Analyzing the Impact of External and Internal Cohesion on the Change-Proneness of Web APIs

Daniele Romano, Maria Kalouda, and Martin Pinzger

Report TUD-SERG-2013-018
Analyzing the Impact of External and Internal Cohesion on the Change-Proneness of Web APIs

Daniele Romano
Software Engineering Research Group
Delft University of Technology
Delft, The Netherlands
daniele.romano@tudelft.nl

Maria Kalouda
Software Engineering Research Group
Delft University of Technology
Delft, The Netherlands
kalouda@student.tudelft.nl

Martin Pinzger
Software Engineering Research Group
University of Klagenfurt
Klagenfurt, Austria
martin.pinzger@aau.at

ABSTRACT
Several metrics have been proposed in literature to highlight change-prone software components in order to ease their maintainability. However, to the best of our knowledge, no such studies exist for web APIs (i.e., APIs exposed and accessible via networks) whose popularity has grown considerably over the last years. Web APIs are considered contracts between providers and consumers and stability is a key quality attribute of them.

We present a qualitative and quantitative study of the change-proneness of web APIs with low external and internal cohesion. First, we report on an online survey to investigate the maintenance scenarios that cause changes to web APIs. Then, we define an internal cohesion metric and analyze its correlation with the changes performed in ten well known WSDL APIs.

Our results provide several insights into the interface, method, and data-type change-proneness of web APIs with low internal and external cohesion. The results assist both providers and consumers in assessing the stability of web APIs, and provide directions for future research.

Categories and Subject Descriptors
D.2.8 [Software Engineering]: Metrics—product metrics

General Terms
Design, Measurement

Keywords
web api stability, change-proneness, internal cohesion, external cohesion

1. INTRODUCTION

Over the last years software systems have grown significantly from isolated software systems to distributed software systems (e.g., service-oriented systems) [30, 29]. These systems consist of interconnected, distributed components that are implemented and deployed by different organizations or different departments within the same organization. In these systems the distributed component’s API, referred to as web API throughout this paper, is considered a contract between the component’s provider and its consumers [14].

One of the key factors for deploying successful web APIs, and in general APIs, is assuring an adequate level of stability [14, 10, 45]. Changes in a web API might break the consumers’ systems forcing them to continuously adapt to new versions of the web API [31, 10]. For this reason, assessing the stability of web APIs is key to reduce the likelihood of continuous updates.

To reduce the effort and the costs to maintain software systems several approaches have been defined to identify change-prone software components [16, 39, 19, 35, 32]. Based on these studies software engineers can use quality indicators (e.g., software metrics, heuristics to measure antipatterns) that can estimate the components’ change frequency and can assist them in taking appropriate countermeasures. However, to the best of our knowledge, none of these studies investigates change indicators for web APIs. We believe that this is mainly due to the lack of publicly available web APIs with long histories which makes performing such studies challenging for researchers.

Change-proneness indicators would bring relevant benefits for both providers and consumers. On the one hand, consumers can estimate the change-proneness of suitable web APIs available on the market and subscribe to the most stable one. On the other hand, providers want to publish stable web APIs to reduce the maintenance effort and to attract more consumers and, consequently, increase their profits.

Among all the structural properties of web APIs (e.g., complexity and size), we believe that the cohesion can affect their change-proneness. Our intuition is based on our previous [37] and existing work [33, 34]. In our previous work [37], we showed that, among the existing source code metrics, the external cohesion has the strongest correlation with the number of changes performed in Java interfaces.

Moreover, Pereplechikov et al. [33, 34] showed that the cohesion can affect the understandability and, consequently, the maintainability of web APIs.

In this paper, to assist both providers and consumers, we use a mixed method approach [9] to analyze the impact of the internal and external cohesion on the change-proneness of web APIs. Internal cohesion measures the cohesion of the
operations (also referred to as methods) of a web API. External cohesion measures the extent to which the operations of a web API are used together by external consumers (also referred to as clients).

In the first part of our study, we use an online survey to investigate 1) the interface and method level change-proneness of web APIs with low external cohesion and 2) the interface and data-type level change-proneness of web APIs with low internal cohesion. The results show the likelihood with which maintenance scenarios can cause changes in web APIs affected by low internal and external cohesion.

The second part of our study consists of a quantitative analysis of the change-proneness of web APIs with low internal cohesion. We first introduce the Data Type Cohesion (DTC) metric to overcome the problem of the existing external cohesion metrics. Based on frequent discussions with industrial partners and colleagues, we believe that the existing metrics should be improved because they do not take into account the cohesion among data types. We then analyze the change-proneness of ten public WSDL1 (Web Service Description Language) APIs investigating the correlation between our DTC metric and the number of changes performed in the WSDL APIs. The results show that the values for the DTC metric are correlated with the number of changes the WSDL APIs undergo to.

The contributions of this paper are:
- insights into the likelihood of maintenance scenarios to cause changes in web APIs with low internal and external cohesion.
- a new internal cohesion metric that takes into account the cohesion among data types to highlight change-prone WSDL APIs.
- guidelines for researchers to investigate the method level of web APIs with low external cohesion and the data-type level of web APIs with low internal cohesion.

2. BACKGROUND

The concept of software cohesion has been widely investigated in the context of different programming paradigms [4, 8, 33, 23, 46]. In this paper we adhere to the classification defined by Perepletchikov et al. [33, 34] who investigated the cohesion of web APIs. According to their classification there are 8 different levels of cohesion involving web APIs: coincidental, logical, temporal, communicational, external, implementation, sequential, and conceptual. In this paper we focus on the external and communicational cohesion to which we refer as internal cohesion. The internal cohesion measures the cohesion of the operations (also referred as methods throughout the paper) declared in a web API. Similar to the method cohesion (i.e., LCOM) defined by Chidamber et al. [6, 7], the internal cohesion expresses the extent to which the operations belong together counting their common parameters. The external cohesion measures the extent to which the operations of a web API are used by external consumers (also called clients). In the next subsections, first, we present existing metrics proposed in literature to measure the external and internal cohesion. Then, we present the existing antipatterns in web APIs that result from low internal and external cohesion.

Figure 1: Example of a Multiservice web API (symptom of low internal cohesion). It exposes operations related to five different business entities (i.e., Order, Inventory, Invoice, Payment, and Credit).

2.1 Cohesion Metrics

To compute the external cohesion of a web API, Perepletchikov et al. [33, 34] proposed the SIUC (Service Interface Usage Cohesion) metric. This metric computes the sum of operations invoked by each client normalized by the number of clients and operations declared in the web API. To the best of our knowledge there are no further studies proposing other metrics for measuring the external cohesion.

Existing studies propose different metrics to measure the internal cohesion of web APIs. Perepletchikov et al. [33, 34] proposed the SIDC (Service Interface Data Cohesion) metric, Sindhgatta et al. [41] proposed the LCOIS (Lack of Cohesion in Service) and the SFCI (Service Functional Cohesion Index). Even though their formulas differ, these metrics have in common that they only measure the degree to which operations use common messages without considering the cohesion of messages. For this reason, in this paper, we refer to these existing metrics as message-level metrics.

2.2 Antipatterns

In literature different antipatterns for web APIs, and more in general for APIs, have been proposed [27, 40, 21, 5, 12, 26]. Among the proposed antipatterns two antipatterns in web APIs are symptoms of low internal and external cohesion: the Multiservice [12] and the Fat [26] antipatterns.

The Multiservice antipattern was originally conceived by Dudney et al. [12] and it is also known as God Object in literature [27]. A Multiservice web API exposes many operations that are related to different business entities. The CommerceAPI shown in Figure 1 is an example of a Multiservice API. This API exposes operations related to five different business entities: Order, Inventory, Invoice, Payment, and Credit. Such a web API ends up to be low internally cohesive because of the different entities encapsulated by it. As a consequence, many clients can invoke simultaneously its operations causing performance bottlenecks [27].

The Fat antipattern was proposed by Martin in [26]. This antipattern occurs in web APIs and other types of APIs, such as Java interfaces. A Fat web API is an API with disjoint sets of operations that are invoked by different clients and, hence, they show low external cohesion. The BankAPI shown in Figure 2 is an example of a Fat API. The Student and Professional clients invoke two disjoint sets of operations. Martin proposed the Interface Segregation Principle (ISP) to refactor such APIs. The ISP states that Fat APIs need to be split into smaller APIs according to its clients’

1http://www.w3.org/TR/wsdl
usage. Each smaller API should be specific to a client and each client should only know about the set of operations it is interested in.

3. RESEARCH QUESTIONS AND APPROACH

The change-proneness of web APIs is relevant to design and maintain large distributed information systems [28, 29]. To better understand the importance of assuring stable web APIs consider the scenario shown in Figure 3.

![Figure 2: Example of a Fat web API (symptom of low external cohesion). The Student and Professional clients invoke disjoint set of operations.](image)

In this scenario, a web API consumer (i.e., BankClient) wants to use a web API to receive payments from its customers. On the market there are three different providers (i.e., Provider1, Provider2, and Provider3) each providing a payment API (i.e., PaymentAPI1, PaymentAPI2, and PaymentAPI3) that adhere to BankClient’s business and functional requirements. BankClient is interested in a stable web API to reduce the need to adapt its system(s). Therefore, he decides to monitor the evolution of the three web APIs for a certain time. After this time he can use the most stable API (i.e., PaymentAPI2) with the lowest change frequency (i.e., 1 change per month).

In a real world scenario, where time-to-market is important for gaining competitive advantage, BankClient typically does not have the time to monitor the stability of different web APIs. Moreover, the number of past changes might not be available and they might not be a good indicator for future changes. For instance, an API might have been refactored to improve its change-proneness. Furthermore, from the perspective of providers, they are interested in providing stable web APIs to increase the likelihood with which clients subscribe to their APIs and, consequently, increasing their profits.

In this paper, we investigate the relationship between internal and external cohesion and the change-proneness of web APIs. The results can assist web APIs consumers and providers in estimating the stability of web APIs. In the following, we motivate and state our research questions, as well as, outline our research approach.

3.1 External Cohesion and Change-Proneness

Concerning web APIs with low external cohesion we want to investigate which scenarios are more likely to cause future changes in Fat web APIs. Moreover, we want to analyze the change-proneness of methods exposed in such web APIs. APIs with low external cohesion can have two different types of methods. Shared methods are methods invoked by all different clients. In Figure 4 requestInsurance() is a shared method since both the Student and Professional clients invoke it. Non-shared methods are methods invoked only by a specific client (e.g., the requestLoanForStudent() method in Figure 4). We believe that these two classes of methods can be changed for different reasons and knowing these reasons can give further insights into the change-proneness of web APIs with low external cohesion. To assist providers in evaluating the change-proneness of their web APIs with low external cohesion, we answer the following research question:

- **RQ1:** What are the scenarios in which developers change web APIs with low external cohesion? In which cases do they change the shared and non-shared methods?

We investigate the change-proneness on two different levels: interface level (i.e., change-proneness of a web API as a whole) and method level (i.e., change-proneness of the methods exposed by a web API). The results from this research question assist only providers because consumers typically do not have access to the information needed to measure the external cohesion (i.e., how other consumers invoke the API).

3.2 Internal Cohesion and Change-Proneness

Similar to external cohesion we investigate which scenarios are more likely to cause changes in Multiservice web APIs (i.e., web APIs with low internal cohesion). Furthermore, we analyze the change-proneness of the data types declared...
within a web API. This allows to highlight the differences between the change-proneness of shared data types (i.e., data types referenced multiple times within a web API) and non-shared data types (i.e., data types referenced only once). To evaluate the change-proneness of web APIs with low internal cohesion, we answer the following research question:

- **RQ2**: What are the scenarios in which developers change web APIs with low internal cohesion? In which cases do they change the shared and non-shared data types?

We investigate the change-proneness on two different levels: interface level (i.e., change-proneness of a web API as a whole) and data-type level (i.e., change-proneness of the data types declared in a web API). Differently to RQ1, the results from RQ2 assist both, providers and consumers. Both have access to the web API to measure the internal cohesion.

### 3.3 Internal Cohesion Metrics as Change Indicators

To make the results from RQ2 actionable in an industrial environment [3] a metric should be used to measure the internal cohesion. However, as shown in Section 2, the existing metrics are message-level metrics that do not consider the usage of data types to compose messages. To understand this drawback consider the two examples in Figure 5.

![Figure 5: Example that shows the drawback of existing message-level internal cohesion metrics SIDC, LCOS, and SFCI.](image)

The web API shown in Figure 5a exposes two operations operation1 and operation2 that use the same message message1. The message-level metrics are capable to detect the cohesion of this web API, however, fail when measuring the cohesion of the web API shown in Figure 5b. This API has two operations operation1 and operation2 that use different messages, namely message1 and message2. In this case the message-level metrics result in a low value of cohesion. However, message1 and message2 reference the same data types type2 and type3. We argue that the web API is cohesive because both, type1 and type4 (referenced by respectively message1 and message2), are complex data types composed of type2 and type3.

To overcome this problem Bansiya et al. [1] defined the CAMC (Cohesion Among Methods of Class) metric that measures the cohesion of object oriented classes. In this paper we adapt the CAMC metric for web APIs proposing the Data Type Cohesion (DTC) metric. For a web API s, DTC is computed as follows:

$$DTC(s) = \sum_{x,y \in Op(s)} C_o(x,y) \frac{|Op(s)|}{|Op(s)|}$$

where $Op(s)$ represents the set of operations exposed in s. $C_o(x,y)$ is the cohesion between two operations x and y, and it is defined as:

$$C_o(x,y) = \sum_{m,n \in MP(s)} C_{dt}(m,n) \frac{|MP(s)|}{|MP(s)|}$$

where $MP(s)$ is the set of all message pairs used by x and y; $C_{dt}(m,n)$ is the cohesion between the messages m and n computed as:

$$C_{dt}(m,n) = \frac{Com(m,n)}{Com(m,n) + Uncom(m,n)}$$

where $Com(m,n)$ represents the number of data types referenced by both messages m and n; and $Uncom(m,n)$ is the number of data types referenced only in one message.

To investigate quantitatively the change-proneness of web APIs with low internal cohesion we answer the following research question:

- **RQ3**: To which extent does the DTC metric highlight change-prone WSDL APIs? Which data types declared in a WSDL API are more change-prone?

Similar to RQ2 we investigate the change-proneness on two different levels: interface level and data-type level. The results from RQ3 are useful for both, providers and consumers, interested in measuring the internal cohesion in order to highlight change-prone web APIs and change-prone data types declared by them.

### 3.4 Research Approach

To answer our research questions we adopt a mixed method approach [9]. First, we answer RQ1 and RQ2 with a qualitative analysis consisting of an online survey. Then, following an exploratory sequential approach [9], we refine the results from RQ2 with a quantitative analysis aimed at answering RQ3. Note, we do not quantitatively refine the results from RQ1 because the needed information (i.e., how consumers invoke web APIs) are not available. We present the study, the analysis methods and the results of the qualitative and quantitative analyses respectively in Section 4 and Section 5.
In the second part, the questions are aimed at investigating the change-proneness of four SOA antipatterns (i.e., Fat, Multiservice, Tiny, and SandPile antipatterns). In this paper, we focus on and report the results about the Fat antipattern (RQ1) and the Multiservice antipattern (RQ2). We do not report the results about the Tiny and the Sand-Pile antipatterns because they are not symptoms of web APIs with low external and/or internal cohesion. In fact, they are symptoms of inadequate granularity [27] and subject of our future work.

In the third and final part of the survey, we asked participants to share their experiences with other design practices that can affect the change-proneness of web APIs and have not been covered by the survey. They are meant to draw directions for future work on this subject. Then, we asked questions to assess their prior knowledge about the antipatterns presented in the survey.

Before publishing our survey, we conducted three rounds of pilots with five software engineering researchers with a strong background in qualitative analyses. In each round we refined the survey questions and its structure based on their feedback. This step was necessary to attract participants in completing the survey. The complete survey is available on our website.²

In the following subsections, we first present information about the participants and their background and, then, we answer our research questions RQ1 and RQ2. For each research question we present the data used, the analysis method, and the results to answer it.

4.1 Participation

Our survey was opened on July 1st, 2013 and closed on July 31st. We forwarded the survey to our industrial partners and academic colleagues actively working on web API development. Moreover, we advertised it in google groups related to web APIs. During this time we collected responses from 79 participants among which 47 (59.5%) completed the entire survey answering all questions. Given that participants needed to answer 36 questions, investing on average approximately 40 minutes of their time, we consider it a good number of participants and a high rate of completion [42].

Among the 79 participants 44 work in industry, 30 in academia, and 5 in both academia and industry. Participants rated their background on a 5-point Likert scale ranging from absent, weak, medium, good, to strong. Participants, who answered the questions of the second and third part, have at least a good background in at least one of the following areas: service-oriented, cloud computing, WSDL APIs, and RESTful APIs. Few participants have an absent or weak background in any of these topics and quit the survey just after the background questions.

Interestingly, 72.7% of the participants answered that they do not know any metric/quality indicators to estimate the change-proneness of web APIs. The most common indicators used by the remaining 27.3% are the response time and information about the changes (e.g., number of changes between two versions, number of operations changed, etc.). This low ratio is an important indicator motivating our research.

4.2 External Cohesion and Change-Proneness

To investigate the answer to RQ1, we analyzed the change-proneness of web APIs with low external cohesion on two different levels: interface level and method level.

4.2.1 Interface Level Change-Proneness

Focusing on the interface level, we asked the participants to rank six scenarios that can lead a Fat web API to be changed. As discussed in Section 3, this antipattern is a symptom of web APIs with low external cohesion. Table 1 shows the list of scenarios. We derived them from our frequent discussions with our industrial and academic partners and colleagues. Furthermore, we asked the participants to state additional scenarios in a text box. For each scenario, 53 participants ranked the likelihood on a 5-point Likert scale: 0 (Won't change), 1 (Might change), 2 (Likely to change), 3 (Very likely to change), and 4 (Sure will change).

Table 1: Scenarios that cause changes in Fat web APIs (i.e., web APIs with disjoint sets of operations that are invoked by different clients indicating low external cohesion).

<table>
<thead>
<tr>
<th>Id</th>
<th>A Fat API is changed because ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fat₁</td>
<td>its clients have troubles in understanding it.</td>
</tr>
<tr>
<td>Fat₂</td>
<td>maintaining all the different operations is problematic.</td>
</tr>
<tr>
<td>Fat₃</td>
<td>it is a bottleneck for the performance of the system.</td>
</tr>
<tr>
<td>Fat₄</td>
<td>it should be split into APIs specific for each different client (i.e., Interface Segregation Principle [26]).</td>
</tr>
<tr>
<td>Fat₅</td>
<td>different developers work on the specific functionalities for the different clients.</td>
</tr>
<tr>
<td>Fat₆</td>
<td>if the functional requirements of a client change, the other clients will be affected as well.</td>
</tr>
<tr>
<td></td>
<td>test cases for all clients should pass before the API can be deployed.</td>
</tr>
</tbody>
</table>

We first used the non-parametric Kruskal-Wallis rank sum test [22] to analyze whether there is a difference between the scenarios to cause changes. Kruskal-Wallis tests whether samples originate from the same distribution comparing three or more sets of scores (i.e., the values of the 5-point Likert scale) that come from different groups (i.e., the different scenarios). We used the non-parametric Kruskal-Wallis test because the distributions of scores given by the participants are ordinal and non-normally distributed. Moreover, this test has been designed to compare three or more distributions in contrast to the non-parametric Mann-Whitney test [24] that compares two distributions. Performing the Kruskal-Wallis rank sum test among the scores given to the different scenarios resulted in a p-value < 0.01. This shows that the given scenarios cause changes to Fat web APIs with different probabilities.

The distributions of the scores given to the different scenarios are reported in Figure 6. To analyze these probabilities, we ranked the scenarios by the median and mean values. According to this ranking the Fat₁ is the most likely scenario with a median value of 3. This means that a Fat web API is very likely to be changed to reduce the amount of clones and ease maintainability. The second most likely scenario is Fat₄. According to its median score of 2, a Fat web API is likely to be changed to improve its understandability for the clients. The other 4 scenarios have median...
values equal to 1 indicating that they might force a Fat web API to be changed.

4.3 Internal Cohesion and Change-Proneness

Similar to RQ1, we answer RQ2 analyzing the change-proneness of web APIs with low internal cohesion on two different levels: interface level and data-type level.

4.3.1 Interface Level Change-Proneness

The first part of RQ2 aims at investigating scenarios that can cause changes in Multiservice web APIs. As discussed in Section 2, this antipattern is a symptom of web APIs with low internal cohesion. Similar to before, we provided the participants with seven scenarios to be ranked on the same 5-point Likert scale. Table 2 lists the seven scenarios stemming from discussions with our industrial and academic partners. Furthermore, we asked them to state additional scenarios in a text box. 51 participants ranked these scenarios.

Table 2: Scenarios that cause changes in Multiservice web APIs (i.e., APIs that expose many operations that are related to different business entities).

<table>
<thead>
<tr>
<th>Id</th>
<th>A Multiservice API is changed because ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>MS1</td>
<td>every business entity can change for different reasons (e.g., different evolving requirements). A new version should be published every time one of these entities changes.</td>
</tr>
<tr>
<td>MS2</td>
<td>changes to the API affect many clients (even though they do not use the changed business entity).</td>
</tr>
<tr>
<td>MS3</td>
<td>all the tests involving the different entities should pass before the entire web API is deployed.</td>
</tr>
<tr>
<td>MS4</td>
<td>the number of invocations to the Multiservice web API is high due to the different business entities.</td>
</tr>
<tr>
<td>MS5</td>
<td>proper pool tuning techniques are needed to achieve adequate performance due to the numerous clients.</td>
</tr>
<tr>
<td>MS6</td>
<td>different developers work on different business entities.</td>
</tr>
<tr>
<td>MS7</td>
<td>many business entities are exposed complicating the understanding of the API.</td>
</tr>
</tbody>
</table>

To analyze the results we followed the same approach used for analyzing the Fat web APIs' results before. First, we used the Kruskal-Wallis rank sum test to verify whether there is a statistical difference between the distributions of scores given to the different scenarios. The test resulted in a p-value < 0.01 indicating that these scenarios cause changes to Multiservice web APIs with different probabilities.

Then, we ranked the scenarios based on the median and mean values of their scores. The distributions of the scores given to the different scenarios are reported in Figure 7. The ranking shows that a Multiservice web API is very likely to be changed because of the different entities encapsulated by these web APIs (MS1). These changes can affect different clients even though they are not interested in the changed entity (MS2). Multiservice web APIs are also very likely to be changed to improve their understandability (MS3). Furthermore, the scenarios MS4, MS5, and MS6 are likely to cause changes.

To conclude and answer the first part of RQ2, we can state that Multiservice web APIs are very likely to be changed because: 1) every time it changes many clients are affected (MS1); 2) the web API can change for different reasons caused by the different entities (MS7); and 3) understanding
the web API is complicated for its clients (MS7).

4.3.2 Data-Type Level Change-Proneness

Addressing the second part of RQ2, we focus on analyzing the change-proneness of data types declared in Multiservice web APIs with low internal cohesion. In our survey, we asked participants to select which of the two classes of data types is more change-prone: shared data types (referred more than once in a web API) and non-shared data types (referred only once). Out of 48 participants who answered this question 30 (62.5%) found that non-shared data types are more likely to be changed, while 18 (37.5%) found that shared data types are more change-prone.

In addition, participants motivated their answers filling in a text box. Applying the card sort technique we manually clustered their motivations into two common groups of answers. On the one hand, 12 out of 18 participants stated that shared data types are likely to be changed because they are used by different messages and/or data types that can force them to change. In other words, they have multiple causes to change. On the other hand, 8 out of 30 participants stated that non-shared data types are more change-prone because developers prefer to share stable data types that represent generic business abstractions.

Similarly to the change-proneness of methods, the participants have two different opinions. However, they do not conflict and give two relevant insights into the change-proneness of data types:

- shared data types are changed when their operations evolve differently.
- otherwise developers tend to change non-shared data types because the impact of a change is lower.

5. Quantitative Analysis

The goal of the quantitative analysis is to provide an answer to RQ3, and consequently refine the results from RQ2. To reach this goal, we analyzed the correlation between the DTC cohesion metric and the number of changes performed in the different versions of ten public WSDL APIs. Table 3 lists the selected WSDL APIs from Amazon, eBay,

FedEx\textsuperscript{5} with their basic characteristics. WSDL is a standard interface description language used by many service-oriented systems to describe the functionality offered by a web API. We selected these WSDL APIs because they have sufficiently long histories as indicated by the increase in number of operations and data types. Furthermore, they have been used and discussed for similar studies in prior research [38, 15]. Even though a bigger data set is desirable, having access to WSDL APIs with long histories is not a trivial task. Most of them are used in a closed environment allowing access only to registered customers.

5.1 Interface Level Change-Proneness of WSDL APIs

For analyzing the change-proneness of the selected WSDL APIs we first computed the values for the DTC metric for each version of each WSDL API. Next, we extracted the changes between each pair of subsequent versions of a WSDL API. The changes were extracted using our WSDLDiff tool [38] that loads the specification of two versions of a WSDL API and compares them by using the differencing algorithm provided by the Eclipse EMF Compare plugin.\textsuperscript{6} In particular, WSDLDiff extracts the types of the elements affected by changes (e.g., Operation, Message, Data Type) and the types of changes (e.g., removal, addition, move, attribute value update). With this, WSDLDiff is capable of extracting changes, such as "a message has been added to an operation" or "the name of an attribute in a data type has been modified". We refer to these changes as fine-grained changes. Using WSDLDiff for each version of a WSDL API we counted the number of fine-grained changes that occurred between the current and previous version.

We used the Spearman rank correlation for computing the correlation between the values of the DTC metric and the number of changes. Spearman compares the ordered ranks of two variables to measure a monotonic relationship. We chose the Spearman correlation because it does not make assumptions about the distribution, variances and the type of the relationship [43]. A Spearman value (i.e., rho) of +1 and -1 indicates high positive or high negative correlation, whereas 0 indicates that the variables under analysis do not correlate at all. Values greater than +0.3 and lower than

\textsuperscript{5}http://www.fedex.com/us/web-services
\textsuperscript{6}http://www.eclipse.org/modeling/emf/

Table 3: WSDL APIs selected for the quantitative analysis showing the name (WSDL/API), the number of versions (Vers), the number of operations in the first and last versions (Ops), and the number of data types in the first and last version (Types).

<table>
<thead>
<tr>
<th>WSDL/API</th>
<th>Vers</th>
<th>Ops</th>
<th>Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>AmazonEC2</td>
<td>22</td>
<td>14-118</td>
<td>60-463</td>
</tr>
<tr>
<td>AmazonFPS</td>
<td>3</td>
<td>29-27</td>
<td>19-18</td>
</tr>
<tr>
<td>AmazonQueueService</td>
<td>4</td>
<td>8-15</td>
<td>26-51</td>
</tr>
<tr>
<td>AWSECommerceService</td>
<td>5</td>
<td>23-23</td>
<td>35-35</td>
</tr>
<tr>
<td>AWSMechanicalTurkRequester</td>
<td>6</td>
<td>40-44</td>
<td>86-102</td>
</tr>
<tr>
<td>eBay</td>
<td>5</td>
<td>156-156</td>
<td>897-902</td>
</tr>
<tr>
<td>FedExPackageMovement</td>
<td>4</td>
<td>2-2</td>
<td>15-15</td>
</tr>
<tr>
<td>FedExRateService</td>
<td>11</td>
<td>1-1</td>
<td>43-140</td>
</tr>
<tr>
<td>FedExShipService</td>
<td>8</td>
<td>1-7</td>
<td>74-166</td>
</tr>
<tr>
<td>FedExTrackService</td>
<td>5</td>
<td>3-4</td>
<td>29-33</td>
</tr>
</tbody>
</table>

Figure 7: Likelihood ranges from 0 (Won’t change) to 4 (Sure will change) for the scenarios causing changes in Multiservice APIs listed in Table 2.
Table 4: Results of the Spearman correlation analysis between the number of references and number of changes of data types. Bold values highlight significant correlations.

<table>
<thead>
<tr>
<th>WSDL API</th>
<th>p-value</th>
<th>rho</th>
</tr>
</thead>
<tbody>
<tr>
<td>AmazonEC2</td>
<td>0.248</td>
<td>-0.048</td>
</tr>
<tr>
<td>AmazonFPS</td>
<td>0.612</td>
<td>-0.104</td>
</tr>
<tr>
<td>AmazonQueueService</td>
<td>0.301</td>
<td>-0.130</td>
</tr>
<tr>
<td>AWSECommerceService</td>
<td>0.089</td>
<td>0.291</td>
</tr>
<tr>
<td>AWSMechanicalTurkRequester</td>
<td>0.000</td>
<td>-0.502</td>
</tr>
<tr>
<td>eBay</td>
<td>0.638</td>
<td>-0.015</td>
</tr>
<tr>
<td>FedExPackageMovement</td>
<td>0.005</td>
<td>-0.512</td>
</tr>
<tr>
<td>FedExRateService</td>
<td>0.000</td>
<td>-0.418</td>
</tr>
<tr>
<td>FedExShipService</td>
<td>0.000</td>
<td>0.193</td>
</tr>
<tr>
<td>FedExTrackService</td>
<td>0.000</td>
<td>-0.559</td>
</tr>
</tbody>
</table>

-0.3 indicate a moderate correlation; values greater than +0.5 and lower than -0.5 are considered to be strong correlations [17].

The result of the Spearman correlation analysis shows that the DTC metric has a significant and substantial negative correlation with a rho value equal to -0.361 (i.e., rho < -0.3) and with a p-value equal to 0.007.

Moreover, we computed the values of the existing message-level metrics (i.e., LCOS, SFCI, and SIDC) on the same WSDL APIs. We found out that their values are always 0 or 1. For instance, the value for LCOS is 1 in 62 out of 73 versions and 0 in 11 versions. Manually analyzing the WSDL APIs we noticed that this is due to their design. As shown by the example in Figure 5b messages reference different data types that are used as wrappers to isolate the data type declarations from the declaration of the operations and messages. For the WSDL APIs under analysis, this result confirms that existing metrics suffer from the problem explained in Section 3 and discussed in previous work [1].

We can conclude that DTC shows a substantial correlation, indicating that in increase in the internal cohesion is associated with a decrease in the number of changes.

5.2 Data-Type Level Change-Proneness of WSDL APIs

To detail these results, we investigated the change-proneness of shared and non-shared data types. For each data type in each version, we computed the number of times they are referenced in the WSDL API and the number of changes as extracted by our WSDLDiff tool. Next, we used Spearman to compute the correlation between these two metrics. Table 4 presents the results of this analysis.

Looking at the p-values of the correlation analysis, we note that significant results were obtained for 5 WSDL APIs (i.e., p-value < 0.01). Among them the values for three WSDL APIs show a strong correlation (i.e., rho < -0.5) and for one WSDL API they show a moderate correlation (i.e., rho < -0.3). These correlations indicate that the more a data type is referenced the less change-prone it is. Manually analyzing a sample set of shared data types, we found that they represent generic business entities or satellite data used by operations of the same domain. Hence, we assume that their requirements do not evolve differently. For instance, the ClientDetail in FedExShipService is a shared data type referenced on average 9 times by shipment operations that require information about the client. This data type encapsulates descriptive data about clients and it did not change across the releases. This result partially confirms the results of our survey namely: shared data types are change-prone if referenced by operations with different requirements, otherwise developers tend to change non-shared data types.

Based on these results, we can answer RQ3 stating that the DTC metric is able to highlight change-prone WSDL APIs. Moreover, we can partially confirm the insights of our participants about change-prone data types. However, to fully validate this result a bigger data set is needed. An ideal data set would consist of several WSDL APIs with long histories and from different domains or companies. This is needed to avoid that the results might be WSDL or company specific. Unfortunately, as already discussed, getting access to these artifacts is challenging.

6. DISCUSSION

In this section we summarize the results of our study, discuss the implications of the results and the threats to validity.

6.1 Summary of the Results

Summarizing the findings of our study, we found that Fat web APIs are very likely to be improved to reduce clones and ease maintainability and they are likely to be changed to improve understandability (RQ1). Multiservice APIs are very likely to be improved because such a web API declares different business entities and a change in one entity typically affects all the clients. Similar to Fat web APIs, Multiservice APIs are also affected by understandability issues (RQ2).

Analyzing the change-proneness of methods and data types we found that both shared messages and shared data types are likely to be changed if they are shared by clients and operations with different requirements (RQ1 and RQ2). For instance, if two clients with different requirements invoke the same operations, these operations change every time one of the two clients’ requirements change. Hence, they are more change-prone.

If modification tasks are not driven by clients’ or operations’ requirements then developers tend to modify non-shared operations and non-shared data types to keep the impact of a change low (RQ1 and RQ2).

To compute the internal cohesion and making the results of RQ2 actionable, useful metrics are needed [3]. This led us to introduce the DTC metric and to investigate its ability to highlight change-prone WSDL APIs. The quantitative study showed that DTC is able to highlight change-prone WSDL APIs. Moreover, we partially confirmed our survey participants’ insight: shared data types are change prone if they are referenced by operations with different requirements, otherwise non-shared data types are more likely to be changed (RQ3).

6.2 Implications of the Results

The results of this study are useful for web API providers, web API consumers, and software engineering researchers. Providers & Consumers. Both, web API providers and consumers, can benefit from a new internal cohesion metric (DTC) that overcomes the problem of the message-level metrics. Using DTC they can measure the internal cohesion to estimate the interface level change-proneness of WSDL APIs (RQ3). Based on the metric values, consumers can select and subscribe to the most stable web API that
shows the best internal cohesion, thereby reducing the risk to continuously update their clients to new web API versions. Providers can use DTC to identify the set of most change-prone web APIs (with low values for DTC) that should undergo a refactoring. For example, in case of a Multiservice API, the provider should consider splitting the API into different web APIs each one encapsulating a different business entity.

Providers. Furthermore, based on the values of the external cohesion metric they can estimate the change-proneness considering the maintenance scenarios likely to cause changes as suggested by our study. For instance, they can measure the SIUIC metric as proposed by Perepletchikov [33, 34] and refactor the web APIs with low values for external cohesion potentially affected by the Fat antipattern. They should refactor these APIs applying the Interface Segregation Principle described by Martin [26]. According to this principle, Fat APIs should be split into different APIs so that clients only have to know about the methods they are interested in.

Researchers. The results of this study are also valuable input to software engineering researchers. In this study we showed the impact of low external and internal cohesion on change-proneness of web APIs. As next step, researchers should investigate techniques for refactoring these kinds of web APIs. For instance, to the best of our knowledge there are no approaches able to apply the Interface Segregation Principle to refactor Fat APIs. Such an approach should mine the usage of a web API clients and, based on it, output the ideal sub APIs. This task is particularly challenging if a web API is invoked differently by many different clients.

In general, the results of this study are a precious input for researchers interested in investigating the change-proneness of web APIs. Each maintenance scenario that causes changes in web APIs should be further investigated to further assist web API providers.

6.3 Threats to Validity

In this study threats to construct validity concern the set of selected scenarios that we used to investigate changes in web APIs. This set is not complete. To mitigate this threat, we asked the participants of the survey to provide additional scenarios. Only three participants provided further scenarios. Hence, we cannot draw any statistical conclusion. Based on this result, we believe that we provided a good first set of scenarios that can be extended in future studies.

With respect to internal validity, the main threat is the possibility that the structure of the survey could have affected the answers of participants. We mitigated this threat by randomly changing the order of the scenarios for each participant. While this randomization worked for the scenarios, the threat stemming from the order of the questions in our survey remains - participants could have gained knowledge from answering the earlier questions that could have affected the answers to latter questions.

The threats to external validity have been mitigated thanks to our participants who work on software systems from different domains (e.g., banking systems, mobile applications, telecommunication systems, financial systems). Moreover, 18 participants are employed in international consulting companies with expertise in a wide range of software systems. Moreover, with regards to the quantitative analysis the set of WSDLs APIs should be enlarged in our future work to improve the generalization of the results. However, accessing WSDLs APIs with a long history is not an easy task. In fact, most of them are used in a closed environment and allowing access only to registered clients.

7. RELATED WORK

We identify three areas of related work: change-proneness, stability of APIs, and analysis of web APIs.

Change-proneness. Khoshgoftaar et al. [28] and Li et al. [25] were among the first researchers to investigate the impact of software structures on change-proneness. Khoshgoftaar et al. trained a regression model and a neural network using size and complexity metrics to predict change-prone components. The results show that the neural network is a stronger predictive model compared to the multiple regression model. Li et al. used the CkK metrics to predict maintenance effort improving the performance of prediction models. Girba et al. [16] defined the Yesterday’s weather approach to predict change-prone classes based on values of metrics and the analysis of their evolution. Di Penta et al. [32] showed that classes participating in antipatterns are more or less change-prone depending on the role they play in the antipattern. Khomh et al. [19] investigated the impact of code smells on the change-proneness of Java classes. Their results show that classes affected by code smells are more change-prone and specific smells are more correlated than others to change-proneness. Zhou et al. [47] examined the confounding effect of class size on the associations between metrics and change-proneness. They show that the size of a class is a relevant confounding variable to take into account to estimate its change-proneness. These studies represent a subset of existing work (e.g., [44, 13]) that underlines the importance of our research on providing indicators for highlighting change-prone software components. However, no study exists that investigates such indicators for highlighting change-prone web APIs.

Stability of APIs. The stability of APIs is a well known problem in the research community. A recent study by Vasquez et al. [45] shows that change-prone APIs negatively impact the success of Android apps. This work does not provide indicators for change-prone APIs but it shows the relevance of assuring an adequate stability of APIs. Recently, Raemaekers et al. [36] analyzed the stability of third parties libraries using four metrics to show how third parties libraries evolve. Hou et al. [18] analyzed the evolution of AWT/Swing APIs and their findings show that the majority of the changes is performed in the early versions. Dig et al. [11] analyzed four frameworks and one library finding that on average 80% of the API breaking changes are due to refactoring. Even though these studies show the relevance of investigating the stability of APIs there are few studies proposing metrics as indicators of change-prone APIs.

In our previous work, we investigated such indicators for APIs. In [39] we analyzed the impact of antipatterns on the change-proneness of Java APIs. The results show that APIs are more change-prone if they participate in ComplexClass, SpaghettiCode, and SwissArmyKnife antipatterns. In [37] we show that the external cohesion is the best performing metric to highlight and predict change-prone Java interfaces. Those studies were on Java APIs while the focus of this paper is on web APIs analyzing metrics and antipatterns specifically defined for web APIs.

Analyses of web APIs. In our previous study [38] we analyzed the evolution of four WSDL APIs. We proposed
the WSDLDiff tool to extract automatically fine-grained changes and we showed that it helps consumers in highlighting the most frequent changes in WSDL APIs. A similar analysis was performed by Fokaefs et al. [15] in 2011. They manually extracted the changes from the different versions of the WSDL APIs. Several antipatterns for web APIs have been proposed in literature, however, none of them has been investigated to indicate change-prone web APIs. Moha et al. [27] proposed an approach for specifying and detecting web API antipatterns. In their work they provide a complete and concise description of the most popular antipatterns.

Perepletchikov et al. [33] proposed five cohesion metrics, but an empirical evaluation of them for indicating change-prone APIs is missing. In their later study [34] they proposed three additional cohesion metrics and a controlled study. The results from this study show that the proposed metrics can help in predicting the analyzability of web APIs early in the software development life cycle, but not their stability. Our work is complementary to this existing work. Starting from the external and internal cohesion defined by Perepletchikov et al. and the antipatterns described in Section 2, we present a qualitative and quantitative study of using cohesion metrics to indicate the change-proneness of web APIs.

8. CONCLUDING REMARKS

Assuring an adequate level of stability of web APIs is one of the key factors for deploying successful distributed systems [14, 10, 45]. While consumers want to rely on stable web APIs in order to prevent continuous updates of their systems, providers want to publish high quality web APIs in order to prevent such updates and to stay successful on the market. Previous work has shown that the cohesion of an API is an indicator for understandability and stability [37, 33, 34]. In this paper, we extended this research to web APIs and investigated the relationship between internal and external cohesion and stability, measured as change-proneness.

We first presented an online survey to rank a number of typical maintenance scenarios to improve web APIs affected by the Multiservice and Fat antipatterns, both symptoms of web APIs with low internal and external cohesion. The results narrow down the many possible scenarios to two scenarios for Fat APIs and three scenarios for Multiservice APIs in which changes are very likely to occur. Focusing on internal cohesion, we detailed these results in a quantitative study with ten public available web APIs specified in WSDL. Results showed that the DTC metric is able to highlight change-prone WSDL APIs.

The results of our studies also open several directions for future work. Specifically, the method level and the data-type level change-proneness needs to be further investigated to better classify change-prone methods and data types. Furthermore, we plan to analyze the impact of granularity on the change-proneness of web APIs, for instance with the SandPile and Tiny antipatterns (both symptoms of APIs with inadequate granularity) [27].

9. REFERENCES


Romano, Kalouda, and Pinzger – Analyzing the Impact of External and Internal Cohesion on the Change-Proneness of Web APIs


