31% of non-UX roles did not. When asked about the reason they disagreed, 51% of non-UX participants doubted that fixing a sub-optimal solution was more costly than getting it right the first time, while the result was 29% in the case of UX related roles. Regarding UXDebt remediation actions, 53% of participants in UX roles consider that serious issues are not addressed in the current or next sprint, while most participants in non-UX roles consider the opposite. Perhaps this difference originates in the different decision hierarchies in the work team.

UXDebt tracking (RQ 1.8): Regarding RQ 1.8 “How is UXDebt tracked? What tools do practitioners use?”, most participants (55%) reported being using the same tools they use for regular TD management (e.g. Jira, Github, RedMine). We could observe that the teams are able to integrate UXDebt within the development cycle, just as they do with the rest of the artifacts. Regarding UX-specific tools, some reported using prototyping tools for documentation (26%). Other questions on the same line inquired about the participants’ perception on the degree to which UXDebt is tracked (questions 20 and 21). When asked whether they monitored UXDebt individually, the most frequent answers were “sometimes” (38%) and “never” (30%). This result was only marginally different by the role, even if UX roles added more “many times/regularly” answers (19% and 14% respectively), still had a comparable proportion of “sometimes/never” (33% and 19%) with non-UX roles (40% and 37%). When asked about the team members that participated in monitoring, roughly half of the participants (55%) answered that “some members” monitor UXD.

3.3. Lessons Learned

The results of the survey with practitioners provide an unprecedented report on the way software practitioners approach UXDebt within the software industry. The results may be summarized in two main lessons.

3.3.1. Lesson 1: UXDebt is a phenomenon recognized by practitioners

UXDebt as a debt related to quality in use and to all stakeholders beyond the technical team is a phenomenon recognized both in the literature (see Section 2) and by the practitioners in the survey. Most respondents expressed familiarity with it and a notable majority agreed with a preliminary definition that emphasizes both the technical aspects of software and the UX actually experienced by users (see answers to RQ 1.1). Consequently, a comprehensive characterization that includes both facets may be

needed. To achieve this, we need to review the effort involved in delivering good UX through a software system.

According to the survey, much of the existence of UXDebt is recognized by the technical team (see RQ 1.4). In other words, for practitioners, most of the UXDebt could be self-admitted. Since some prioritization of the UX attributes to be included in each cycle is inevitable in the UX-Agile integration, it is possible that this prioritization is one seed of the UXDebt. Indeed, the need to choose some UX attributes for the current increment, while postponing others, can hinder or conspire against achieving some key principles of good UX design, such as knowing users and their needs thoroughly and maintaining consistency throughout the system (Cooper et al., 2015). In the quest to keep pace, each cycle can become an opportunity to deliver one or more technical UX artifacts with sub-optimal solutions (e.g., a user journey map that is known to be incomplete but requires a research effort that is postponed, an information architecture with little user evaluation, or a form that needs micro-interactions but needs to be integrated as-is into the current version). At the same time, RQ 1.1 has shown that a meaningful number of practitioners disagree with the statement “the cost of solving problems after launch is always higher than launching ideal solutions from the start.” This could be related to the idea that sometimes a low-impact/low-interest UXDebt is tolerable, even desirable for the sake of closer benefits (such as a quick release of a new version) (Kuusinen, 2016). Alternatively, it may reflect the need for a better definition that makes it easier to distinguish self-admitted UXDebt from what are just the only viable design alternatives at hand.

An artifact or task that includes sub-optimal solutions, whether deliberately or not, is likely to generate sub-optimal experiences for users. When it does not have a significant impact on future technical work, it will be a debt incurred by the development team but paid primarily by users or other stakeholders, as it has been suggested elsewhere (Baltes and Dashuber, 2021; Zazworka et al., 2013). However, to the extent that all artifacts released by UX experts constitute part of the technical solution that must evolve throughout the software life cycle, we may say those sub-optimal UX artifacts “set a technical context that may make future changes more costly or impossible” (Avgeriou et al., 2016). Thus, the technical aspect of the UXDebt will have increased. In short, there may be a part of the UXDebt that is technical and a part that is not.

3.3.2. Lesson 2: UXDebt receives little attention and resources

The technical team does not devote many resources (or attention) to UXDebt in terms of tracking and repayment, as shown by RQ 1.5-1.8: there is a small percentage of team members assigned to UXDebt tracking; diminished UX improvement, awareness and tracking of UXDebt as the system ages; intense friction between different team roles.

This may be originated just from team priorities, but it may also suggest difficulties in carrying out integrated, comprehensive UXDebt management. Survey respondents stated that their organizations have not yet established practices for managing UXDebt (RQ 1.8), and practitioners long for methods and tools

to let them strategically plan, track, and repay complex debt, as they do with other kinds of TD (Avgeriou et al., 2016).

Sometimes, UXDebt is decided to be accrued for the sake of rapid release, like an example of our survey from a software engineer (answers to RQ 1.2-3): *"(...) Although discrepancies with the requested interface were noticed, due to lack of time the user tests were not done and it was published so that users could already use it(...)".* . Spending time and effort on refactoring activities usually means that less time can be spent on implementing new features. This is one of the main reasons why software companies do not always spend additional budget and effort on debt repayment, as they tend to focus on delivering user features. The answers in our survey on resource allocation to UXDebt identification, prioritization, and repayment can be explained by this drive to allocate the UX team to work on new features, rather than to perform evaluations and refactorings of released features. The above excerpt continued with: "*(...) they immediately started to upload requests to correct everything that was not defined by the client, but those issues never became a high priority in the backlog (...)*".

To maintain an adequate level of debt, a team could choose between to alternatives: to avoid acquiring new debt (prevention) or to continuously repay it. None of them is considered realistic (Ampatzoglou et al., 2015). In a complex scenario with multiple and often conflicting demands between the business goals, the value desired by users, and the actual capabilities of the technical team, managing an adequate level of debt is challenging. This is when management tasks related to the identification and prioritization of newly acquired debt become vital. Indeed, the balance between devoting developers' time, effort, and resources to the implementation of new features or to TD remediation activities, bug fixes, or other system improvements becomes central (Lenarduzzi et al., 2021).

4. Empirical Findings In Issue Tracking Systems

This section is intended to present empirical evidence of the presence of UXDebt in project repositories and the fitness of ITSs to represent UXDebt items. We selected GitHub because of its widespread use and its friendly API to query issues and their attributes.

Through this evaluation we were interested to answer the following research question:

- **RQ 2.0**: What information about UXDebt can current ITSs provide?

To operationalize this RQ, we break it down into three more specific questions:

- **RQ 2.1**: To what degree is UXDebt acknowledged by the community of ITS contributors?
- **RQ 2.2**: Are UX smells a significant indicator to measure UXDebt?
- **RQ 2.3**: How much attention is paid to solving potential UXDebt items?

The motivation behind RQ 2.1 is to find out whether UXDebt is recognized by practitioners in the GitHub community, and particularly if there are UXDebt items that are introduced explicitly as short-term

shortcuts. Moreover, finding a high number of self-admitted UXDebt items would mean that developers are paying attention to UXDebt. Beyond that, RQ 2.2 is intended to find other instances of UXDebt items that were not self-admitted, probably arising from post-production evaluations. For this purpose, we propose to use the concept of UX smells. The rationale behind this proposal is that the concept of UX smells has been defined to mirror code smells in the UX world, that is, they represent design or implementation constructs that, while they are not bugs in the UI, weaken the quality of the user interaction. Thus, since code smells are generally used to automatically measure TD in IDEs or code repositories (Fontana et al., 2012; Ramirez Lahti et al., 2021) , finding a significant number of UX smells could similarly be used to estimate UXDebt. RQ 2.3 has the purpose of analyzing, for this potential UXDebt item characterized as UX smells, how much repair cost they represent and how much has already been repaid.

To answer RQ 2.1, we queried all GitHub projects to see if the concept of UX-related debt appears among GitHub contributors. In the case of RQ 2.2 and RQ 2.3, we evaluated 7 GitHub projects automatically and manually. The projects were selected because of their significant UI aspects, their popularity (with a minimum of 10k stars), and their diversity in terms of size (between 2458 and 122985 reported issues). The selected projects and their basic stats are reported in Table 2. We next describe the experimental design and results for each RQ.

4.1. RQ 2.1: To what degree is UXDebt acknowledged by the community?

SATD is a particular case of TD where developers explicitly admit their sub-optimal implementation decisions (Potdar and Shihab, 2014). Xavier et al. proposed to identify SATD in ITSs by looking for issues labeled with terms such as “technical debt” or “debt” (Xavier et al., 2020). Following the same idea, we searched labeled issues in GitHub to assess if there is selfadmitted UXD. In our case we used the following label combinations: “ux debt”, “ux”+“technical debt”, “ux”+“debt”, “usability”+“technical debt” and “user experience”+“technical debt”. We found 257 issues in total from different projects. They were manually analyzed to assess if they really correspond to UX improvements that were postponed. As a result, 54 were ruled out, including bug reports, which we do not consider as debt, and questions, leaving a final number of 203 issues. Some examples of the issues are:

Improve user messages.*We have several places, where the messages can be improved. I suppose, we can add comments to this issue if we will find something related during usage.* (eclipse/che-che4z-explorer-for-endevor).

Only show “Uncategorized” when not empty.*Currently, the “Uncategorized” category is always displayed, even when there are no posts that are missing a category. If the count of Uncategorized posts is zero, it shouldn’t be displayed.* (paragonie/airship).

Link preview text on hover is inconsistent.*Our link preview text*

on hover are inconsistent across various views, we should work to make all of them consistent. (microsoft/vscode).

The issues found show evidence that at least an incipient set of GitHub contributors and collaborators recognize the existence of UXDebt and identify issues as contributing to the UXD.

4.2. RQ 2.2: Are UX smells a significant indicator to measure UXDebt?

Since the amount of self-admitted UXDebt items is not outstanding, we decided to evaluate the existence of other issues which, although not labeled as UXDebt items, may contribute to the occurrence of this type of debt.

The process of analyzing the presence of UXDebt items among the issues of each project required some preliminary definitions of *(i)* what we consider as a UX-related issue and *(ii)* in which cases we consider that a UX-related issue may potentially contribute to UXD.

In order to identify all the issues related to UX aspects, we started with an initial set of terms and iteratively refine them at least 5 times through an automatic search using the GitHub API, followed by a manual check on a randomly selected subset of results by two of the authors. The initial set of terms was composed of labels that repository contributors usually apply to UI issues like “UX”, “Usability”, “UI” or “Design”, plus the UI elements commonly appearing with the previous labels such as “button”, “menu” and “link”, as well as verbs describing user actions like “click”, “scroll” and “filter”. An issue was considered UX-related if at least it included 3 keywords of the analyzed set. The final list includes 41 search terms, listed elsewhere[2].

For the second step of identifying UX issues that may contribute to UXDebt, as previously explained, we have used the concept of UX smells.

Thus, we proceeded by filtering from the previously selected UX issues, the ones that represent UX smells: issues that require some re-design of the target UI to improve UX but are not bugs nor they add new functionality. To discard bugs from the list of UX issues, we searched for issues containing the keywords “bug”, “crash” or “defect”. Then, to identify UX smells, we elaborated a list of keywords by manually analyzing a random subset of UX issues from each project. These keywords are terms describing awkward user interaction such as “annoying”, “confusing”, and “frustrating”. Another source of keywords to identify UX smells were Bugzilla tags that match UX smells definitions. Three experts independently classified the tags and then agreed on the final list (e.g., tag “ux-error-prevention” is a UX smell that may be solved with the refactoring “Anticipate validation” (Grigera et al., 2017)). The complete list of search terms for UX smells is also reported elsewhere[2]. This again was an iterative step where a manual check of a random sample followed the automatic search a minimum of 5 times.

Table 2: Projects analyzed

Project	Stars	#Issues	UX Issues		
			smells	bugs	others
keystonejs/keystone-classic	14.8k	2.458	74	127	186
jitsi/jitsi-meet	17k	4.815	160	134	205
matomo-org/matomo	15.6k	11.853	394	431	325
frappe/erpnext	10k	12.289	245	473	545
atom/atom	54.3k	16.765	581	888	1019
metabase/metabase	26.5k	10.026	518	1129	858
microsoft/vscode	125k	122.985	5091	9053	9832

As a result of this process, for each project, we classified UX issues into three categories: smells, bugs, and others. The latter includes things like questions and feature requests proposed by contributors and users. Table 2 shows, for each project, the total number of issues and the amount of UX issues found in each category. An example issue categorized as UX smell follows:

Context menu in MacOS: "Show in File Manager" should be

"Show in Finder". *When I right-click a file in the tree explorer on MacOS, I no longer have an option for "Show in Finder"; it now says "Show in File Manager". This is rather confusing. I don't have a program called "File Manager".* (atom/atom).

In all the analyzed projects, at least 20% of the total UX issues are UX smells. Based on this result, we conclude that there is a significant number of issues that may potentially contribute to the occurrence of UXD.

4.3. RQ 2.3: How much attention is paid to solving potential UXDebt items?

We were also interested in quantifying how much attention developers pay to the resolution of potential UXDebt items as compared to other categories of UX issues, and to this aim, we analyzed the proportion of closed issues and their resolution times (we assume that closed issues are solved issues). With respect to the resolution time (number of days that the issue remained open), we normalized it by dividing it by the number of lines of code (#LOC) changed to solve the issue. Normalizing resolution time by size lets us fairly compare different types of issues. In order to obtain the #LOC changed by an issue solution, we developed a web scrapper that queries the pull requests (PR) linked to the issues and gets the #LOC of the PRs that were successfully merged. Closed issues with no linked PRs were not included in the resolution time analysis.

Concerning the items found in answering RQ 2.1 as self-admitted UX debt, only half of them are closed. This evidences the little attention that is being paid to UXDebt items, even for self-admitted debt. The detailed results are listed elsewhere[3].

In the case of the items classified as UXDebt in RQ 2.2, we calculated the proportion of issues solved and the resolution times for the three categories of UX issues (smells, bugs and others). Our hypothesis was that the proportion of solved smells was smaller than the proportion of solved bugs and other issues, and that those solved smells have larger resolution time than the other two categories. Table 3 shows for each project and each issue category, the proportion of issues solved along with their resolution times. In order to assess if the differences in resolution time were statistically significant, we ran a series of tests comparing the resolution time of smells with the remaining issues (bugs and other issues were considered as a single group). Given the non-normality of the data, we applied the one-tailed variant of the Mann-Whitney U test. The results are shown in the p-value column of each project.

It can be observed that out of the 7 projects, 4 have around or at least 20% of smells not solved. That is a significant proportion of potential UXDebt items. Moreover, in the projects *matomo-org/matomo* and *metabase/metabase*, the proportions of closed issues show that smells are the least solved problems. The issue resolution times of both projects also show that UX smells receive less attention than the other two categories, **but only in *matomo-org/matomo*this difference is statistically significant (p-value$\leq$0.05)**. In the case of *jitsi/jitsi-meet*, although there is not a considerable difference between the solved proportion of the three issue types, the resolution time is larger for smells than for bugs and other issues, which is confirmed by the resulting p-value. Something similar happens with *frappe/erpnext* project, with the difference that the proportion of solved bugs is lower than smells and others issues.

Regarding the remaining three projects, *keystonejs/keystone-classic*, *atom /atom* and *microsoft/vscode*, the smells are not the least solved issues and their mean resolution time is not much greater than the other two categories. The p-value$\geq$0.05 for *keystonejs/keystone-classic* suggests that the smells do not seem no a to receive less attention than the remaining UX issues. In the case of *atom /atom* and *microsoft/vscode*, although the differences in resolution time are not remarkable, they are statistically significant according to the p-value, so the smells may require more time to be solved than bugs and other issues.

4.4. Lesson 3: UX smells in issue tracking repositories may be used to estimate the technical aspects of UXDebt

In the lessons at the end of Section 3, we uncover that UXDebt has both a strictly technical side and another non-technical side. The evidence gathered from mining GitHub allows us to tackle some of the technical aspects of UXDebt with UX smells.

That is, UX smells, equivalently to code smells, represent possible sources of problems that may indicate the presence of debt (Fontana et al., 2012; Ramirez Lahti et al., 2021), in this case, UXDebt. Going back to the definition of TD, smells are realized as actual debt items when they “make future changes more costly

or impossible (Avgeriou et al., 2016). In the case of UX, this may happen when extra changes are needed for example to maintain consistency in presentation or offered interaction across UI components or screens of an application.

Having UX smells as indicators of UXDebt is beneficial in that UX smells may be cataloged as concrete, identifiable problems that may even be detected automatically by analyzing user interaction logs (Grigera et al., 2017). Besides, each UX smell may include standardized solutions as UX refactorings. In fact, there are catalogs of UX smells that name identifiable problems of user interaction and link each issue to specific UX refactorings that may solve it (Grigera et al., 2017; Gardey et al., 2020). While catalogs are not complete, they are extensible and may provide a good starting point to label UXDebt issues in a way that its remediation is easy to identify.

Table 3: Resolution times for UX issues.

Issues resolution time (#days/#LOC)						
keystonejs/keystone-classic						
Issue Category	solved	min	max	SD	mean	p-value
smells	90%	0	75.5	18.2	11.1	
bugs	68.5%	0	125.5	25.3	11.06	0.2
others	91.9%	0	307.5	55.5	21.8	
jitsi/jitsi-meet						
Issue Category	solved	min	max	SD	mean	p-value
smells	81%	0	304.3	62.2	16.9	
bugs	82.8%	0	24.2	5.5	2.73	0.0001
others	86.8%	0	32	7.09	4.03	
matomo-org/matomo						
Issue Category	solved	min	max	SD	mean	p-value
smells	69%	0	1387	183.42	40.41	
bugs	85.8%	0	539	56.55	15.11	0.033
others	75%	0	335.2	39	12.45	
frappe/erpnext						
Issue Category	solved	min	max	SD	mean	p-value
smells	73%	0	910	140.8	30.5	
bugs	57.5%	0	240	37.8	10.6	0.001
others	78.8%	0	102.7	15.6	5.76	
atom/atom						
Issue Category	solved	min	max	SD	mean	p-value
smells	93%	0	340	37.38	11.13	
bugs	91.8%	0	433	38.78	8.44	≈0
others	94.1%	0	425	52.34	13.45	
metabase/metabase						
Issue Category	solved	min	max	SD	mean	p-value
smells	69.1%	0	280.3	39.4	11.91	

bugs	77.7%	0	370	25.97	5.34	0.16
others	77.1%	0	779	49.76	9.92	
microsoft/vscode						
Issue Category	solved	min	max	SD	mean	p-value
smells	92%	0	443	36	8.86	
bugs	93%	0	339.5	27.27	8.02	≈0
others	94.5%	0	242	28.7	8.1	

For example, the issue described in Section 4.2 about a confusing item name in a MacOS context menu, could be identified by the UX smell “Undescriptive element”, which in our catalog has two alternative refactorings associated with it, “Rename Element” and “Add Tooltip” (Grigera et al., 2017). Since these refactorings involve different implementation efforts, once a refactoring option is selected, the related UXDebt item can be quantified by the cost involved and prioritized according to the cost/benefit of remediation.

Therefore, the principal of the UXDebt item generated by that UX smell can be determined by the repair cost of the associated UX refactoring. Each UX smell can generate an evolution or maintenance effort that increases over time which would allow quantifying the interest corresponding to that UXDebt item. This interest is caused by the ripple effect implied in postponing its correction. Twidale points out the ripple effect of usability issues (Twidale and Nichols, 2005). When the solution of an interface problem breaks the consistency of the design and requires it to be extended to nonproblematic elements. The example shown in answer to RQ 2.1 about the lack of consistency between link preview texts on hover generates the need to solve each case independently as a kind of interest for future releases.

We have also learned from our experiment that most of the issues identified as UX smells in GitHub repositories are related to the pragmatic or instrumental aspects of the UX. Nevertheless, the pragmatic aspects of UX perceived and experienced (e.g., UI usability) have a significant impact on the hedonic ones (perceived aesthetics and hedonic identification quality) (Tuch et al., 2012). Low usability results, high interaction effort, and the presence of UX smells will contribute to a low level of hedonic quality perceived. Improving UX by eliminating flaws in the pragmatic aspects (e.g., by refactoring smells), leads to transforming a possible negative user experience into a neutral one (Hassenzahl et al., 2007). Therefore, in any phase of technical work, it will be important to identify them and quantify the UXDebt to which smells contribute because their reduction is relevant in terms of the offered UX. Identifying the principal owed on pragmatic aspects of UX is important for the management process and to be able to determine the break-even point beyond which it is convenient to repay the debt.

In addition, UX literature has shown that for the step of moving from a neutral to a positive UX it is necessary to understand the positive psychological needs of users and their relevance in the specific context (e.g. (Hassenzahl et al., 2007)). To evaluate the hedonic quality of a system requires a team to perform three tasks: 1) identify a list of the user's psychological needs and link them to the attributes of the product (simple-complicated, original-typical, attractive-repulsive, etc.); 2) select the right level of granularity on which to evaluate; 3) identify relevance and importance of the attribute. Tasks 1 and 3 almost certainly have a UXDebt impact: if user research is eliminated or reduced, the team will lack elements against which to test the hedonic aspects of the actual UX, nor will you know what aspects are important to the user. Task 2 enables UX evaluation not only at the level of the entire product or system but also at a lower level of granularity, such as that which allows us to identify UX smells.

[2] https://shorturl.ae/FwHcC
[3] https://shorturl.ae/FwHcC

5. A Conceptual Model For Uxdebt

We present here a conceptual model for UXDebt based on the evidence gathered in the three previous sections. The model reflects the complexity and extent of the scenario that the evidence has shown. This model is depicted as a diagram in Fig. 3, which we will use as a guide in the following description.

Starting from the top of Fig. 3, we have learned from Section 2 and Lesson 1 that the existence of UXDebt can be suspected when gaps are detected between a desirable (optimal) UX and the actual (sub-optimal) UX offered by the system and/or experienced by users. These gaps can compromise both UX's pragmatic and hedonic aspects. The technical decisions that generate them can be made at any point in the product life cycle. Sometimes with the awareness of generating a sub-optimal solution in exchange for immediate benefits, in other cases without realizing it.

These gaps can manifest themselves through different symptoms (see Symptoms box). Some of them may be identified by the technical team itself during its work and attributed to some software artifacts in the development process (Technical Symptoms). Others may surface to end users or other stakeholders but without a direct link to a specific development artifact, usually after deliveries (Non-technical Symptoms).

Throughout the entire life cycle, the technical team should evaluate the quality of UX being offered. Any of the artifacts that the UX experts generate can be tested against the project requirements and detect Technical Symptoms of UX gaps such as lack of suitability for some user profiles, low levels of usability test results, or the presence of UX smells (see Lesson 3). Non-technical symptoms manifest themselves beyond the realm of the technical team and affect end users and other stakeholders in the organization, not project artifacts. For example, a drop in the number of active or returning users, cart abandonment in e-commerce applications, growing demand in the customer service sector for problems with the system or product (with the loss of business value of the system or the negative impact on other areas of the operation).

In both types of symptoms, there may be some extra negative effects that will remain while the original version is not fixed, or even may increase over time (see Effects box). For example, when the technical team discovers that the UX evaluation results are low because the information architecture design is too complex, when the aesthetics are perceived as inconsistent or when there are too many UX smells. It is possible that these situations generate the need to multiply efforts when implementing new versions (for example, duplicating code of components that should be refactored, generating extra links to hard-to-find functions, etc.). This extra cost, whose origin is the persistence of the sub-optimal solution, may increase over time due to the ripple effect (see Lesson 3) and be estimated by the technical team. In the non-technical symptoms, a correspondent non-technical effect may also be generated: more and more users give up or have shorter interactions, and therefore conversions persistently decrease, comments on social networks are increasingly negative so fewer and fewer users give the system a chance, etc. The quantification methods of these intangible aspects of UX investments are still under research (e.g. (Erdös, 2019)).

Whether the symptom detected is technical or non-technical, how to fill the UX gap will at some point require technical action on the UX offered. It may include deeper user research, better interaction design, or some refactoring to deliver a better UX (Solutions box). However, for that symptom to indicate the presence of a UXDebt item, it must have two associated costs: the principal and the interest (Curtis et al., 2012). The cost of implementing the known desirable solution allows for quantifying the principal owed. The extra cost due to the persistence of the sub-optimal status quo allows for quantifying the interest of the UXDebt item.

When both the capital owed and the interest fall directly on the technical team, we can speak of a UXDebt on the technical side. Following the consensus definition of TD (Avgeriou et al., 2016), we will be in the presence of a system state that makes the implementation of the optimal solution increasingly difficult and complex. The principal owed will be the cost of implementing the correct technical solution, just as the increased extra cost of maintaining the current state will also have a direct impact on the technical team.

However, when interest is paid mainly by end users or other stakeholders, we can still talk about UXDebt, but it will no longer be fully technical. This part of UXDebt, in the same way as other debts such as PD or SD, affects people and is not easily assignable to a specific artifact, but originates from the decisions of the technical team (see Section 2). For example, a proposed UX design may be the result of a trade-off between different stakeholder needs, but some trade-offs may be better than others. To paraphrase Martini (Martini et al., 2020), the UXDebt item will not exist just because the system contains a gap between the optimal desired and the sub-optimal offered, but because a better trade-off (and thus better UX) could have been achieved to solve the same problem. Although there may be no extra cost to the development team, offering a diminished UX can generate increased interest for the entire business in terms of decreased users, lost ROI, increased support calls, etc. Other debts such as PD or SD can also be indirect causes of UXDebt with non-technical negative effects. In our survey, we have clearly seen how there is a kind of mutual mistrust between software engineers and UX specialists about who causes the

most UI bugs (see Section 3 in RQ 1.7). The presence of a UX department that is unaware of the technology decisions that make certain interactions impossible, or a technology team that does not have a user-centered view of work and prioritizes its knowledge of technology over user research requirements will drive almost inevitably to sub-optimal decisions in terms of the trade-offs mentioned above.

Following the Fowler quadrant, when the existence of both, principal and interest, are known and they are accepted in exchange for immediate benefits, we will be in the presence of a self-admitted or deliberate UXDebt. As long as only the cost of the right solution is known, but the extra complexity in development and maintenance has not been identified, we can speak of inadvertent UXDebt (Fowler, 2009).

In conclusion, we define UXDebt as “the emergence of sub-optimal UX solutions that, although beneficial in the short term, create a long-term context in which there is a negative impact on the work of the technical development team or any of the non-technical stakeholders complicating future improvements”.

6. Preliminary Proposals For Uxdebt Management

UX is a relevant component in current software development (Hassenzahl and Tractinsky, 2006). The integration of UX designers and their processes for non-functional requirements definition, design, and construction with the rest of the technical team members, especially in the context of agile methodologies, is challenging and conflicting (Brhel et al., 2015). As shown by survey responses in Section 3, these challenges often translate into offering a poor UX, whose persistence decreases the value of the software and causes extra costs in the technical work. TD management aims to prevent potential debt or maintain it under control through activities like identification, analysis, prioritization, monitoring, measurement, and repayment (Li et al., 2015). Proper UXDebt management requires a characterization that accounts for its extent and complexity, and the construction of tools to carry out each of those phases.

The **identification** of UXDebt items is often carried out manually through the application of UX evaluation methods on early designs or the actual system in production. For instance, user testing provides quantitative and qualitative measures of the system in operation, so it is the preferred way to obtain actual usage data (Sauro and Lewis, 2016). Thus, UXDebt identification and measurement could be accomplished during user testing, as described in Section 2.2, by collecting a combination of metrics. There are tools for remote user testing that help in collecting some of these metrics, though they still require much handcrafting from experts. This manual effort makes it mostly unfeasible to establish a continuous UXDebt monitoring method with the agile current practice, as practitioners have with code TD tracking tools. Large companies may afford an automatic monitoring process through A/B testing, but current A/B testing tools do not provide a good measure of UX (Nielsen, 2005), nor do its visualizations provide hints on concrete solutions to UX problems.

Furthermore, in the context of agile development, rapid identification and quantification of UXDebt items are crucial. For the technical side of UXDebt, in Section 4 we propose to identify candidates of UXDebt items in ITSs by searching for UX smells. We have defined keywords and specific filters to identify UX

smells automatically, although there is an opportunity for further refinement. In previous works, we have also proposed automatic approaches for UX smell detection by analyzing user interaction logs of a system under production (Grigera et al., 2017). We have also worked on the automatic comparison of designs in terms of *interaction effort*, which we propose as a measure of UX (Gardey et al., 2022) and we believe it may provide faster UXDebt identification and better monitoring.

Product backlog grooming is an essential practice where TD items must be carefully analyzed and **prioritized**. However, it is difficult to measure and prioritize TD items since it depends on the remediation mechanisms that are finally used, and the possibility of quantifying the benefits they can bring in the future (refactoring benefits) (Li et al., 2015). In this regard, Lavazza et al. (Lavazza et al., 2018) postulate that TD is an external attribute to a software artifact because it cannot be quantified solely on the basis of the artifact, but it is also necessary to take into account factors of its context, many of which are human and imprecise factors, that are dependent on people involved in software development (Lavazza et al., 2018). This becomes particularly significant in the case of the UXDebt, where clearly the human factor is the most relevant in the measurement, and it is impossible to calculate it just by reviewing the UI code. Moreover, it is hard to evaluate the benefits of a feature before the product is available. Another mechanism that could be used to prioritize UXDebt items is the severity of the associated UX smell (which combines frequency and impact on users (Sauro and Lewis, 2016)). In this regard, user tests or automatic UX smell detection tools may provide severity measures like the number of users that run into each UX smell, or how much of a barrier it creates for users to finish a task.

Regarding UXDebt **repayment**, associating technical UX debt items to UX smells is practical in that the possible solutions may be coded as UX refactorings (Grigera et al., 2017). Nevertheless, a UX smell usually has several possible UX refactorings to solve it, depending on the context, as exemplified in Section 2.2. To shorten the time to decide on a refactoring, the visual programming tool called UX-Painter allows designers to automatically apply and compare alternative UX refactorings on the client side of a web application, creating different versions for user testing without involving developers (Gardey et al., 2020). Moreover, once a refactoring is chosen, the tool may generate a preliminary server-side implementation for the most used front-end frameworks such as ReactJS (Gardey et al., 2021). The goal of generating refactorings code is to minimize the UXDebt repayment cost by reducing the developer's effort to code the solutions. Yet, an added complexity to UXDebt repayment is that fixing a smelly UI component may provoke a ripple effect to maintain consistency with other, non-smelly ones, that may potentially be anywhere in the application (Twidale and Nichols, 2005). This is an aspect that will require more work in the future, besides extending UXPainter with a broader set of possible refactorings.

Measurement of UXDebt is calculated in terms of principal and interest. While the principal may be calculated as the cost of remediation, interest is very difficult to measure since it depends on the expected future development, not just on a rate and the passing of time (as discussed in Section 2.1), and in the case of UXDebt, it depends heavily on human factors. This is an area for future research

Regarding the **non-technical side of UXDebt**, at least two points of view should be contemplated. On the one hand, the real users and the actual UX they experienced. On the other, the impact that this behavior (which depends in part on the UX offered) has on business objectives.

UX design offers indicators of the level achieved by the UX of a system and a wide set of methods to obtain them, both in pragmatic and hedonic aspects. Some of these metrics may be translated into behavioral indicators that the technical team can use as goals for their work (such as Key Performance Indicators or KPI (Hinderks et al., 2019)). Other proposals seek to align technical UX design work with business objectives (e.g., IBM's "7 steps to bridge UX and business value" (Gregerson and Rizzi, 2021)). In any case, the only way to identify this non-technical aspect of the UXDebt is to perform user evaluation on an ongoing basis after deliveries.

From the UX literature, there is evidence that a good UX is good for business (e.g., (Hassenzahl et al., 2007)), but to convince software investment managers it is necessary to show calculations in their economic language. For example, the cost-benefit analysis can be applied as a decision support method (Erdös, 2019). Software project management offers several empirical cost estimation models that can be useful (e.g. COCOMO, COCOMO II (Boehm et al., 2009)). To estimate the benefits of UX for the company, it may be convenient to have an appropriate technology acceptance model (such as the TAM (Lee et al., 2003)) and to translate business objectives into indicators with correlation in the work of the technical team. After that, analysts will be in a position of doing usual economic calculations such as Return on Investment or Internal Rate of Return). The challenge here may be to reconcile some unawareness of financial analysts about the intangible benefits of UX and the need for project managers to know the language of finance. The management of UXDebt in its non-technical side impacting the business value requires the identification of the relationships (and eventual correlations) between UX technical decisions, the actual user behavior, and the business objectives, which remains an open research topic.

7. Conclusion

Research studies have proposed the notions of usability debt or UX debt but mainly recognize the cost that users pay through sub-optimal UX. Yet, the evidence presented in Sections 4 and 3 shows that although the existence of a cost for end-users is clear, there is also a cost for the agile team due to decisions under pressure, lack of resources, social problems within the team, etc. That is, it "sets a technical context that may make future changes more costly or impossible" (Avgeriou et al., 2016). In a way, this presents UXDebt as a phenomenon that could be paid twice. We believe this is a topic for further research in the future.

In this work, we have collected evidence from a survey with practitioners and the mining of GitHub, and propose a conceptual model that characterizes UXDebt, both in its technical and non-technical aspects. We have also shown that the notion of UX smells may allow identifying a technical form of UXDebt, both in ITS or running systems. Once identified, other management activities like measurement, prioritization and repayment become viable, and UX refactoring becomes a useful tool for these activities.

We have come a long way in proposing and empirically evaluating UX smells and refactorings as building blocks for the systematic improvement of UX during a development cycle, including several tools; however, there is still a research gap to fill with automatic methods and tools for the systematic remediation of UXDebt and proper UXDebt management. Our future work includes defining an ample catalog of UX smells and UX refactorings, as well as evaluating their concrete benefits towards UXDebt management. We are also interested in studying the non-technical aspects of UXDebt, which similarly to SD and PD, are difficult to identify, measure and repair.

Declarations

Data availability

The original form and the answers (in Spanish) for the Survey in Section 3 are available in *http://t.ly/d_6P*. The queries and results for analysis in Section 4 are available in *https://shorturl.ae/ugllf*.

Authorship

Conceptualization, Methodology, Writing - review and editing: Andrés Rodríguez, Juan Cruz Gardey, Julián Grigera, Alejandra Garrido, Gustavo Rossi; Investigation: Andrés Rodríguez, Juan Cruz Gardey, Julián Grigera; Formal analysis: Juan Cruz Gardey, Julián Grigera; Writing - original draft preparation: Andrés Rodríguez, Juan Cruz Gardey; Funding acquisition: Alejandra Garrido; Supervision: Alejandra Garrido, Gustavo Rossi

Acknowledgments

This work was supported by the Argentinian National Agency for Research, Technological Development, and Innovation (Agencia I+D+i), grant number PICT-2019-02485.

Conflict of Interest Statement

The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Alves, N. S., Mendes, T. S., de Mendonca, M. G., Spınola, R. O., Shull, F., & Seaman, C. (2016). Identification and management of technical debt: A systematic mapping study.Information and Software Technology70.
2. Ampatzoglou, A., Ampatzoglou, A., Chatzigeorgiou, A., & Avgeriou, P. (2015). The financial aspect of managing technical debt: A systematic literature review.Information and Software Technology64.
3. Avgeriou, P., Kruchten, P., Ozkaya, I., & Seaman, C. (2016). *Managing technical debt in software engineering*. in: Dagstuhl Reports, Schloss DagstuhlLeibniz-Zentrum fuer Informatik.

4. Baltes, S., & Dashuber, V. (2021). Ux debt: Developers borrow while users pay.arXiv preprint arXiv:2104.06908.
5. Behutiye, W. N., Rodriguez, P., Oivo, M., & Tosun, A. (2017). Analyzing the concept of technical debt in the context of agile software development: A systematic literature review.Information and Software Technology82.
6. Besker, T., Martini, A., & Bosch, J. (2017). The pricey bill of technical debt: When and by whom will it be paid?, in: IEEE International Conference on Software Maintenance and Evolution, IEEE.
7. Besker, T., Martini, A., & Bosch, J. (2019). Software developer productivity loss due to technical debt—a replication and extension study examining developers' development work.Journal of Systems and Software156.
8. Blair, J., Czaja, R. F., & Blair, E. A. (2013). *Designing surveys: A guide to decisions and procedures*. Sage publications.
9. Boehm, B. W., Abts, C., Brown, A. W., Chulani, S., Clark, B. K., Horowitz, E., Madachy, R., Reifer, D. J., & Steece, B. (2009). *Software cost estimation with COCOMO II*. Prentice Hall Press.
10. Brhel, M., Maedche, & Werder (2015). Exploring principles of user-centered agile software development: A literature review.Information and Software Technology61.
11. Bruun, A., Larusdottir, M., Nielsen, L., Nielsen, P., & Persson, J. (2018). The role of ux professionals in agile development: a case study from industry, in: Proceedings of the 10th Nordic Conference on HCI.
12. Chan, V. (2017). Here's the reverse: 7 things i wished developers did more of when working with designers. URL: https://t.ly/erXp.
13. Ciolkowski, M., Lenarduzzi, V., & Martini, A. (2021). 10 years of technical debt research and practice: Past, present, and future.IEEE Software38.
14. Cooper, A., Reimann, R., & Cronin, D. (2015). About face. The Essentials of Interaction Design 3.
15. Cunningham, W. (1992). The wycash portfolio management system. ACM SIGPLAN OOPS Messenger 4.
16. Curtis, B., Sappidi, J., & Szynkarski, A. (2012). Estimating the size, cost, and types of technical debt, in: 2012 Third International Workshop on Managing Technical Debt, IEEE.
17. Da Silva, T., Silveira, M., Maurer, F., & Silveira, F. (2018). The evolution of agile UXD.Information and Software Technology102.
18. Da Silva, T. S., Martin, A., Maurer, F., & Silveira, M. (2011). User-centered design and agile methods: A systematic review, in: Agile 2011.
19. Djamasbi, S., McAuliffe, D., Gomez, W., Kardzhaliyski, G., Liu, W., & Oglesby, F. (2014). In F. F. H. Nah (Ed.), *Designing for success: Creating business value with mobile ux*. HCI in Business, Springer International Publishing.
20. Erdös, F. (2019). Economical aspects of ux design and development, in: 10th IEEE International Conference on Cognitive Infocommunications (CogInfoCom).

21. Firmenich, S., Garrido, A., Grigera, J., Rivero, J., & Rossi, G. (2019). Usability improvement through a/b testing and refactoring.Software Quality Journal27.
22. da Lage, F., Kalinowski, L., Trevisan, M., & Spinola, D. (2019). R., Usability technical debt in software projects: A multi-case study, in: 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, IEEE.
23. Fontana, F., Ferme, V., & Spinelli, S. (2012). Investigating the impact of code smells debt on quality code evaluation, in: 2012 Third International Workshop on Managing Technical Debt, IEEE.
24. Fowler, M. (2009). Technical debt quadrant. URL: http://martinfowler.com/bliki/TechnicalDebtQuadrant.html.
25. Gardey, J. C., Garrido, A., Firmenich, S., Grigera, J., & Rossi, G. (2020). Uxpainter: An approach to explore interaction fixes in the browser. EICS 4.
26. Gardey, J. C., Grigera, J., Rodrıguez, A., Rossi, G., & Garrido, A. (2022). Predicting interaction effort in web interface widgets. *International Journal of Human-Computer Studies*, *168*, 102919.
27. Gardey, J. C., Grigera, J., Rossi, G., & Garrido, A. (2021). UX-Painter: Fostering UX Improvement in an Agile Setting, in 11th Brazilian Workshop on Agile Methods.
28. Garrido, A., Firmenich, S., Grigera, J., & Rossi, G. (2017). Data-driven usability refactoring: tools and challenges, in 6th International Workshop on Software Mining, IEEE.
29. Garrido, A., Rossi, G., & Distante, D. (2011). Refactoring for usability in web applications. *IEEE Software*, *28*, 60–67.
30. Gothelf, J., & Seiden, J. (2021). In U. X. Lean (Ed.), *Applying Lean Principles to Improve User Experience*. O'Reilly Media, Inc.
31. Gregerson, D., & Rizzi, T. (2021). 7 steps to bridge user experience and business value. URL: https://t.ly/gjR4.
32. Grigera, J., Garrido, A., Rivero, J., & Rossi, G. (2017). Automatic detection of usability smells in web applications.International Journal of Human-Computer Studies97.
33. Hassenzahl, M., Burmester, M., & Koller, F. (2021). User experience is all there is. i-com 20.
34. Hassenzahl, M., & Tractinsky, N. (2006). User experience-a research agenda.Behaviour & information technology25.
35. Hassenzahl, M., et al. (2007).Being and doing–a perspective on user experience and its measurement.
36. Hinderks, A., Schrepp, M., Mayo, F. J. D., Escalona, M. J., & Thomaschewski, J. (2019). Developing a ux kpi based on the user experience questionnaire. *Computer Standards & Interfaces*, *65*, 38–44.
37. ISO (2011). ISO-IEC 25010: 2011 Systems and Software Engineering-Systems and Software Quality Requirements and Evaluation. Technical Report.
38. ISO (2019). ISO 9241 – 210:2019 Ergonomics of human-system interaction Part 210: Human-centred design for interactive systems. ISO/TC 159/SC 4.
39. Kaley, A. (2018). UX debt: How to identify, prioritize, and resolve. URL: https://t.ly/iUNC.

40. Kohavi, R., Tang, D., & Xu, Y. (2020). *Trustworthy online controlled experiments: A practical guide to a/b testing*. Cambridge University Press.
41. Kuusinen, K. (2016). Bob: a framework for organizing within-iteration ux work in agile development. *Integrating User-Centred Design in Agile Development*. Springer.
42. Lavazza, L., Morasca, S., & Tosi, D. (2018). Technical debt as an external software attribute, in: Proceedings of the 2018 International Conference on Technical Debt.
43. Law, E. L. C., Roto, V., Hassenzahl, M., Vermeeren, A. P., & Kort, J. (2009). Understanding, scoping and defining user experience: a survey approach, in: Proceedings of the SIGCHI conference on human factors in computing systems.
44. Lee, Y., Kozar, K. A., & Larsen, K. R. (2003). The technology acceptance model: Past, present, and future. *Communications of the Association for information systems*, *12*, 50.
45. Lenarduzzi, V., Besker, T., Taibi, D., Martini, A., & Fontana, F. A. (2021). A systematic literature review on technical debt prioritization: Strategies, processes, factors, and tools.Journal of Systems and Software171.
46. Li, A., & Liang (2015). A systematic mapping study on technical debt and its management.Journal of Systems and Software101.
47. Martini, A., Besker, T., & Bosch, J. (2020). Process debt: a first exploration, in: 27th Asia-Pacific Software Engineering Conference, IEEE.
48. Maudet, N., Leiva, G., Beaudouin-Lafon, M., & Mackay, W. (2017). Design breakdowns: Designer-developer gaps in representing and interpreting interactive systems, ACM.
49. Nielsen, J. (2005). *Putting a/b testing in its place*. Useit.com Alertbox.
50. Nielsen, J. (2020). 10 usability heuristics for user interface design. URL: https://t.ly/7VE4.
51. Potdar, A., & Shihab, E. (2014). An exploratory study on self-admitted technical debt, in: IEEE Int. Conference on Software Maintenance and Evolution, IEEE.
52. Ramirez Lahti, J., Tuovinen, A. P., Mikkonen, T., et al. (2021). Experiences on managing technical debt with code smells and antipatterns, in: IEEE/ACM International Conference on Technical Debt, IEEE.
53. Rios, N., de Mendonc¸a Neto, M. G., & Sp´ınola, R. (2018). A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners.Information and Software Technology102.
54. Sauro, J., & Lewis, J. R. (2016). *Quantifying the user experience: Practical statistics for user research*. Morgan Kaufmann.
55. Silva, T. S., Silveira, M. S., Melo, O., & Parzianello, C. (2013). L.C., Understanding the ux designer's role within agile teams, in: International Conference of Design, User Experience, and Usability, Springer.
56. Störrle, H., & Ciolkowski, M. (2019). Stepping away from the lamppost: Domain-level technical debt, in: 45th Euromicro Conference on Software Engineering and Advanced Applications, IEEE.
57. Tamburri, D. A., Kruchten, P., Lago, P., & van Vliet, H. (2013). What is social debt in software engineering? in 6th International Workshop on Cooperative and Human Aspects of Software

Engineering, IEEE.

58. Theodoropoulos, T., Hofberg, M., & Kern, D. (2011). Technical debt from the stakeholder perspective, in: Proceedings of the 2nd Workshop on Managing Technical Debt.
59. Tom, E., Aurum, A., & Vidgen, R. (2013). An exploration of technical debt.Journal of Systems and Software86.
60. Tsoukalas, D., Siavvas, M., Jankovic, M., Kehagias, D., Chatzigeorgiou, A., & Tzovaras, D. (2018). Methods and tools for td estimation and forecasting: A state-of-the-art survey, in: 2018 IS, IEEE.
61. Tuch, A. N., Roth, S. P., Hornbæk, K., Opwis, K., & Bargas-Avila, J. A. (2012). Is beautiful really usable? toward understanding the relation between usability, aesthetics, and affect in hci. *Computers in Human Behavior*, *28*, 1596–1607.
62. Twidale, M., & Nichols, D. (2005). Exploring usability discussions in open-source development, in: Proceedings of the 38th Annual Hawaii International Conference on System Sciences, IEEE.
63. Wohlin, C., Runeson, P., H¨ost, M., Ohlsson, M. C., Regnell, B., & Wesslen, A. (2012). *Experimentation in software engineering*. Springer Science & Business Media.
64. Wright, A. (2013). User experience debt. URL: https://t.ly/RHuk.
65. Xavier, L., Ferreira, F., Brito, R., & Valente, M. (2020). Beyond the code: Mining self-admitted technical debt in issue tracker systems, in: Proceedings of the 17th International Conference on Mining Software Repositories.
66. Zazworka, N., Spinola, R., Vetro, A., Shull, F., & Seaman, C. (2013). A case study on effectively identifying technical debt, in: Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering.

Figures

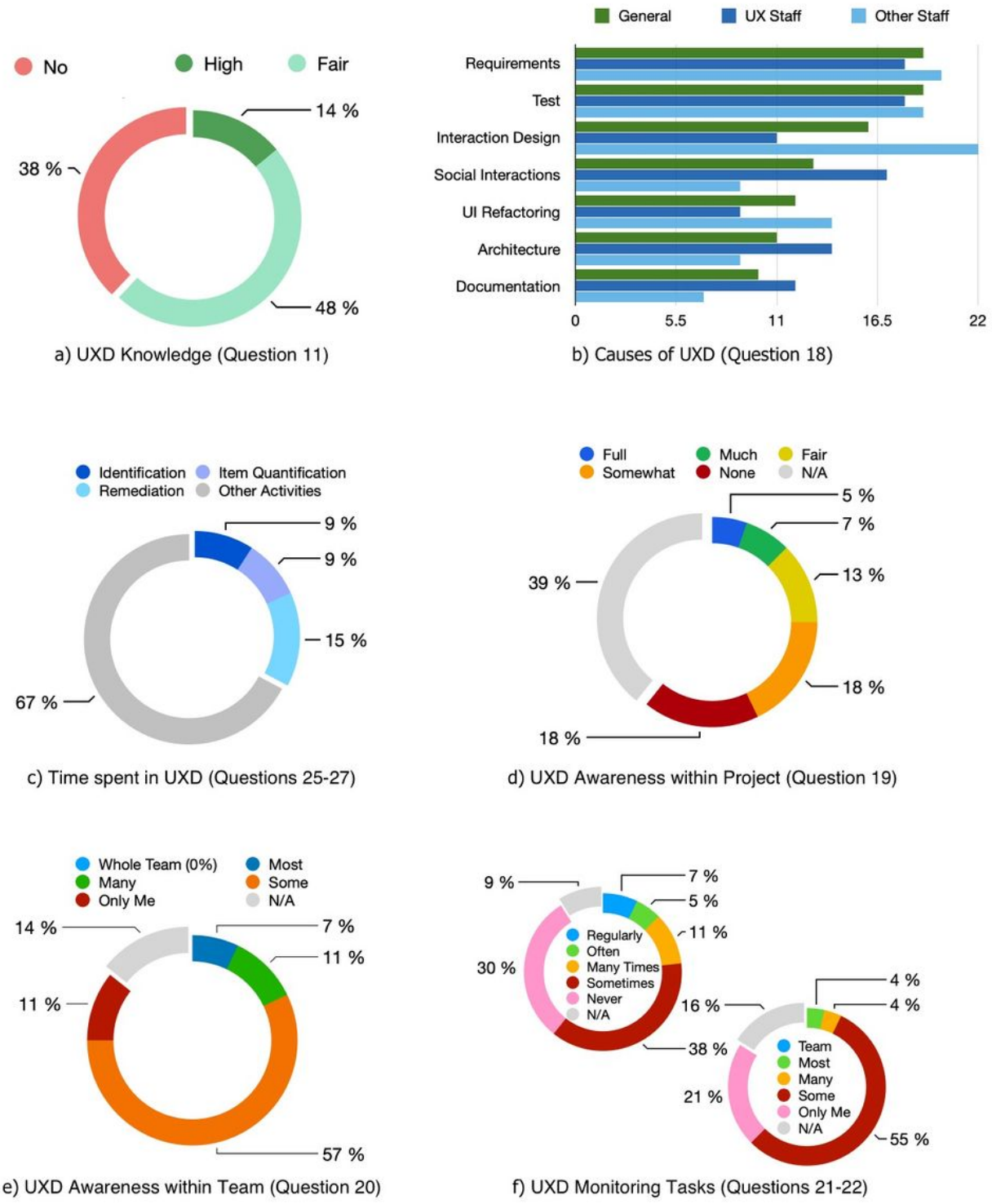
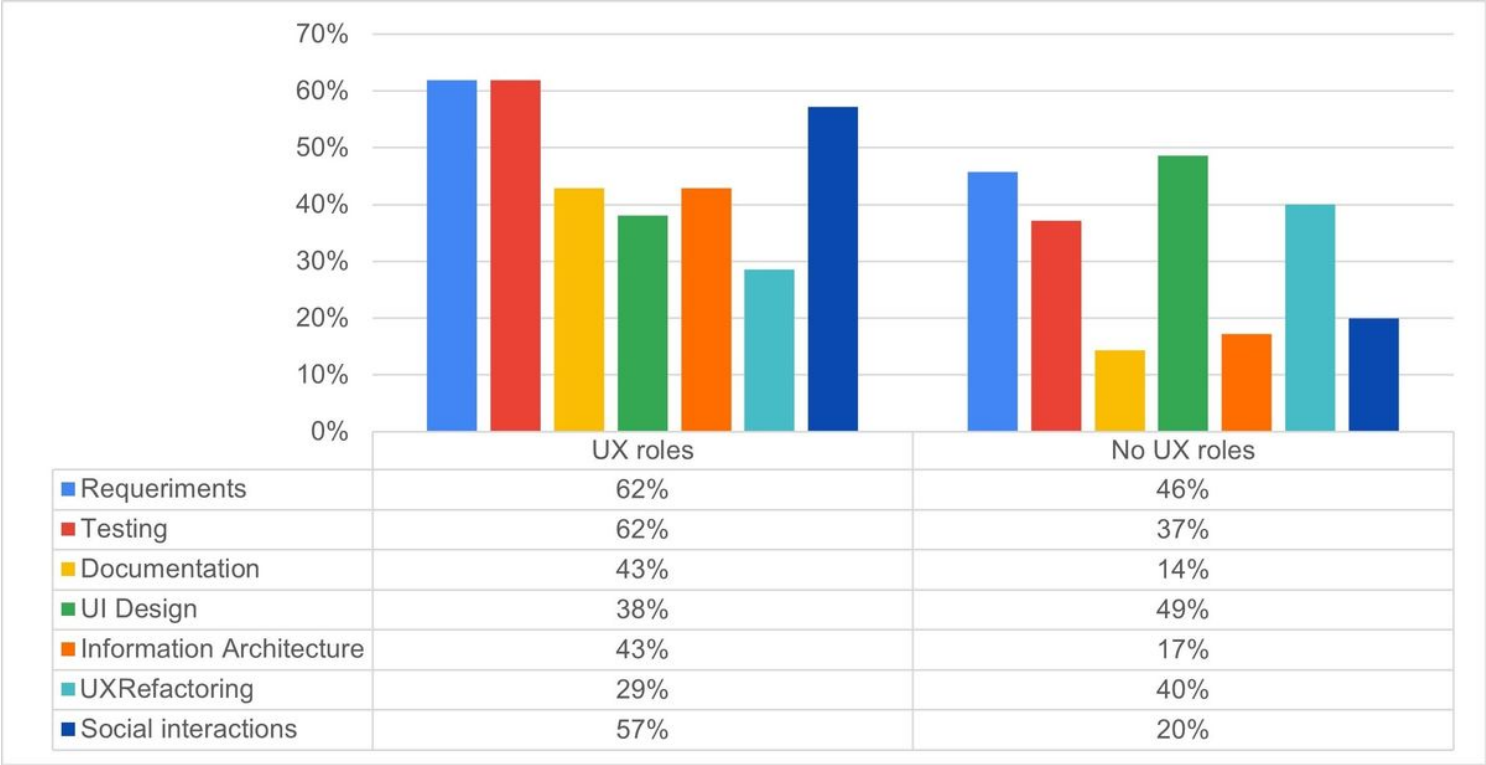


Figure 1

Charts illustrating RQ answers with quantitative data



	UX roles	No UX roles
Requeriments	62%	46%
Testing	62%	37%
Documentation	43%	14%
UI Design	38%	49%
Information Architecture	43%	17%
UXRefactoring	29%	40%
Social interactions	57%	20%

Figure 2

UXDebt causes as identified by roles. Labels identify the probable causes and percentages show the proportion of each group reporting it

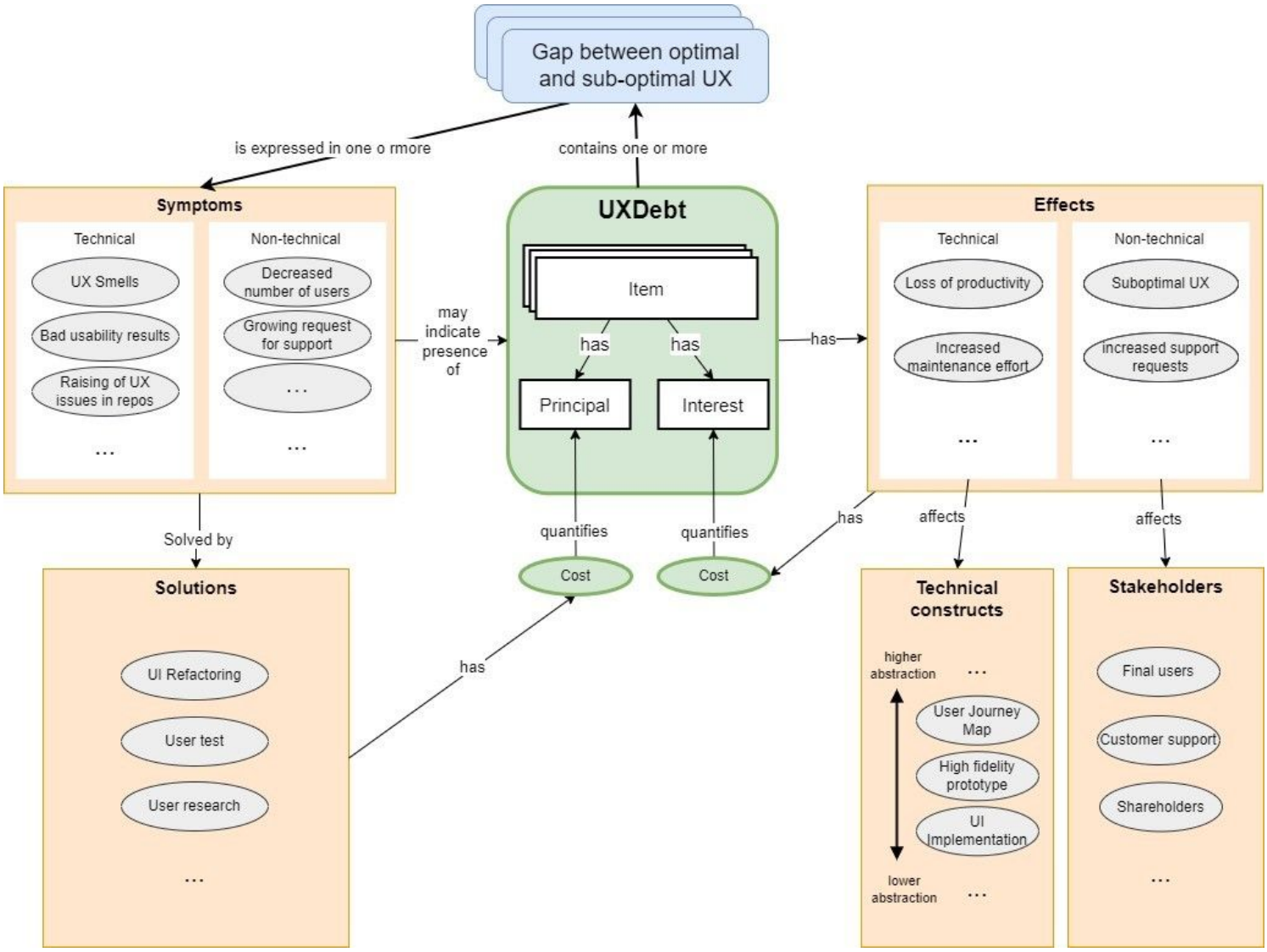


Figure 3

Conceptual model of UXDebt