Architectural Technical Debt Identification Based on Architecture Decisions and Change Scenarios

CONFERENCE PAPER · MAY 2015

DOI: 10.1109/WICSA.2015.19

3 AUTHORS:

Zengyang Li

Wuhan University

11 PUBLICATIONS **50** CITATIONS

SEE PROFILE

Peng Liang

Wuhan University

90 PUBLICATIONS **486** CITATIONS

SEE PROFILE

Paris Avgeriou

University of Groningen

199 PUBLICATIONS **1,883** CITATIONS

SEE PROFILE

Architectural Technical Debt Identification based on Architecture Decisions and Change Scenarios

Zengyang Li
Department of Mathematics and Computing Science
University of Groningen
Groningen, The Netherlands
zengyangli@gmail.com

Peng Liang
State Key Lab of Software Engineering, School of Computer
Wuhan University
Wuhan, China
liangp@whu.edu.cn

Paris Avgeriou
Department of Mathematics and Computing Science
University of Groningen
Groningen, The Netherlands
paris@cs.rug.nl

***Abstract*[1] —Architectural technical debt (ATD) is incurred by design decisions that intentionally or unintentionally compromise system-wide quality attributes, particularly maintainability and evolvability. ATD is harmful to the system's long-term health, thus it needs to be identified for further management. However, existing ATD identification approaches are mainly based on source code analysis and thus suffer from certain shortcomings: they can only identify issues at the system implementation; they can only be employed after the systems is implemented in code; they lack a mechanism to confirm whether the potential ATD identified is real ATD or not. To address these issues, we proposed an ATD identification approach based on architecture decisions and change scenarios. To evaluate the effectiveness and usability of this approach, we conducted a case study with an information system in a large telecommunications company. The results show that the proposed approach is useful and easy to use, and it supports release planning and ATD interest measurement.**

***Keywords*—Architectural technical debt, architectural technical debt identification, architecture decision, change scenario**

I. Introduction

Architectural technical debt (ATD) is caused by architecture decisions (ADs) that consciously or unconsciously compromise system-wide quality attributes (QAs), especially maintainability and evolvability [1]. Typical ATD includes violations of best practices, consistency, and integrity of software architectures (SAs), such as breaking the principles of architectural patterns and introducing architecture smells (e.g., package dependency cycles). ATD needs to be well managed in order to maintain the long-term health of the system [1, 2]. If left unmanaged, the ATD will accumulate interest incrementally, potentially causing risks: making it challenging to conduct maintenance tasks, introduce new features, as well as meet system QAs, etc.

The first step in an ATD management (ATDM) process is to identify the existing ATD in the software system [1]. ATD identification detects the locations of the ATD in the system, and identifies the causes of the ATD, the compromised QAs, and the impact on future development [1, 3, 4]. In our recent mapping study on technical debt [3], we found that existing approaches on identifying ATD are mainly based on source code analysis. However, such ATD identification approaches can only identify, as expected, source code-related issues, e.g., modularity violations. They cannot identify ATD caused by ADs that are not reflected in the code, such as inappropriate, immature or obsolete technologies used, and architecture nonconformance. Furthermore, those ATD identification approaches can only be applied after the system is implemented, which may lead to significant rework in order to repay the ATD. Finally, the source code analysis-based approaches can only provide indicators for ATD, i.e., identify the location of *potential* ATD, but provide no means to confirm whether it is actual ATD or not.

To overcome the shortcomings of source code analysis-based ATD identification approaches, we propose to identify ATD based on ADs made during the architecting process and change scenarios for software maintenance and evolution. An AD is a design decision that affects the architecture design space for a target software system [5]. A change scenario is a maintenance or evolution task to be performed in the software system. The proposed approach takes ADs and change scenarios as input, and then comes up with ATD items that are caused by the ADs or negatively impact the change scenarios. To validate our proposed approach, we conducted an industrial case study which evaluates the effectiveness and usability of identifying ATD using this approach. The results show that the approach is useful and easy to use, and it supports release planning and ATD interest measurement.

The rest of this paper is organized as follows. Section II discusses related work regarding ATD identification, while Section III and IV present, respectively, the proposed approach and its evaluation through a case study in an industrial environment. The results of the case study are discussed in Section V, followed by the conclusions of this work and discussion of future work in Section VI.

II. Related Work

Not much work has been done on ATD identification according to our recent mapping study on technical debt [3]. Existing research focuses on identifying ATD based on source code analysis. In [6, 7], Curtis et al. used a tool (CAST's Software's Applications Intelligence Platform) to detect highly

[1] This work is partially supported by AFR grant No. 895528, and NSFC grants No. 61170025 and 61472286. Thank the participants of the case study, as well as Christian Manteuffel and Daniel Feitosa for reviewing an early version of this paper.

coupled components based on source code analysis. In [8], Eisenberg applied Sonar (a code analysis platform) to calculate package interdependencies in order to identify potential tightly coupled packages. Wong et al. proposed to identify modularity violations (a type of ATD) by comparing modular structure of the system obtained by source code analysis, with actually co-changing components obtained by revision history parsing [9], and this approach was also employed in [10, 11] to identify modularity violations as ATD. Wang et al. proposed an approach to detect so-called bad dependencies (i.e., underutilized and inconsistent modular dependencies) via analyzing symbol-level dependencies in source code [12]. Letouzey and Ilkiewicz mentioned that architecture-related changeability issues, such as cyclic dependence, can be detected by source code analysis [13].

Brondum and Zhu proposed an approach to visualize structural and behavioral dependency relationships between architecture elements (e.g., components) [14], and the authors claimed that this visualization can help to identify ATD, but they did not formulate a systematic method or operational guidelines for ATD identification.

Nord et al. defined a metric for managing ATD as the total cost, per release, of the implementation of new architectural elements (e.g., components) in this release and the rework of pre-existing elements in previous releases [15]. This metric in principle can be used to identify a more cost-efficient evolution path (i.e., release plan) by comparing the cumulative values of the metric of alternative evolution paths. In this sense, an evolution path with a larger value of the metric, is considered to incur more ATD. However, it is unclear what types of ATD can be identified and where the location of the ATD is.

III. Approach

In this section, we first present the rationale of the ATD identification approach, then describe the approach in detail, and finally discuss how the proposed approach addresses the shortcomings of existing source code analysis-based ATD identification approaches.

A. Rationale

ATD is often represented in the form of ATD items [1]. An ATD item is a unit of ATD in a system. An ATD item is incurred by an AD that compromises one of the following QAs: *modularity*, *reusability*, *analyzability*, *modifiability*, *testability*, and *evolvability* [1]. The first five QAs are the sub-QAs of maintainability according to ISO 25010 [16]. We consider that maintainability is about how easy bugs can be fixed and improvements can be completed, while evolvability is defined as the ease of adding new requirements [1]. An ATD item has a negative impact on one or more change scenarios. The impacted scenarios need more effort to be completed when the ATD item is unresolved than if the ATD item did not exist. The extra effort for completing those impacted change scenarios is called *interest* of the ATD item. This means that the interest of an ATD item is affected by its impacted change scenarios. If a change scenario reoccurs regularly, ATD measurement needs to calculate the scenario's interest in a long period instead of just the upcoming release or next six months. The relationships between ATD items, ADs, change scenarios, QAs, and interest are shown in a conceptual model (see Fig. 1).

We use an example to further explain these concepts and their relationships. In a strict layered software system, a new architect made an AD of *cross-layer dependencies*, i.e., allowing a higher layer to have dependencies to layers other than the one directly below it. This AD compromised system *modularity* thus incurred an (unintentional) ATD item called *violation of the strict layered pattern*. This ATD item negatively impacted the change scenario of *changing the APIs of the lowest layer*, resulting in extra effort needed to change multiple higher layers, rather than only changing the layer directly above the lowest one. This extra effort constitutes the ATD item's interest affected by the change scenario.

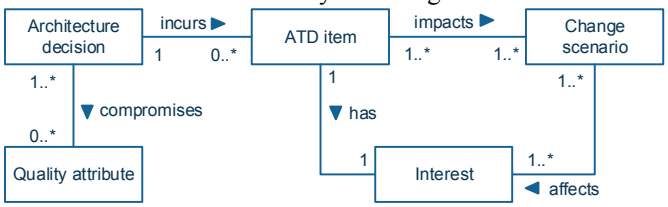


Fig. 1. Relationships between ATD related concepts

Considering the causal relationships between ATD items and ADs as well as the fact that the interest of ATD items is affected by their impacted change scenarios, we propose to identify ATD items based on ADs and change scenarios. We present the details of the proposed approach in the next section.

B. Approach Description

Before identifying ATD in a software system using this approach, architects need to record the ADs using at least the listed elements in the AD template shown in TABLE I. This step can be omitted if the ADs have already been recorded. The AD template contains the major AD elements identified in [17] and an additional element "architecture diagram". All ADs are put in an AD list. An initial prioritization of the list of ADs could help balance effort and benefit for users when it is not possible to analyze all ADs.

TABLE I. Architecture Decision Template

ID	A unique identifier of this AD
Name	A brief name for this architecture decision (AD)
Description	The details about the AD, i.e., how a design problem is solved with this AD
Alternatives	Alternative solutions to the same design problem
Rationale	The reason why the decision was chosen over alternatives
Pros	The advantages of the decision
Cons	The disadvantages of the decision
Architecture diagram	A diagram or model that illustrates the affected part (e.g., affected components) in the architecture design

Similarly, the change scenarios also need to be established. At first, a list of change scenarios needs to be created. The project's list of unresolved bugs, planned improvements on QAs and functionalities, as well as new requirements (on both QAs and functionalities) could serve as candidate change scenarios. In the next step, users (i.e., project members) mark the change scenarios that are hard to complete or have been postponed since those change scenarios are more likely to be

negatively impacted by ATD. The list of candidate change scenarios is subsequently reduced to include only the marked change scenarios. In practice, users need to choose a reasonable length of development time period in which the planned change scenarios will be analyzed to identify ATD. The longer the chosen time period, the more change scenarios will be analyzed, and thus more ATD items may be identified for further management. However, more change scenarios to be analyzed entails more required effort for each execution of the proposed ATD identification approach. To balance the required effort for executing the approach, against the risk of the system's long-term health (maintainability and evolvability), users could limit the number of to-be-analyzed change scenarios, for example, only analyzing the change scenarios for the upcoming release or the following six months. An initial prioritization of the list of change scenarios is recommended when it is not possible to analyze all the change scenarios.

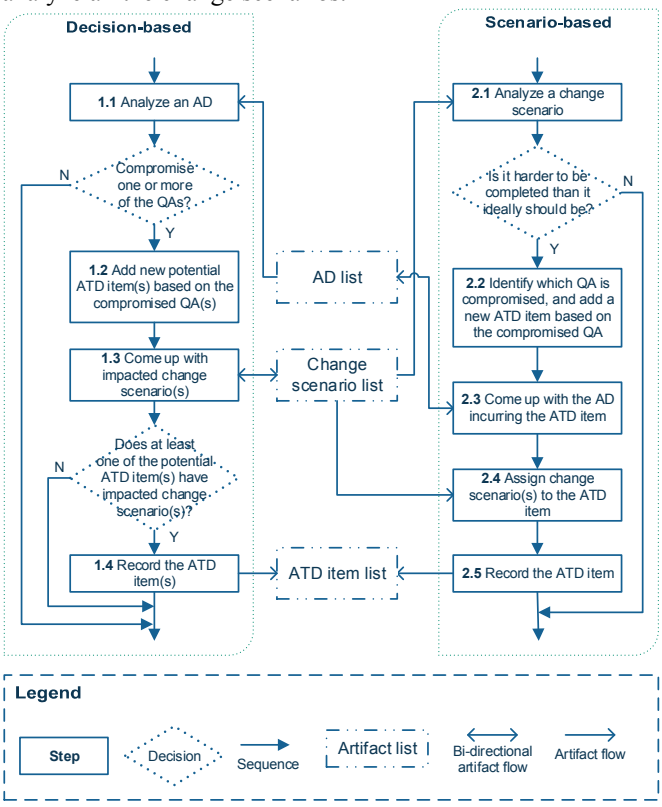


Fig. 2. Logical flow of the core of the ATD identification approach

As shown in Fig. 2, the proposed approach is comprised of two parts: a decision-based part and a scenario-based part. The two parts can be executed in an arbitrary order. It is not mandatory to execute both parts. Instead, users may choose to execute one of the two parts and skip the other one. Since new ADs and change scenarios may be added during the execution of the approach, it can be executed iteratively to recheck those ADs and scenarios until the iteration in which no new ADs and change scenarios are added. Note that the execution of the ATD identification approach requires the project members (the architects, project manager, and development team members) to work together in a workshop, physically or virtually.

1) Decision-based part

The decision-based part starts with analyzing ADs, in order to come up with ATD items incurred by the ADs, and related change scenarios that are impacted. Each AD is dealt with in four steps.

Step 1.1. The architects choose an AD from the AD list (which is prioritized as aforementioned), then check whether one or more of the six aforementioned QAs are compromised by the AD, according to its description (see in TABLE I). How to analyze ADs, is out of the scope of this paper (for example, the Decision-Centric Architecture Reviews (DCAR) method [18] can be employed).

Step 1.2. If none of the aforementioned QAs is compromised, the identification process ends. Otherwise, there are one or more potential ATD items identified. Each compromised QA corresponds to a new potential ATD item.

Step 1.3. The project members come up with change scenarios negatively impacted by the potential ATD items. If a change scenario is not in the change scenario list, then it must be added. If a potential ATD item does not have any impacted change scenario, this ATD item is not a true ATD item, thus should be dropped.

Step 1.4. The architects record all the ATD items resulting from Step 1.3 using the template in TABLE II, and add them to the ATD item list. This ATD item template is an adaptation based on the template in our previous work [1].

TABLE II. ATD ITEM TEMPLATE

ID	A unique identifier of this ATD item
Name	The name of this ATD item
Date	The date when this ATD item was identified or updated
Intentionality	Intentional or unintentional
Incurred by	The AD that incurs this ATD item
Compromised QA	The QA that is compromised (modularity, reusability, analyzability, modifiability, testability, or evolvability)
Rationale	The reason why the ATD item is incurred
Benefit	The value gained if the ATD item remains unresolved
Cost	The cost suffered by incurring this ATD item, which is the sum of principal and interest described below
Principal	The cost if this ATD item is resolved at the time when the ATD item is identified
Interest	The interest that this ATD item accumulates (the interest is the sum of the interest of all change scenarios described below)
Change scenarios	The change scenarios impacted by this ATD item, their consequences, scenario interest, and probabilities of the scenarios
Architecture diagram	A diagram or model that illustrates the concerned part (e.g., affected components) in the architecture design

2) Scenario-based part

The scenario-based part starts with analyzing change scenarios for collecting symptoms of ATD items, and then finds out the ADs that cause the ATD items. Each change scenario is dealt with in five steps.

Step 2.1. The architects choose one change scenario from the list (which is prioritized as aforementioned), and then the project members analyze whether the change scenario is more difficult to be completed than it ideally should be, i.e., whether a better (*ideal*) architecture design exists compared to the

current one (*as-is*). The difference between the *ideal* and *as-is* architecture designs is the root of ATD.

Step 2.2. If not, the scenario-based process of this change scenario ends. Otherwise, the project members identify which QA is compromised regarding the change scenario, and add a new ATD item.

Step 2.3. The project members (particularly the architects) come up with an AD that influences the change scenario regarding the compromised QA. This means that this AD incurs the ATD item. If the AD is not in the AD list, then it must be added.

Step 2.4. The project members check whether any other change scenarios in the list are negatively impacted by this ATD item regarding the compromised QA. If so, these scenarios must be assigned to the ATD item.

Step 2.5. The architects record the ATD item using the template provided in TABLE II.

C. Going beyond the State of the Art

This section describes how the proposed ATD identification approach addresses the shortcomings in the existing source code analysis-based ATD identification approaches as mentioned in Sections I (Introduction) and II (Related Work).

ATD types identified. Due to the diversity of ADs and change scenarios, the proposed approach can identify any type of ATD, both through the decision-based part and the scenario-based part. Typical ATD types are architecture smells, system-wide structure quality issues, and architectural compliance issues [3]. An example for the decision-based part is to identify ATD caused by the decision to implement most features of a software system for Windows platform first and then adjust the architecture to make the system portable to Mac OS and Android platforms. An example for the scenario-based part is to analyze a change scenario being delayed because it needs a new technology that the current architecture does not support. Both examples of ATD cannot be identified by source code analysis-based approaches.

When ATD identification can be applied. Due to the independence from source code, both the decision-based and scenario-based parts can be used during the early stages of the development life cycle, before ADs are implemented in code. This can potentially avoid substantial rework when paying the interest incurred by ATD, which is much more expensive when performed after the system implementation.

ATD confirmation. Source code analysis-based ATD identification approaches only provide indicators for ATD, but cannot confirm whether the identified potential ATD is actual ATD or not. The proposed approach relates an ATD item to the actual impacted change scenarios, which allows stakeholders to understand how the identified ATD item negatively impacts software development. Therefore it connects an ATD item to real cost in terms of the incurred interest of the impacted change scenarios. Therefore, stakeholders can make sure that the identified ATD is true ATD.

D. Known Limitations

One known limitation of the proposed approach is that some ADs may be forgotten and thus not recorded; this affects the coverage of the identified ATD items. If an AD that incurs a certain ATD item is not recorded, this ATD item may not be identified thus posing a threat to system maintainability and evolvability. Similarly, change scenarios that are missed during the change scenario collection phase also influence the coverage of the ATD items identified. However, the decision-based and scenario-based parts of the approach, to some extent, help mitigate this risk as one part can fill in the gaps of the other. On the one hand, the decision-based part may identify the ATD items that cannot be identified by the scenario-based part. If a change scenario that is negatively impacted by an ATD item, is not in the change scenario list, while the AD that incurs this ATD item is in the AD list, then this ATD item can be identified by the decision-based part through analyzing the AD. On the other hand, the scenario-based part may identify the ATD items that cannot be identified by the decision-based part. If an AD that incurs a specific ATD item is not in the AD list, while the negatively impacted change scenarios are in the change scenario list, this ATD item can be identified by the scenario-based part through analyzing the impacted change scenarios.

Another limitation of the proposed approach is that the coverage of identified ATD items depends on how well the users of the approach can analyze the ADs and change scenarios. This further depends on the expertise and experience of the users (i.e., the project members), as well as available time and effort.

IV. CASE STUDY

To evaluate the proposed ATD identification approach, we conducted a case study in a large telecommunications company in China. This case study was designed and reported by following the guidelines proposed by Runeson and Höst [19].

A. Study Objective and Research Questions

The goal of this case study, described using the Goal-Question-Metric approach [20], is: *to analyze* the proposed ATD identification approach *for the purpose of* evaluation *with respect to* its effectiveness and usability, *from the point of view of* practitioners *in the context of* industrial software development.

This goal results in two research questions (RQs), one for effectiveness (RQ1) and one for usability (RQ2). With the term "effectiveness", we are interested in three specific factors (see sub-questions of RQ1).

RQ1: How *effective* is the approach?

Rationale: This RQ concerns the extent to which the proposed approach can identify real ATD (RQ1.1), as well as whether the output of this approach can facilitate ATD-related activities that are closely related to ATD identification. In particular we are interested in two such activities: measuring ATD interest (RQ1.2) and optimizing the release planning of the project (RQ1.3).

- **RQ1.1**: How *useful* is the ATD identification approach for identifying ATD in the project?
  Rationale: The approach should be able to help practitioners find real ATD in their project.

- **RQ1.2**: How *appropriate* is the granularity of the change scenarios used in the ATD identification approach for ATD interest measurement in the project?
  Rationale: ATD interest indicates the risk of ATD to future development, and thus should be measured and made visible. Since the interest of an ATD item is affected by its associated change scenarios, the granularity of the change scenarios has an impact on the measurement of ATD interest.
- **RQ1.3**: How *helpful* are the identified ATD items to the release planning of the project?
  Rationale: The identified ATD items have a negative impact on future maintenance and evolution, thus should be taken into account in the release planning of the project. To ease future development, some ATD items should be resolved prior to specific maintenance and evolution tasks. This may lead to adjusting the release planning.

RQ2: How *easy* is it to execute the ATD identification approach for the project?
Rationale: The usability of the approach largely reflects the possibility that the approach will be adopted by users.

B. Case and Units of Analysis

This is an embedded single case study, since multiple units of analysis (i.e., the participants) are studied for one case: an information system in a large telecommunications company in China. The participants are the units of analysis instead of the identified ATD items, because we evaluated the proposed ATD identification approach through the analysis of the participants' feedback rather than the identified ATD items. The information system analyzes test data in various formats of telecommunications equipment and generates various types of reports about the quality of the tested telecommunications equipment. This system also provides the functionality of managing and controlling whether a piece of telecommunications equipment is allowed to proceed in a set of sequential tests. The system has a history of around seven years and a size of 760,000 lines of code in C#. Around 290 person-months of development effort has been invested in this project. Nine project members participated in the case study, including two architects (A1, A2), one project manager (M1), five developers (D1 – D5), and one tester (T1).

C. Data Collection

1) Data collected

To answer the RQs defined in Section IV.A, we collected the data items listed in TABLE III, which also lists the target RQ of each data item. In addition, we collected information about the study participants' experience in software industry (see TABLE IV).

2) Data collection methods

We used interviews as the main data collection method in this case study. As suggested in [19], interviews allow us to get in-depth knowledge about the topics of interest in the case study, by asking a series of questions about the interview topic to the study participants. Interviews can be categorized into unstructured, semi-structured, and fully structured [19]. In this case study we employed semi-structured interviews, which allowed us to adjust the order of the planned questions according to the development of the conversation between the researcher and the participants. Furthermore, semi-structured interviews allowed us to explore in more depth the interview topics by asking follow-up questions based on the participants' answers. We interviewed all nine participants.

TABLE III. DATA ITEMS COLLECTED

#	Data item	Scale	Range	RQ
DI1	Usefulness of the approach	Ordinal	Ten-point Likert. One for completely useless, ten for extremely useful	RQ1.1
DI2	The total number of ATD items identified	Ratio	Positive natural numbers	RQ1.1
DI3	The number of ATD items identified by the decision-based part	Ratio	Positive natural numbers	RQ1.1
DI4	The number of ATD items identified by the scenario-based part	Ratio	Positive natural numbers	RQ1.1
DI5	Appropriateness of the granularity of the change scenarios used in the ATD approach for interest measurement	Ordinal	Five-point Likert: too coarse, a bit too coarse but acceptable, appropriate, a bit too small but acceptable, too small	RQ1.2
DI6	Helpfulness of the ATD items identified by the approach to the release planning	Ordinal	Three-point Likert: unhelpful, insignificantly helpful, significantly helpful	RQ1.3
DI7	Ease of use in executing the approach	Ordinal	Ten-point Likert. One for completely unusable, ten for extremely usable	RQ2

TABLE IV. INFORMATION COLLECTED ABOUT STUDY PARTICIPANTS

#	Participant data item	Scale	Unit	Range
PDI1	Time the participants have worked in software industry	Ratio	Years	Positive natural numbers
PDI2	Time the participants have worked as developers	Ratio	Years	Positive natural numbers
PDI3	Time the participants have worked in the current company	Ratio	Years	Positive natural numbers
PDI4	Time the participants have worked in the domain of the case	Ratio	Years	Positive natural numbers
PDI5	Time the participants worked in their current role in software industry	Ratio	Years	Positive natural numbers
PDI6	Time the participants have been involved in the current project	Ratio	Years	Positive natural numbers
PDI7	Received dedicated training in SA	Nominal	n.a.	Yes or No
PDI8	Experience level in SA	Ordinal	n.a.	Five-point Likert scale[2]

[2] The five-point Likert scale: a) No knowledge on SA, b) Some knowledge on SA but never involved in architecting, c) Experience in architecting small software systems (<=50,000 lines of code), d) Experience in architecting big software systems (>50,000 lines of code), and e) Chief architect of big software systems.

Participant observations are a secondary data collection method used in the case study. As discussed in [19], participant observations are conducted in order to investigate how a specific task is performed by study participants. Participant observations can be conducted using different approaches. We conducted observations during the workshop to gather information about how the participants identify ATD using the decision-based and scenario-based parts of the proposed approach.

3) Data collection procedure

The procedure that we followed is comprised of five tasks and is shown in Fig. 3. Task2, Task3, and Task4 took place within a workshop in which a researcher (the first author) and nine participants attended. We briefly describe each task of the procedure below.

Task1: The architects recalled the ADs of the selected case following the guidelines provided by the researchers, and then recorded the ADs using the template in TABLE I. TABLE V shows one of the ADs that were recalled during Task1.

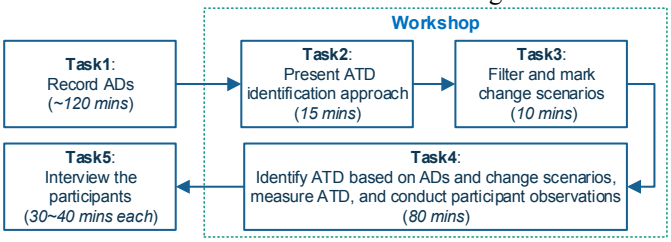
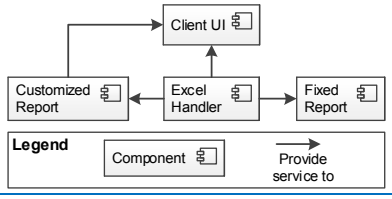


Fig. 3. Procedure of the case study

TABLE V. AN EXAMPLE ARCHITECTURE DECISION

ID	AD10
Name	Using pre-defined Excel templates for product quality reports
Description	We need to first design the formats and styles for different types of reports for different product lines and models, and then write the numbers on the product quality to the right cells of the right templates.
Alternatives	Excel automation: to generate reports and change report formats and styles by programming.
Rationale	We need to speed up the development to meet the tight deadline of the first release. Although we have already known Excel automation is a better solution, all the development team members currently are not experienced in Excel automation.
Pros	We can save considerable effort and time to deliver the first release of the system.
Cons	It will be time-consuming and error-prone to add new product models and product lines, as it needs to change the report formats of all report types. It is also hard to modify report styles since it needs the developers to manually change the styles of all the report templates. Also, considerable effort is required to validate the changes to the report templates.
Architecture diagram	

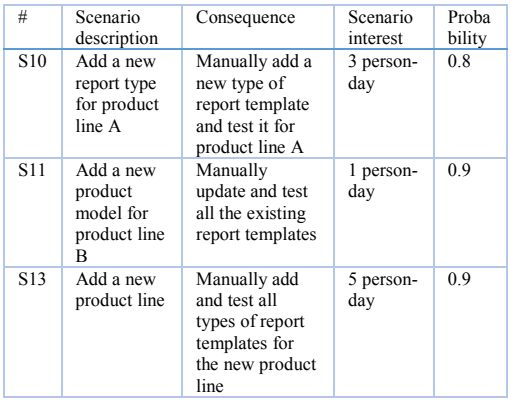
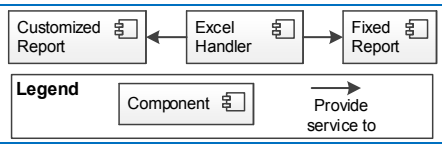
Task2: The researcher presented the ATD identification approach and related concepts to the participants in a tutorial.

Task3: The project manager provided the project's change scenario list that was maintained in an issue tracking system. Before the workshop, the manager had marked the change scenarios that have been delayed. The participants marked the change scenarios that are difficult to complete.

Task4: Based on the lists of ADs and change scenarios, the participants applied the proposed approach to identify ATD items. Specifically, the participants first used the decision-based part of the proposed approach and then the scenario-based part. Along the ATD identification, the chief architect recorded the ATD items using the template in TABLE II. An example ATD item is shown in TABLE VI. The participants also estimated the benefit and cost of each ATD item according to their expertise and experience in the project after the ATD item was identified. During this task, the researcher conducted participant observations on how the participants acted during this task.

Task5: After the workshop, we interviewed all nine participants with semi-structured interviews, asking a series of questions along the lines of the research questions. We interviewed the participants one by one, and each interview took 30 to 40 minutes.

TABLE VI. AN EXAMPLE ATD ITEM

ID	ATD1
Name	Poor support for report format and style customization
Date	22-08-2014
Intentionality	Intentional
Incurred by	AD10
Compromised QA	Evolvability
Rationale	To speed up the implementation of the feature of product quality reports, we decided to use the pre-defined Excel templates instead of Excel automation to set the formats and styles of the report files, since we did not have experience in Excel automation. We saved 15 person-days.
Benefit	15 person-days
Cost	32.8 person-days
Principal	25 person-days
Interest	3×0.8+1×0.9+5×0.9=7.8 person-days
Change scenarios	
Architecture diagram	

#	Scenario description	Consequence	Scenario interest	Probability
S10	Add a new report type for product line A	Manually add a new type of report template and test it for product line A	3 person-day	0.8
S11	Add a new product model for product line B	Manually update and test all the existing report templates	1 person-day	0.9
S13	Add a new product line	Manually add and test all types of report templates for the new product line	5 person-day	0.9

D. Data Analysis

We used descriptive statistics and qualitative analysis to examine the data collected during the case study. Descriptive statistics was used to analyze the participants' selections (e.g., the scores assigned to the usefulness of the proposed approach) that directly answered the interview questions, as well as the participants' professional experience information. Constant

comparison method [21], a well-established theory generation method in qualitative analysis [22], was used to analyze the data gathered in order to further explore the reasons behind the scores assigned and selections made by the participants.

E. Study Results

As shown in TABLE VII, all the participants had worked in software industry for seven or more years and as software developers for more than 5 years, except one who had 3.5 years of experience in software industry and as a developer. Four participants had experience in architecting big software systems (more than 50,000 LOC); two had experience in architecting small software systems (less than 50,000 LOC); while the others had basic knowledge on architecture only (without architecting experience).

TABLE VII. PARTICIPANTS' PROFESSIONAL EXPERIENCE INFORMATION

Participant	PD1: Years in software industry	PD2: Years as a developer	PD3: Years in the company	PD4: Years in the domain	PD5: Years in the current role	PD6: Years in the project	PD7: Dedicated SA training	PD8: Experience level in software architecture (SA)
A1	8	8	8	6	6	6	Y	Chief architect of big software systems
A2	9	6	9	2	5	2	Y	Experience in architecting big software systems
M1	10	5	13	7	7	7	Y	Experience in architecting big software systems
D1	7	7	7	5	5	5	N	Some SA knowledge but never involved in architecting
D2	8	8	5	3	7	3	N	Experience in architecting small software systems
D3	9	9	4	2	1	1	Y	Experience in architecting small software systems
D4	9	6	6	3	3	3	N	Some SA knowledge but never involved in architecting
D5	3.5	3.5	3.5	1.5	3.5	1.5	Y	Experience in architecting big software systems
T1	7	7	6	0.1	0.1	0.1	N	Some SA knowledge but never involved in architecting

1) Effectiveness (RQ1)

We report the results of RQ1 according to its three sub-RQs.

a) Usefulness (RQ1.1)

During the workshop, the participants identified 10 ATD items covering various aspects of the architecture, e.g., database, application server, obsolete .NET technology, external system dependency, and report generation mechanisms. Eight of these ATD items could not be identified by source code analysis. Out of the 10 ATD items, 6 were identified by the decision-based part of the approach, while the remaining 4 were identified by the scenario-based part. 20 ADs and 26 change scenarios were used in this case study.

In the interviews with the participants, we asked how useful they found the approach to be in identifying the ATD in the project. The results of this question are shown in TABLE VIII. The mean score is 8.2 out of 10. This score indicates that the approach is very useful in identifying ATD in this case. Eight participants gave scores no less than 8 while only one (D3) gave a score below 8.

TABLE VIII. USEFULNESS OF ATD IDENTIFICATION

Participant	A1	A2	M1	D1	D2	D3	D4	D5	T1	Mean
Score	8	9	8	8	8	7	9	9	8	8.2

Subsequently, we asked the participants follow-up questions to elaborate on their responses about usefulness. The participants expressed the following points:

- The architects believed that the proposed approach can identify most ATD items in the project. The chief architect (A1) thought that the most serious ATD items was in fact identified. He also argued that, the decision-based part of the approach is especially useful for identifying the ATD incurred by the ADs that intentionally compromise some of the QAs.
- The developers D2, D4, and D5 expressed their satisfaction with using the scenario-based part of the approach, since it successfully helped them identify several ATD items that the three developers had been struggling with.
- The developers D2, D3, D5 and the tester (T1) added that, the scenario-based part of the approach is more useful and easier to use for them since they are working on the scenarios in their daily work.

From the participant observations during the workshop, we found that ATD items that were incurred by ADs, which related to concrete functionalities, are more likely to be identified by the scenario-based part of the approach. In this case study, the decision-based part seems more useful in identifying ATD items that were caused by the fundamental ADs on the infrastructure of the software architecture. One reason is that some ADs related to concrete functionalities that had not been recalled during the AD recording stage.

Another observation is that the ATD items identified by the scenario-based part are more urgent to be resolved than the ones identified by the decision-based part. It was not a surprise. We found that, the change scenarios that were used to successfully identify ATD items using the scenario-based part, were the ones that had troubled the developers in the recent period. In contrast, most of the ATD items identified by the decision-based part were not so urgent but at a high level.

b) Appropriateness of change scenarios for ATD interest measurement (RQ1.2)

During the workshop, the participants also measured the benefit, principal, and interest of each ATD item after it was identified. We particularly paid attention to the impact of the change scenarios on interest measurement, since the interest of an ATD item reflects the risk of the ATD item on future development. In the interviews with the participants, we asked

how appropriate the granularity of the used change scenarios is for measuring the interest of the identified ATD items. The results of this question are shown in TABLE IX. Seven out of nine participants regarded that the granularity of the change scenarios was appropriate for measuring the interest of ATD items. Particularly, all the developers and the manager regarded that the granularity of the change scenarios was appropriate for interest measurement.

TABLE IX. APPROPRIATENESS OF THE CHANGE SCENARIOS USED TO INTEREST MEASUREMENT

Participant	Scenario granularity
A1	A little bit too small, but acceptable
A2	Appropriate
M1	Appropriate
D1	Appropriate
D2	Appropriate
D3	Appropriate
D4	Appropriate
D5	Appropriate
T1	A little bit too small, but acceptable

The participants further explained their perceptions on the granularity of the used change scenarios to interest measurement as follows:

- Architect A1 thought that the granularity of part of the change scenarios used in the case study is a little bit too small but acceptable. He admitted that change scenarios of smaller granularity may help to make a more accurate estimation of the interest of an ATD item. However, he postulated that estimating ATD interest based on change scenarios of relatively coarse granularity can reduce the needed time for the interest estimation and obtain acceptable accuracy of interest estimation at the same time.
- The manager considered the change scenario granularity as appropriate for ATD interest measurement, since he conducted daily project management based on the list of such change scenarios.
- The developers and architect A2 felt confident of measuring the interest of ATD items with the change scenarios used in the ATD identification, since the granularity of these change scenarios is the same as the granularity they handle in their daily work.

c) Helpfulness of the identified ATD items for release planning (RQ1.3)

TABLE X. HELPFULNESS OF THE IDENTIFIED ATD ITEMS TO RELEASE PLANNING

Participant	Option
A1	Significantly helpful
A2	Significantly helpful
M1	Significantly helpful
D1	Insignificantly helpful
D2	Significantly helpful
D3	Significantly helpful
D4	Significantly helpful
D5	Significantly helpful
T1	Insignificantly helpful

In the interviews, we asked the participants how helpful the identified ATD items are for release planning. As shown in TABLE X, seven participants (including the two architects and the project manager) considered that the identified ATD items are significantly helpful in their release planning. Two participants (D1 and T1) thought that the identified ATD items are helpful but not significantly for release planning.

2) Usability (RQ2)

In the interviews, we asked the participants how easy they found the approach to be. The results for this question are shown in TABLE XI. The mean score is 7.9 out of 10. This score indicates that the approach is relatively easy to use. Six participants gave scores no less than 8 while three gave scores below 8.

TABLE XI. USABILITY OF ATD IDENTIFICATION

Participant	A1	A2	M1	D1	D2	D3	D4	D5	T1	Mean
Score	8	9	8	6	7	6	9	9	9	7.9

The participants gave further remarks about usability during the interviews:

- Most participants considered that there were no significant difficulties in ATD identification using the approach.
- The developers D1 and D3 had difficulties sometimes in judging whether a development decision is an architectural one or not, as there is no clear boundary between ADs and other development decisions.
- The developer D2 argued that sometimes he was not very confident whether a maintainability issue is an ATD item or technical debt at a lower level.

From the participant observations during the workshop, we noted that participants from different roles acted differently when using the two parts of the approach (the decision-based and the scenario-based). The developers were more enthusiastic in identifying ATD items using the scenario-based part. The main reason is that each developer was familiar with part of the change scenarios that were used in this case study. Thus, the developers gave many remarks on specific change scenarios when they were analyzing change scenarios during the scenario-based ATD identification. We also observed that the developers tended to discuss technical details about the ATD items identified by the scenario-based part.

F. Threats to Validity

We discuss the threats to different validity types that are suggested in the guidelines of reporting case study research [19]. Internal validity is not discussed since we did not investigate causal relationships but only evaluate the ATD identification approach that we proposed.

Construct validity refers to the extent to which the studied operational measures really reflect the research questions (RQs) [19]. A frequent threat is that the data collected during a case study cannot effectively answer the RQs. To reduce this threat, we clearly defined the RQs and the data items (i.e., operational measures) that need to be gathered for answering each RQ in the study protocol and iterated this protocol multiple times. Furthermore we applied data source triangulation by gathering data both from interviews as well as participant observations. Another threat is that the participants may not have the same

understanding on the interview questions as the researchers'. To mitigate this threat, we piloted the data collection instruments with an architect from another company. The feedback from this architect helped us to significantly reduce the ambiguities in the interview questions.

External validity refers to the generalizability of the case study results [19]. The study results may be generalizable for software projects with similar project and team sizes. Since only one case is studied in one application domain (i.e., information systems), the results may not be generalizable to other application domains. Additionally, because the case study was conducted in China, the cultural context may have played a role in the results. Hence, replication of the study is desirable in other countries to validate the results in other cultural contexts.

Reliability is concerned with the data and the analysis being dependent on certain researchers [19]. To ensure the reliability of the study results, we defined a case study protocol in which we clearly defined the RQs, data items to be collected for the RQs, interview questions to collect the needed data items, and concrete operation steps. We also conducted a pilot study following the protocol with an architect from another company as mentioned before. Finally, even though the data analysis just involved descriptive statistics and simple data categorization, data analysis was performed independently by two authors and subsequently cross-checked for consistency.

V. Discussion

In this section, we interpret the study results as well as advantages and disadvantages of the proposed approach.

A. Interpretation on the Results of the RQs

Interpretation RQ1.1. The study results show that the proposed ATD identification approach is very useful considering that the mean score for usefulness is 8.2. Although the ATD items identified by the decision-based part of the approach are more than the ones identified by the scenario-based part in this case study, it does not mean that the decision-based part is more powerful in identifying ATD than the scenario-based part. In the case study, the decision-based part was used first and followed by the scenario-based part. Some of the ATD items identified by the decision-based part could also be identified by the scenario-based part.

As mentioned in Section III.D, the extent to which participants can recall ADs and collect change scenarios influences the coverage of the ATD items identified. In the interviews with the architects, they were confident that they have enough knowledge about the ADs to be able to find out the compromises made on system maintainability and evolvability. In addition, all the participants, particularly the developers and tester, considered that analyzing the change scenarios was not a difficult task for them.

Interpretation RQ1.2. The interest of an ATD item that is identified using the proposed approach, is relatively easy to measure, considering that most participants (7 out of 9) regarded the granularity of the change scenarios to be appropriate for ATD interest measurement. Since change scenarios are not necessarily independent from each other, it is possible that multiple change scenarios share a common part that needs to be modified to resolve the related ATD items. In this case, it is better to combine such change scenarios for interest measurement, in order to obtain a more accurate estimation. For the same purpose, a coarse change scenario should be decomposed if the change scenario is impacted by multiple ATD items with different compromised QAs.

Interpretation RQ1.3. The fact that most of the participants considered that the identified ATD items are significantly helpful to release planning, indicates that the identified ATD items are real problems influencing the future development of the project. The participants were already aware that there were some problems related to part of the identified ATD items. However, they did not have a good mechanism to characterize the problems, and thus did not carefully and specifically explore the actual impact of those problems on future development. As a result, those problems had been left unresolved though some participants were struggling with those problems. During the workshop, some participants expressed their appreciation for the introduction of the concept of ATD. As the developer D2 said, "*Finally, I find a good term to describe the problems that I have suffered from.*"

The proposed ATD identification approach does not only detect the location of an ATD item, but also identifies which change scenarios are negatively impacted by the ATD item. These change scenarios allow the participants to assess the real impact of the ATD item based on them (the scenarios). Therefore, the participants have enough confidence to determine whether a release plan needs to be adjusted according to the assessment of the impact of the identified ATD items.

Interpretation RQ2. The study results indicate that the approach is relatively easy to use, considering that the mean score of usability is 7.9. AD and ATD are the key concepts for using the approach. However, these two concepts had not been introduced in the project, and participants might not have thoroughly understood them, given that the researcher briefly introduced them in a 15-minute tutorial. This might have prevented the usability of the approach from being higher.

Before the ATD identification, the participants needed to recall ADs and collect change scenarios. All the participants thought that the change scenarios were relatively easy to collect, since all the scenarios had been recorded in an issue tracking system. For the ADs, there is no dedicated AD documentation in the project. The architects said that "*If ATDM is introduced in the beginning of the project, it would be easier to collect ADs; if ATDM is introduced after the project started, the preparation of ADs is not so easy since not all ADs can be recalled/recovered and it is hard to precisely recover the related information of all ADs, such as alternatives, rationale, and pros and cons*."

B. Advantages

- The architects of the project regarded that the decision-based part was particularly effective in identifying the ATD intentionally incurred by the ADs they made.

- All the project members (i.e., the project manager, architects, developers, and tester) agreed that the reuse of the existing well-recorded change scenarios saves much time and effort. In addition, they are very familiar with these change scenarios as they need to examine (part of) the scenarios in every development iteration.
- The development team members considered that the scenario-based part of the approach is particularly useful and easy to conduct, since they are familiar with the change scenarios and are even under pressure from delayed change scenarios that are negatively impacted by certain ATD items.
- The architects agreed that associating ATD items to concrete impacted change scenarios is helpful to estimate the interest of ATD items.
- Both architects argued that the decision-based and scenario-based parts of the approach are complementary to each other in terms of covering all ATD items in a software system.

C. Disadvantages

- There is no dedicated tool supporting the ATD identification process. Ideally a tool should be provided that is integrated with the issue tracking systems to synchronize with the change scenarios, and codifies ADs, ATD items as well as their traceability information. This could significantly reduce the documentation effort and save time in querying related information during ATD identification.
- Some users (e.g., new developers) may not be familiar with ADs and ATD items, though the templates provided in TABLE I and TABLE II may partially address this issue.
- The preparation of ADs may take time when the ADs are not documented, especially when the ADs were made a long time ago as the knowledge about ADs tends to vaporize.

VI. Conclusion and Future Work

We propose an approach to identify ATD based on architecture decisions and change scenarios. This approach provides the capability to identify ATD types that existing ATD identification approaches, mainly based on source code analysis, cannot identify. The results of the industrial case study show that the proposed approach is effective in identifying ATD. Specifically, the approach is useful in identifying real ATD, the granularity of the change scenarios used for ATD identification is appropriate for ATD interest measurement, and the identified ATD items are helpful to release planning. The case study results also show that the approach is easy to use. Our approach shows a high potential in identifying ATD that makes it visible to stakeholders.

We plan to replicate the case study in more industrial projects with different sizes and from various domains. We also plan to customize the approach in order to make it suitable for an agile context, with appropriate tool support.

References

[1] Z. Li, P. Liang, and P. Avgeriou, "Architectural debt management in value-oriented architecting," in *Economics-Driven Software Architecture*, I. Mistrik, R. Bahsoon, R. Kazman, and Y. Zhang, Eds., ed: Elsevier, 2014, pp. 183-204.

[2] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten*, et al.*, "Managing technical debt in software-reliant systems," presented at the Proceedings of the FSE/SDP workshop on Future of software engineering research (FoSER'10), Santa Fe, New Mexico, USA, 2010.

[3] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," *Journal of Systems and Software,* vol. 101, pp. 193-220, 2015.

[4] C. Seaman and Y. Guo, "Measuring and Monitoring Technical Debt," in *Advances in Computers* vol. 82, M. Zelkowitz, Ed., ed: Elsevier, 2011, pp. 25-45.

[5] P. Kruchten, "An Ontology of Architectural Design Decisions in Software Intensive Systems," in *2nd Groningen Workshop Software Variability*, 2004, pp. 54-61.

[6] B. Curtis, J. Sappidi, and A. Szynkarski, "Estimating the Principal of an Application's Technical Debt," *IEEE Software,* vol. 29, pp. 34-42, 2012.

[7] B. Curtis, J. Sappidi, and A. Szynkarski, "Estimating the size, cost, and types of Technical Debt," in *Proceedings of the 3rd International Workshop on Managing Technical Debt (MTD'12)*, Zurich, Switzerland, 2012, pp. 49-53.

[8] R. J. Eisenberg, "A threshold based approach to technical debt," *SIGSOFT Software Engineering Notes,* vol. 37, pp. 1-6, 2012.

[9] S. Wong, Y. Cai, M. Kim, and M. Dalton, "Detecting software modularity violations," in *Software Engineering (ICSE), 2011 33rd International Conference on*, 2011, pp. 411-420.

[10] C. Izurieta, A. Vetrò, N. Zazworka, Y. Cai, C. Seaman, and F. Shull, "Organizing the technical debt landscape," in *Proceedings of the 3rd International Workshop on Managing Technical Debt (MTD'12)*, Zurich, Switzerland, 2012, pp. 23-26.

[11] N. Zazworka, A. Vetro', C. Izurieta, S. Wong, Y. Cai, C. Seaman*, et al.*, "Comparing four approaches for technical debt identification," *Software Quality Journal,* pp. 1-24, 2013.

[12] P. Wang, J. Yang, L. Tan, R. Kroeger, and J. D. Morgenthaler, "Generating precise dependencies for large software," in *Proceedings of the 4th International Workshop on Managing Technical Debt (MTD'13)*, San Francisco, CA, USA, 2013, pp. 47-50.

[13] J.-L. Letouzey and M. Ilkiewicz, "Managing Technical Debt with the SQALE Method," *IEEE Software,* vol. 29, pp. 44-51, 2012.

[14] J. Brondum and L. Zhu, "Visualising architectural dependencies," in *Proceedings of the 3rd International Workshop on Managing Technical Debt (MTD'12)*, Zurich, Switzerland, 2012, pp. 7-14.

[15] R. L. Nord, I. Ozkaya, P. Kruchten, and M. Gonzalez-Rojas, "In search of a metric for managing architectural technical debt," presented at the Proceedings of the 10th Working IEEE/IFIP Conference on Software Architecture (WICSA'12), Helsinki, Finland, 2012.

[16] ISO/IEC, "Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models," in *ISO/IEC FDIS 25010:2011*, ed, 2011, pp. 1-34.

[17] M. Shahin, P. Liang, and M. R. Khayyambashi, "Architectural design decision: Existing models and tools," in *Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture. WICSA/ECSA'09*, 2009, pp. 293-296.

[18] U. van Heesch, V. Eloranta, P. Avgeriou, K. Koskimies, and N. Harrison, "DCAR - Decision-Centric Architecture Reviews," *Software, IEEE,* vol. 31, pp. 69-76, 2013.

[19] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering,* vol. 14, pp. 131-164, 2009.

[20] V. R. Basili, "Software modeling and measurement: the Goal/Question/Metric paradigm," University of Maryland 1992.

[21] B. G. Glaser and A. L. Strauss, *The discovery of grounded theory: Strategies for qualitative research*. New York: Aldine Publishing, 1967.

[22] C. B. Seaman, "Qualitative methods in empirical studies of software engineering," *Software Engineering, IEEE Transactions on,* vol. 25, pp. 557-572, 1999.