Keep Calm & Wait for the Spike! Insights on the Evolution of Amazon Services

Apostolos V. Zarras, Panos Vassiliadis, and Ioannis Dinos

Department of Computer Science and Engineering, University of Ioannina, Greece
{zarras, pvassil, idinos}@cs.uoi.gr

Abstract. Web services are black box dependency magnets. Hence, studying how they evolve is both important and challenging. In this paper, we focus on one of the most successful stories of the service-oriented paradigm in industry, i.e., the Amazon services. We perform a principled empirical study, that detects evolution patterns and regularities, based on Lehman's laws of software evolution. Our findings indicate that service evolution comes with spikes of change, followed by calm periods where the service is internally enhanced. Although spikes come with unpredictable volume, developers can count in the near certainty of the calm periods following them to allow their absorption. As deletions rarely occur, both the complexity and the exported functionality of a service increase over time (in fact, predictably). Based on the above findings, we provide recommendations that can be used by the developers of Web service applications for service selection and application maintenance.

Keywords: Software evolution, Web services, Lehman's laws

1 Introduction

Web services expose their functionalities through the Web, via application programming interfaces (APIs), which can be invoked by the client applications. Concerning software evolution, Web services are *black box dependency magnets*[1]. As application developers have no access to the internals of the services they use, they are clearly dealing with software modules of a black box nature. At the same time, even a small change in a Web service can reflect to a vast number of applications that use it. In particular, when a conventional API changes, the developers of the dependent applications can avoid dealing with the changes by sticking with an older version of the API. On the contrary, when a Web service changes, the evolution typically comes with a strict time plan, within which the developers of the dependent applications must migrate to the new version [4].

Understanding the evolution of Web services is therefore both difficult and important for application developers who need to know whether they can depend on the stability of the services they use and whether there are patterns and

[1] The term dependency magnet refers to a software module that is used by many others [13].

regularities concerning their evolution. Unfortunately, although existing research ([5], [15]) has provided valuable information on the statistical breakdown of the changes occurring to the services' interfaces, the understanding of regularities and patterns during the lifetime of services has not been investigated yet.

In this paper, *we focus on one of the most successful stories of the service-oriented paradigm in industry, i.e., the Amazon Web Services (AWS)*[2]. In this context, *we perform a principled empirical study that detects evolution patterns and regularities, based on Lehman's laws of software evolution* [3, 11]. Our findings indicate that *service evolution typically comes with spikes of change, during which operations are added or updated, followed by longer or shorter "calm" periods* that focus on internal improvements of service correctness, performance and security and allow the developers of Web service applications to absorb the changes. Typically, *the provided functionality increases in a predictable manner*, whereas, unfortunately, its *incremental growth is not predicable.* Based on our findings, we provide recommendations that can be used by the developers of Web service applications for service selection and application maintenance. In particular, evolution histories with calm periods between spikes are a desirable feature in service selection, as it allows the absorbtion of change. Thus, the study of the change history and functionality expansion can be used to attest on the suitability of a service during service selection. Concerning maintenance, functionality expansion can be predicted. At the same time, as the heartbeat of change itself is unpredictable, resources and time must be allocated to keep the applications up to date.

The rest of this paper is structured as follows: Section 2 discusses related work on software evolution; Section 3 provides the basic concepts of our approach, along with the setup of the empirical study; Section 4 details our method and findings; Section 5 discusses the practical implications of the study for the developers of Web service applications; Section 6 concerns threats to validity; finally, Section 7 summarizes our contribution and discusses future work.

2 Lehman's Laws & Related Studies

To come up with patterns and detailed insights in the evolution of Web services, we resort to traditional tools from the area of software evolution. Back in the 70's, Meir Lehman and his colleagues initiated their study on the evolution of software systems [3] and continued to refine and extend it for the next 40 years (e.g., [12, 11]). Lehman's laws concentrate on the evolution of *E-type systems*, i.e., software systems that solve a problem, or address an application in the real world. The essence of Lehman's laws is that *the evolution of an E-type system is a controlled process that follows the behavior of a feedback-based mechanism.* More specifically, the evolution process is driven by *positive feedback* that reflects the need to adapt to the changing environment, by *adding functionalities* to the evolving system. The growth of the system is constrained by *negative feedback*

[2] aws.amazon.com/

that reflects the need to perform *maintenance activities*, so as to prevent the deterioration of the system's quality. In [9], the authors provide a detailed historical survey of the evolution of Lehmans's laws. Further studies revealed the diverse behavior of software concerning the validity of Lehman's laws. In [16], for instance, the authors found evidence that commercial software is typically more faithful to the laws than academic and research software. A number of studies focused on open source software (e.g., [7, 10, 18, 8]), while in [17], we employed Lehman's laws to investigate the evolution of open-source databases; the common ground in all these studies is that they found support for the laws of continuing change and growth.

So far, the efforts that concern the evolution of Web services focus on classifications of changes, compatibility checks, version control and so on [1, 5, 6, 15]. [5] and [15] provide a first valuable insight on the evolution of real-world services. In both of these works the authors observed that the changes occurring to the services' interfaces are mostly additions and updates, while the deletions were relatively few. Another interesting empirical study on Web service evolution is reported in [4]. In this study the authors interview the developers of Web service applications to investigate the problems they encounter, due to the evolution of the services they use. Moreover, they investigate the evolution policies employed by the service providers. The findings of this study showed that different providers follow different practices and essential features like versioning are sometimes neglected. Moreover, the providers force changes upon the developers of Web service applications, so as to co-evolve with the services.

Going beyond the state of the art, in this paper we perform, for the first time in the related literature, a principled empirical study that exploits Lehman's laws of software evolution to provide detailed insights on the evolution of Amazon services. Based on our findings we further provide recommendations that can be used by the developers of Web service applications for service selection and application maintenance.

3 Basic Concepts & Setup

In this section, we discuss the basic concepts and the overall setting of our study.

3.1 Basic Concepts

We study the evolution of services that follow the standard Web services architecture[3] and expose their functionalities via WSDL specifications. The core concept of the data model that we employ in our study is the *service evolution history*, which provides information about the way that a service evolves.

Definition 1. ***Service evolution history*** - *The evolution history for a service, s, is a list, $H_s = \{r_1^s, r_2^s, \ldots, r_N^s\}$, that consists of the different releases of s. The elements of H_s are totaly ordered with respect to the their corresponding release dates.*

[3] www.w3.org/TR/ws-arch/

Definition 2. ***Service release*** - *A service release that belongs to the evolution history, H_s, of a service, s, is a tuple, $r_i^s = (ID, date, Size, Change)$ that consists of the following elements:*

- *ID is the release identifier that reflects the order of r_i^s in H_s.*
- *date, is the release date of r_i^s.*
- *Size, is a tuple of basic size metrics that concern different parts of the WSDL specification of r_i^s. Specifically, $Size[Interfaces]$, $Size[Opers]$, $Size[Types]$, denote the number of interfaces, operations, and XML types, respectively.*
- *Change, is a tuple of basic change metrics that concern the transition from r_{i-1}^s to r_i^s. In particular, $Change[Adds]$, $Change[Dels]$, and $Change[Upds]$, denote the number of operation additions, removals, and updates, respectively* [4]*. For the purpose of our study, we consider operations as composite elements, whose updates involve (a) changes in their own structure (e.g., attributes, annotations), or (b) updates in the structure of their constituents (e.g., messages, XML types).*

3.2 Amazon Web Services

Amazon is a major service provider that provides a variety of services, hosted on the AWS infrastructure. AWS is very popular, having customers such as NASA, NASDAQ, Netflix, Facebook, Adobe, D-Link, etc.[5]. In our study, we selected 6 Web services, for which it was possible to recover their detailed evolution history. Following, we provide further details regarding the functionalities of the examined services, while Table 1 gives information concerning the service releases that we considered.

Table 1. Description of the data-sets.

Dataset	**Releases**	**URL**
EC2	73	aws.amazon.com/ec2
ELB	14	aws.amazon.com/elasticloadbalancing/
AS	12	aws.amazon.com/autoscaling/
SQS	16	aws.amazon.com/sqs/
RDS	41	aws.amazon.com/rds/
MTurk	20	aws.amazon.com/mturk/

Elastic Compute Cloud (EC2) allows to reuse computational resources, via the allocation and management of virtual servers, deployed on the AWS in-

[4] We empirically observed that bindings and ports rarely change, while changes to XML types and messages are very strongly correlated with changes to operations (Spearman's correlation is typically close to 1). For lack of space, we omit these findings.

[5] aws.amazon.com/solutions/case-studies/all/

frastructure. Elastic Load Balancing (ELB) can be used together with EC2, to balance the load that is handled by a set of virtual servers, which have been allocated, via EC2. Auto Scaling (AS) provides means for scaling up, or down, a set of virtual servers that have been allocated via EC2. Simple Queue Service (SQS) allows message-based communication via queues. Relational Database Service (RDS) provides means for managing and using relational databases, over the AWS infrastructure. Mechanical Turk (MTurk) provides access to a scalable workforce, via an interface that offers operations for the creation of tasks, the qualification/selection of workers who are going to perform the tasks, the retrieval/approval of the work done, the payment of the workers, etc.

3.3 Release History Extraction & Assessment Method

To proceed with our study we calculated the evolution history for each one of the examined Web services. To this end, we exploited the public service release notes and the WSDL specifications that are available in the service provider's Web portal. To automate the evolution history extraction we used Membrane SOA[6] which allows to parse and compare WSDL specifications. We exported the evolution histories in the form of an Excel spreadsheet, from which we produced more advanced metrics and graphical representations of the data, which facilitated the assessment of Lehman's laws in our context (Sections 4 and 5). In our deliberations, we consider the latest definitions of Lehman's laws that are given in [11]. At a glance, the steps that constitute our assessment method are summarized below.

- For each law, we identify the criteria for the validation of the law in the context of Web services.
- Then, we evaluate these criteria in the case of Amazon services.
- We conclude on the validity of the law.
- Finally, we draw conclusions on what practitioners should conclude.

4 Findings

For each law, we discuss the main findings and we provide indicative graphical representations of the data for the most interesting cases. An extended report with all the results can be found in the Appendix.

4.1 Continuing Change (Law I)

"An E-type system must be continually adapted, or else it becomes less satisfactory in use" [11]. The intuition behind the first law is that as time goes by, both the operational environment and the users' needs change, causing the need for the system to change too [11]. In the service-oriented paradigm, as the services are publicly available through the Web, the number of the applications

[6] http://www.membrane-soa.org/soa-model/

that depend upon them and their diversity is potentially unlimited. Hence, the potentials for the emergence of new requirements and the need for changing the services can be very high. In our study, we assess the first law based on *the heartbeat of changes* that have been performed during the service evolution history.

The case of Amazon services: The study of the heartbeat of changes in the case of Amazon services reveals two main observations. The first one is that *for a large number of service releases, the service interface remains unchanged* (Figure 5). According to the service release logs, in these releases most of the activity concerns bug fixing, documentation, security and performance improvements, deployment extensions, provision of client side APIs for specific programming languages and environments. The second observation, which is inline with prior studies on the evolution of services [5, 15], is that *the overwhelming majority of changes concern additions and updates*; there are very few cases of deletions. Overall, *the essence of the law holds* for the examined services. Notable properties are that *changes are mostly internal and involve the structure of the exported operations less frequently; when they do, they involve mostly updates and additions.*

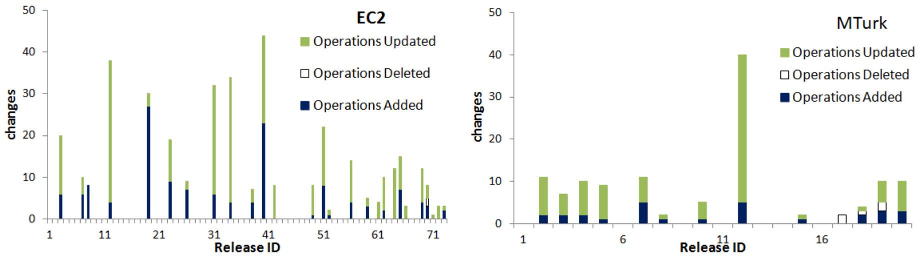

Fig. 1. Distribution of operation changes per release ID (releases with zero change also included).

4.2 Increasing Complexity (Law II)

"As an E-type system is changed its complexity increases and becomes more difficult to evolve, unless work is done to maintain or reduce the complexity" [11]. Software complexity is a vast concept that involves several aspects (Lehman et al. [11] refer to requirements and specification complexity, architecture complexity, design and implementation complexity, structural complexity, etc.) and metrics widely discussed in the literature (see e.g., [18] for a large number of complexity metrics). Addressing all these aspects in one study is simply not possible. Complexity from the viewpoint of the developers of Web service applications is mostly related to the effort required for (a) understanding, using and testing

the functionalities of the interfaces exposed by the employed services [14], and, (b) keeping the client applications up to date [2]. Regarding these aspects of service complexity, in the empirical study that we performed in [2], we found that the developers of Web service applications suggest the decomposition of fat interfaces, as they find them hard to understand and use. Based on this finding, in this paper we assess the second law with respect to the ratio of interfaces to operations. We consider that a particular service release is hard to understand, use and test if the provided interfaces consist of many operations. Following, we formally define the metric that we employ, with respect to the concept of service evolution history.

Definition 3. ***Interface Complexity*** **-** *For a service release r_i^s that belongs to the evolution history, H_s, of a service, s, we assess interface complexity in terms of the complement of the ratio of the provided interfaces to the operations offered by these interfaces, i.e., $C(r_i^s) = 1 - \frac{r_i^s.Size[Interfaces]}{r_i^s.Size[Opers]}$.*

The case of Amazon services: Regarding the second law, in all cases we observed that the value of $C(r_i^s)$ is generally high, which means that the interfaces of the examined services are composed of many operations (Figure 6). This holds, even for the initial releases of the services, where the value of $C(r_i^s)$ is typically higher than 0.9 for all datasets (except SQS). *As time passes, the value of $C(r_i^s)$ increases smoothly with a slow logarithmic trend.* To further validate this observation we performed a logarithmic regression analysis. In all cases except for SQS (Figure 6), the R^2 values that we obtained are high (from 0.576 to 0.928, 0.83 on average), indicating the logarithmic trend that we observed. *Notably, there is no evidence of the existence of a control mechanism for the aspect of complexity that we assess.* As we discussed in the case of the first law, the deletions of operations are very rare. Interface decomposition would be another possible way to reduce complexity (e.g., [2]). Nevertheless, for the examined services the number of provided interfaces is typically constant during the evolution history of the services. To sum up, *we conclude that the second law holds.* Specifically, the results brought out the following properties: *Interface complexity, measured in terms of the ratio of interfaces to operations, is high; it smoothly increases over time; usually the increase is logarithmic.*

4.3 Self Regulation (Law III)

"Global E-type system evolution is feedback regulated" [11]. The term regulation is used in the third law to emphasize that the system evolves in a controlled way, guided by (a) positive feedback activities that cause the system's functional capacity to grow, and, (b) negative feedback maintenance activities that resist to the unrestrained growth.

The validity of the third law is typically demonstrated by the existence of patterns in the incremental growth of a system [11, 18, 17]. Specifically, Lehman and his colleagues, observed ripples, which reflect the existence of a stabilization mechanism. Spikes indicate releases where the positive feedback activities that

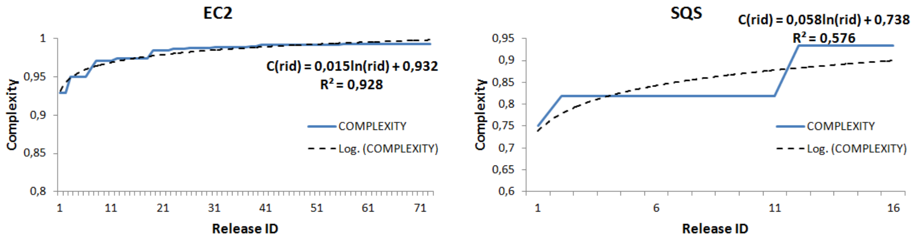

Fig. 2. Interface complexity - measured in terms of the ratio of interfaces to operations - per release ID.

grow the functional capacity of the system dominate, while valleys indicate releases of small or even negative growth, where most of the effort is spent for negative feedback maintenance activities. Further studies report similar observations in open source software [18] and database schemas [17]. In the aforementioned studies, the incremental growth of the system is measured as the size difference between two subsequent releases. The size of the system is typically measured with respect to the system implementation (e.g, number of modules, number of schema tables). To assess the third law, in the case of services we follow a similar track, by looking for patterns in the incremental growth of a service. In our study, we measure the incremental growth of a service in terms of the difference in the number of operations provided by subsequent service releases, as this is the actual increase in the functional capacity of the service that is perceived by the developers of Web service applications.

Definition 4. ***Incremental growth*** **-** *For a pair of subsequent service releases r_i^s, r_{i+1}^s that belong to the evolution history, H_s, of a service, s, the operations' incremental growth $IG_{op}(r_i^s, r_{i+1}^s)$ is the difference in the number of operations defined in the specification of r_{i+1}^s and r_i^s, i.e., $IG_{op}(r_i^s, r_{i+1}^s) = r_{i+1}^s.Size[Opers] - r_i^s.Size[Opers]$.*

The case of Amazon services: As the case of typical E-type systems, *in the incremental growth of Amazon services we also have spikes*. However, an interesting *difference* is that the *spikes* are usually sparse interrupted by *periods of calmness*, where the functional capacity of the Web service does not grow. MTurk appears to be an exception with consequent spikes, and only one short calmness period. In traditional E-type systems, negative feedback is concerned with correction actions, documentation improvement, dead code elimination, structural cleanups and restructurings, aiming to restrict uncontrolled change and its side effects. In the case of Amazon services, although we do not observe significant restructuring activities that are externally visible (resulting in an increased structural complexity – Law II), we can however claim that the growth that results from the positive feedback is not uncontrolled or continuous: on the contrary, we frequently see occasions where documentation improvements, bug fixing, security patching and extension of programming facilities take place (see Law I). In conclusion, *there is evidence that the third law holds.* Although there are no visible restructurings and consolidations, we observe *two patterns of incremental growth, specifically, spikes and calmness periods, which together indicate the existence of a stabilization mechanism.*

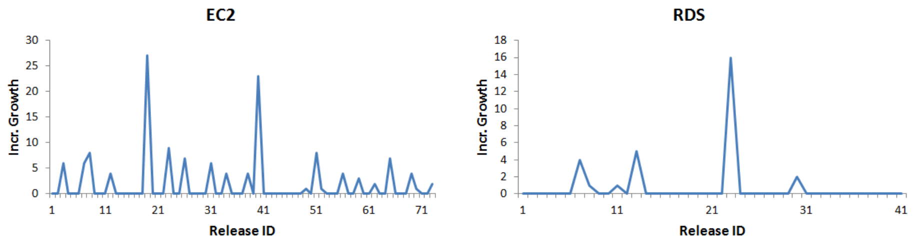

Fig. 3. Incremental growth –difference between the number of operations in subsequent service releases– per release ID.

4.4 Conservation of Organizational Stability (Law IV)

"The work rate of an organization evolving an E-type system tends to be constant over the operational lifetime of that system, or phases of that lifetime" [11]. In practice, a constant work rate, along with the patterns that control the self-regulated evolution of the E-type system, facilitate the planning of resources that are needed for the E-type system's evolution activities. Unfortunately, though, the assessment of the law has been problematic early on (see e.g., [12] or [18]), as the available information on indicators like personnel time dedicated to software evolution is typically unavailable and inaccurate. An approximation suggested by Lehman et al. [12] involves measuring number of changes performed per release.

The case of Amazon services: In all cases we observed that *the amount of changes is not invariant* during the lifetime of the Amazon services; on the contrary, *it may vary a lot* (Figure 5). Also, it is not possible to speak about phases in which the amount of changes remains constant. On the other hand, it is not possible to know precisely the work done behind the scenes (e.g., refactorings, repairs, etc.). Therefore, based on our results *we can not confirm or disprove the fourth law of software evolution.*

4.5 Conservation of Familiarity (Law V)

"The incremental growth of E-type systems is constrained by the need to maintain familiarity" [11]. Based on the observations of Lehman and his colleagues [12], the validity of the law is demonstrated by two factors that relate with incremental

growth. The first one (which is more related to the wording of the law) is that releases characterized by high incremental growth are followed by releases with lower incremental growth, thus, smoothening the process of understanding and mastering the performed changes. The second factor is that in the long term there is a declining trend in the incremental growth of the system, due to the increasing complexity of the system, which hardens the understanding of the changed context.

The case of Amazon services: We do not observe a clear declining trend in the incremental growth of the operations that are provided by the examined services (Figure 7). However, as pointed out in the case of the third law (Section 4.3), spikes in the incremental growth of the operations are typically followed by calmness periods of zero growth. Overall, we conclude that *the essence of the fifth law holds*. Specifically, the property that comes out from the results is that *releases characterized by non-zero incremental growth, tend to be followed by releases of zero incremental growth.*

4.6 Continuing Growth (Law VI)

"The functional capability of E-type systems must be continually enhanced to maintain user satisfaction over system lifetime" [11]. As discussed in detail in [11] the sixth law reflects the addition of new functionalities, while the first law generally concerns functional and behavioral changes. Based on the exact wordings of the law, its validity for a particular system involves the existence of a continuous increasing trend in the growth of the system. To assess the law in the case of Web services, we employ a metric typically used in the related literature [11, 18], the functionality growth. In our study, we measure the growth of the service as the number of operations provided by a particular service release. More formally, we employ the following metric.

Definition 5. ***Growth*** - *The growth of the operations $G_{op}(r_i^s)$ for a service release r_i^s that belongs to the evolution history, H_s, of a service, s, is defined as the number of operations provided by r_i^s, i.e., $G_{op}(r_i^s) = r_i^s.Size[Opers]$.*

The case of Amazon services: In all of the Amazon services we observed *an increasing trend in the growth of the service operations*. However, the periods of growth are *interrupted by periods of calmness*, consisting of subsequent service releases that offer the same number of operations (in Figure 9, the solid lines depict the actual growth, while the dashed and the dotted lines give growth predictions, discussed later in law VIII). Hence, *the results that we obtained indicate that the sixth law holds*. Nevertheless, the services *do not grow exactly as originally stated in the law*; *their functional capacity increases, but the increase is not continuous.*

4.7 Declining Quality (Law VII)

"The quality of an E-type system will appear to be declining, unless rigorously maintained and adapted to operational environment changes" [11].

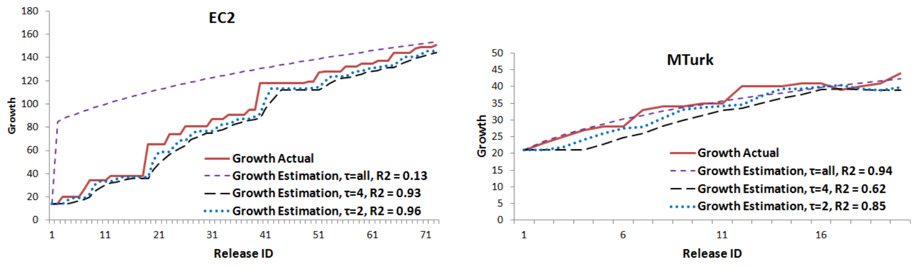

Fig. 4. Growth –number of operations– per release ID, and inverse square model predictions.

As discussed in [11] this law is closely related with the first and the sixth law, in the sense that the system must be adapted and extended, with respect to the evolving operational environment. Otherwise, it is likely that the provided functionalities will not be satisfactory for the users and the overall perceived quality of the system will downgrade. Regarding the assessment of the the seventh law, Lehman and his colleagues do not provide a concrete definition of quality. On the contrary, in [11] they state that the quality of an E-type system is a function of many factors, whose definition, modeling, measurement and monitoring depend on several aspects, which may include organization, product, process properties and goals. In the case of Web services, the main problem concerning the assessment of the seventh law is that the required data are typically not publicly available to the developer of Web service applications. Nevertheless, in [12] and [11], Lehman and his colleagues discuss a more general strategy to support the seventh law, which can be used in the case of Web services. Their strategy relies on logical induction, in the sense that the decline of the system quality, logically follows from the growth of the system's functional capacity (law VI), and from the increasing complexity that comes along with functionality growth (law II).

The case of Amazon services: So far, in our study we have evidence of the growing functional capacity of the examined services (Figure 9) that confirm the validity of the sixth law (Section 4.6). Moreover, we have some evidence of the increasing interface structural complexity of the examined services (Figure 6) that support the validity of the second law (Section 4.2). Therefore, by following the general strategy suggested by Lehman and his colleagues in [12, 11], we have indications that the seventh law holds for the examined services. However, given that the publicly available specifications of the examined services are not accompanied by concrete qualitative evaluations, *we cannot confirm or disprove the law, based on indisputable objective measurements.*

4.8 Feedback System (Law VIII)

"E-type evolution processes are multi-level multi-loop, multi-agent feedback systems" [11]. According to Lehman, the last law is a concise summary of the other

seven. In the literature, the main evidence for the validity of the law is to show that the actual growth of the system adheres to the inverse square (IS) model, which provides a *feedback-based growth prediction formula* [11, 18, 17]. In our study we also rely on this strategy. Following, we adapt the IS growth prediction formula, with respect to the definition of growth that we provided in Section 4.6.

Definition 6. ***IS model for services*** - *According to the IS model, the predicted operations' growth, $\widehat{G_{op}}(r_i^s)$, for a service release r_i^s that belongs to the evolution history, H_s, of a service, s, is: $\widehat{G_{op}}(r_i^s) = \widehat{G_{op}}(r_{i-1}^s) + \frac{\overline{E}}{\widehat{G_{op}}(r_{i-1}^s)^2}$, where $\widehat{G_{op}}(r_{i-1}^s)$ is the estimated operations' growth for the previous service release, r_{i-1}^s, and $\overline{E}$ estimates effort. More specifically, $\overline{E}$ is the average of individual E_j, calculated for the service release history H_s, as follows: $E_j = (G_{op}(r_j^s) - G_{op}(r_{j-1}^s)) * G_{op}(r_{j-1}^s)^2$, where $G_{op}(r_j^s)$ refers to the actual operations' growth for a service release r_j^s, and $G_{op}(r_{j-1}^s)$ refers to the actual operations' growth for the previous service release r_{j-1}^s.*

The case of Amazon services: To assess the eighth law we calculated the estimated values of the operations' growth, with respect to the IS model, and compared them with the respective actual values, derived from the evolution histories of the Amazon services. Specifically, we considered 3 variants of the model. The first variant, denoted by $\tau = all$, corresponds to the original IS model employed by Lehman in [12], where $\overline{E}$ is computed over the entire evolution history. The other two variants (inspired from [17]), compute an average effort taking only the recent past into consideration. Specifically, in $\tau = 2$ and $\tau = 4$, $\overline{E}$ is computed for every release r_i^s, over the previous 2 and 4 releases, respectively. To assess the quality of the estimated values, compared to the actual ones, we further calculated the values of the R^2 statistic for the IS model variants. Based on the results, we observed that $\tau = 2$ gives quite good estimations (R^2 ranges from 0.60 to 0.96, 0.78 on average). In fact, $\tau = 2$ gives the best estimations in all cases, but MTurk where $\tau = all$ outperforms the other two variants (Figure 9). So, overall, *we have evidence that the eighth law holds*; specifically, we can safely state that *the growth of the examined Web services can be accurately estimated via a feedback-based formula that exploits changes in previous service releases.*

5 Practical Implications & Recommendations

Our study revealed that Amazon services live quite normal lives. Although their loyalty to Lehman's laws can not be fully confirmed, the Amazon services are popular and constitute an integral part of a successful platform. Therefore, we consider the evolution patterns that we observed in our findings, as a baseline for a list of recommendations that target the developers of Web service applications and concern service selection and usage.

How can I tell if this service lives a healthy life? Check the change heartbeat of the service; calm lives consisting of frequent periods of calmness, where the functional capacity of a service does not change, interrupted by spikes of change that involve mostly additions and updates, indicate a normal life. During the calm periods, the service is typically enhanced in terms of correctness, documentation, security, performance and usability, showing that the service provider takes perfective maintenance seriously, performs bug fixing and improvements. Check the incremental growth of the service; again, the existence of spikes and calmness periods indicates that the service evolves normally, with respect to an underlying stabilization mechanism.

Will I have time to absorb changes? Check the incremental growth of the service; if you observe that releases of non-zero incremental growth, tend to be followed by releases of zero incremental growth, it means that there will be ample time to absorb the changes that take place.

Is the heartbeat of changes predictable in some way? Even for healthy services, it may not really be possible to forecast the heartbeat of changes that occur over time. Thus, you have to accommodate for resources for the worst case.

Will I have time to learn about new functionalities? Check the growth of the service; typically there is an increasing trend in the growth of the service operations. If you observe that the increase is not continuous, you can count on the interval for the understanding of the new features.

Is the amount of new functionalities predictable in some way? It is likely that you could coarsely predict the expansion of the offered operations and plan accordingly; try to do this via a feedback-based regression formula that is based on Lehman's IS model.

Will the complexity of the service be a problem for service usage? Even for healthy services, complexity could be an issue. You have to assess the specific aspects of complexity that concern you (e.g., specification, architectural, structural, etc.). The complexity could be quite high and it may increase over time. However, if the increase is smooth and predictable, you do not have to worry too much. In any case, you will have to anticipate the need to allocate time and resources for understanding and using the service.

Will the quality of the service improve, decline, remain as is? Most likely you wont be able to tell, based on available information; first you will have to determine the aspects of quality that concern you, then you you will have to assess them by yourself. Anticipate the need to allocate time and resources for QoS evaluation.

6 Threats to Validity

Regarding *external validity*, our study focused on the in-depth analysis of Amazon services. We studied the evolution of Amazon services that provide various functionalities. The population of the examined services is reasonable, with respect to similar studies [5, 15, 18, 17]. More importantly, we considered services that have already made a notable impact in industry. Hence, we are confident that our findings are representative of the overall population of Amazon services and that they are of interest to a broad community of developers. Nevertheless, the reader should be careful not to overgeneralize the results to the overall population of existing Web services. At the same time, our approach for the assessment of Web service evolution is general and can be used to perform further similar studies. Also, the recommendations that we provide for service selection and usage are general and concern the overall population of Web services.

When it comes to *construct validity*, we used Membrane SOA, a well-known open-source API, for the accurate construction of evolution histories. Moreover, we manually inspected random samples of the collected data. *Internal validity*, is not a major issue in our study as we do not attempt to establish any particular cause-effect relationships. Regarding *conclusion validity*, we validated the observed relations and trends with well-known statistic methods.

7 Conclusion

From a broader perspective, we believe that the success of the service-oriented development paradigm strongly depends on whether services live a normal life. Understanding the patterns and regularities that rule the evolution of services is a key factor for increasing the developers' confidence on the services that they use, or intend to use. In this paper, we studied the evolution of Amazon services. To perform our study we followed a principled approach that is based on Lehman's laws of software evolution. Although our findings showed that Amazon services are healthy, this cannot be taken for granted for all services. To this end, developers of Web service applications can exploit our principled approach, to assess the health of the services they are interested in. Based on our study, we further derived a list of practical recommendations that target the developers of Web service applications and concern service selection and application maintenance.

Regarding the future directions of this line of research, two open issues of significant practical importance are the forecasting of service evolution, and the study of the relationship between the evolution of services (e.g., number, size, frequency of spikes) and the applications that use them (e.g., number of clients, usage profile).Metrics for service complexity and growth that account for more factors (e.g., input/output parameters) is also a research issue that deserves to be further investigated. Finally, another interesting challenge is to introduce a principled patterns-based method that allows the developers of Web service applications to assess the healthiness of the Web services' that they use.

Acknowledgments. We would like to thank the reviewers of the paper for their helpful comments. This work was supported from the European Community's FP7/2007-2013 under grant agreement number 257178.

References

1. Andrikopoulos, V., Benbernou, S., Papazoglou, M.P.: On the Evolution of Services. IEEE Transactions on Software Engineering 38(3), 609–628 (2012)

2. Athanasopoulos, D., Zarras, A., Miskos, G., Issarny, V., Vassiliadis, P.: Cohesion-Driven Decomposition of Service Interfaces Without Access to Source Code. IEEE Transactions on Services Computing 8(4), 550–562 (2015)
3. Belady, L.A., Lehman, M.M.: A Model of Large Program Development. IBM Systems Journal 15(3), 225–252 (1976)
4. Espinha, T., Zaidman, A., Gross, H.G.: Web API Growing Pains: Loosely Coupled Yet Strongly Tied. Journal of Systems and Software 100, 27–43 (2015)
5. Fokaefs, M., Mikhaiel, R., Tsantalis, N., Stroulia, E., Lau, A.: An Empirical Study on Web Service Evolution. In: Proceedings of the 18th IEEE International Conference on Web Services (ICWS). pp. 49–56 (2011)
6. Fokaefs, M., Stroulia, E.: Using WADL Specifications to Develop and Maintain REST Client Applications. In: Proceedings of the 22nd IEEE International Conference on Web Services (ICWS). pp. 81–88 (2015)
7. Godfrey, M.W., Tu, Q.: Evolution in Open Source Software: A Case Study. In: Proceedings of the 16th IEEE International Conference on Software Maintenance (ICSM). pp. 131–142 (2000)
8. Herraiz, I., Robles, G., Gonzalez-Barahona, J.M., Capiluppi, A., Ramil, J.F.: Comparison Between SLOCs and Number of Files As Size Metrics for Software Evolution Analysis. In: Proceedings of the 10th IEEE European Conference on Software Maintenance and Reengineering (CSMR). pp. 206–213 (2006)
9. Herraiz, I., Rodriguez, D., Robles, G., Gonzalez-Barahona, J.M.: The Evolution of the Laws of Software Evolution: A Discussion Based on a Systematic Literature Review. ACM Computing Surveys 46(2), 1–28 (2013)
10. Koch, S.: Software Evolution in Open Source Projects: a Large-scale Investigation. Journal on Software Maintenance and Evolution 19(6), 361–382 (2007)
11. Lehman, M.M., Fernandez-Ramil, J.C.: Software Evolution and Feedback: Theory and Practice, chap. Rules and Tools for Software Evolution Planning and Management. Wiley (2006)
12. Lehman, M.M., Fernandez-Ramil, J.C., Perry, D.E.: On Evidence Supporting the FEAST Hypothesis and the Laws of Software Evolution. In: Proceedings of the 5th IEEE International Software Metrics Symposium (METRICS). pp. 84–88 (1998)
13. Martin, R.C.: Clean Code. Prentice Hall (2009)
14. Perepletchikov, M., Ryan, C., Tari, Z.: The Impact of Service Cohesion on the Analyzability of Service-Oriented Software. IEEE Transactions on Services Computing 3(2), 89–103 (2010)
15. Romano, D., Pinzger, M.: Analyzing the Evolution of Web Services Using Fine-Grained Changes. In: Proceedings of the 19th IEEE International Conference on Web Services (ICWS). pp. 392–399 (2012)
16. Siebel, N.T., Cook, S., Satpathy, M., Rodríguez, D.: Latitudinal and longitudinal process diversity. Journal of Software Maintenance 15(1) (2003)
17. Skoulis, I., Vassiliadis, P., Zarras, A.: Open-Source Databases: Within, Outside, or Beyond Lehman's Laws of Software Evolution? In: Proceedings of the 26th International Conference on Advanced Information Systems Engineering (CAiSE). pp. 379–393 (2014)
18. Xie, G., Chen, J., Neamtiu, I.: Towards a Better Understanding of Software Evolution: An Empirical Study on Open Source Software. In: Proceedings of the 25th IEEE International Conference on Software Maintenance (ICSM). pp. 51–60 (2009)

8 Appendix

In this section, we provide the detailed results concerning the evolution of the Amazon services that we considered in our empirical study.

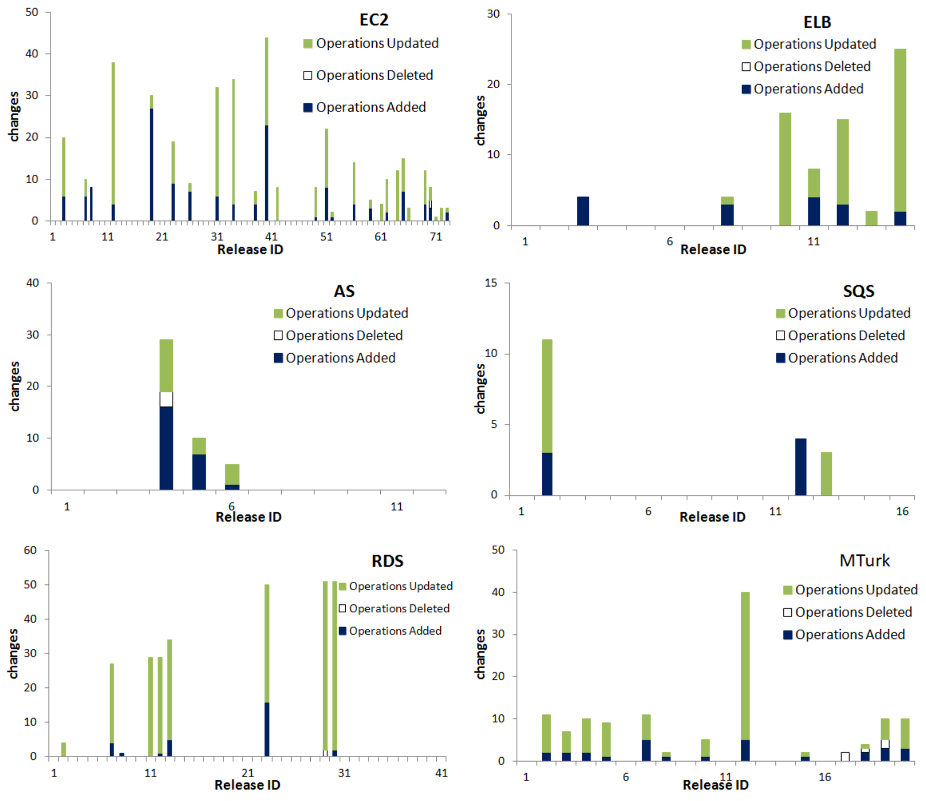

Fig. 5. Distribution of operation level changes per release ID (releases with zero change also included).

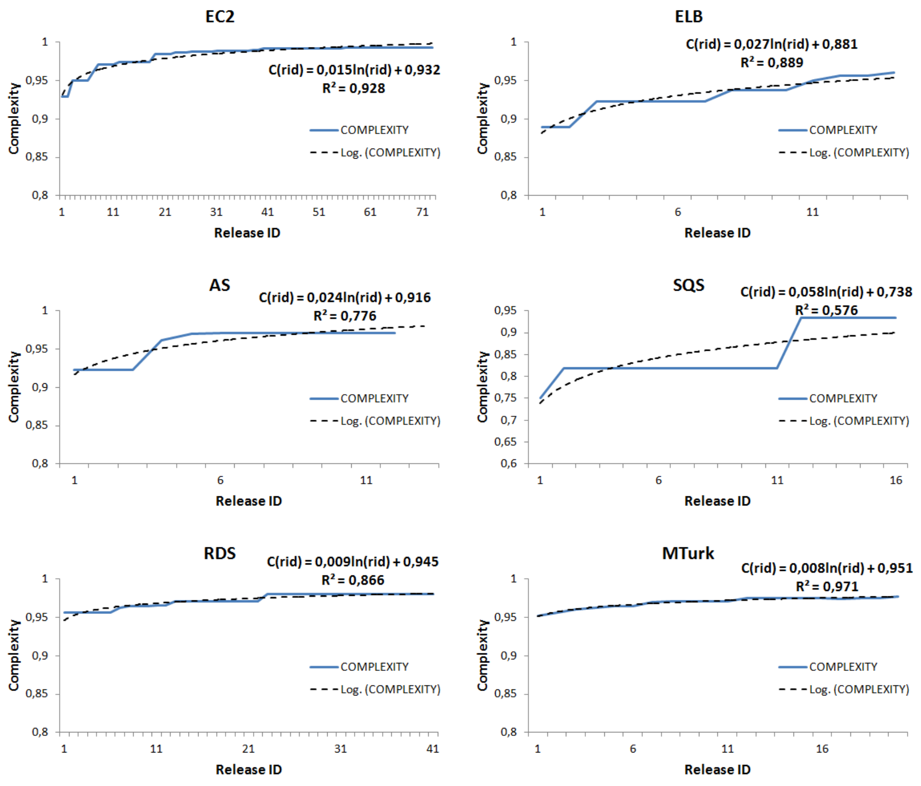

Fig. 6. Complexity of service interfaces per release ID.

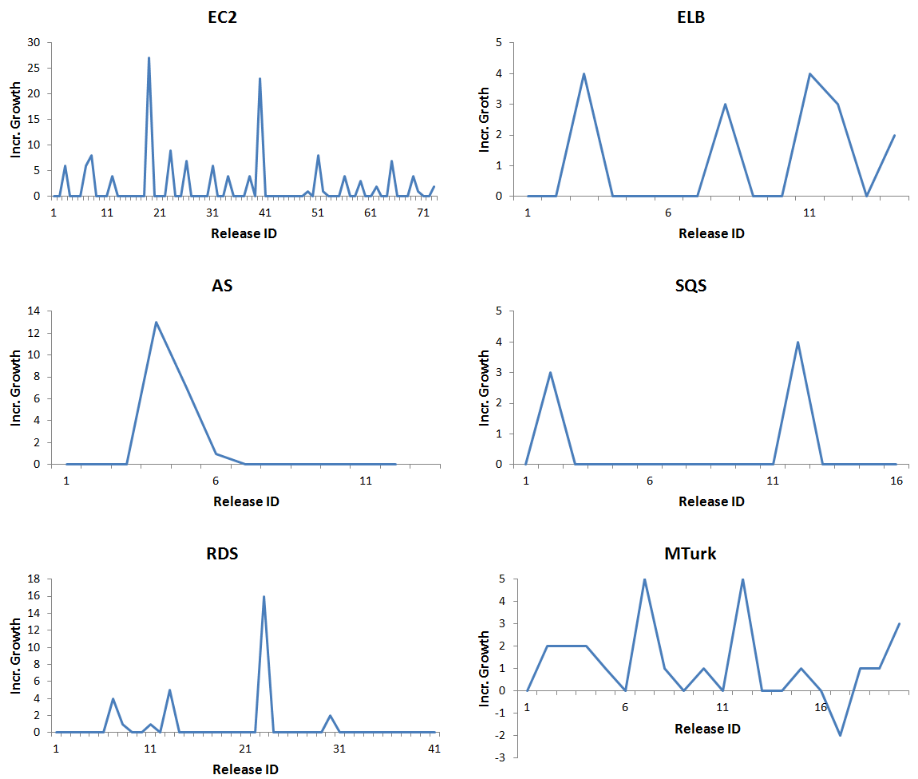

Fig. 7. Incremental growth –difference between the number of operations in subsequent service releases– per release ID.

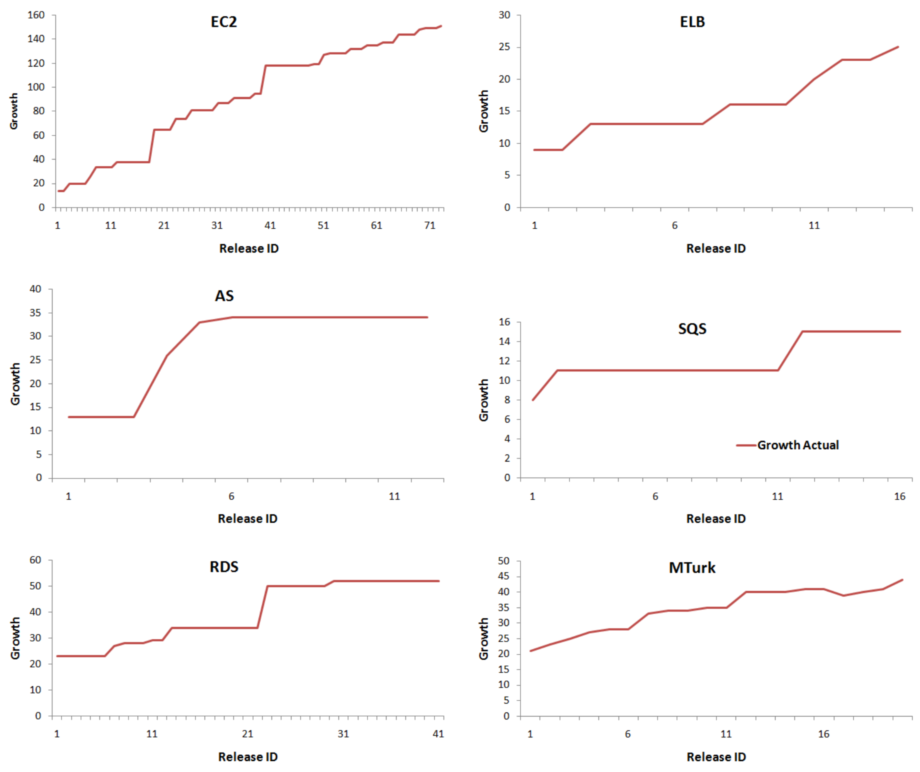

Fig. 8. Growth –number of operations– per release ID.

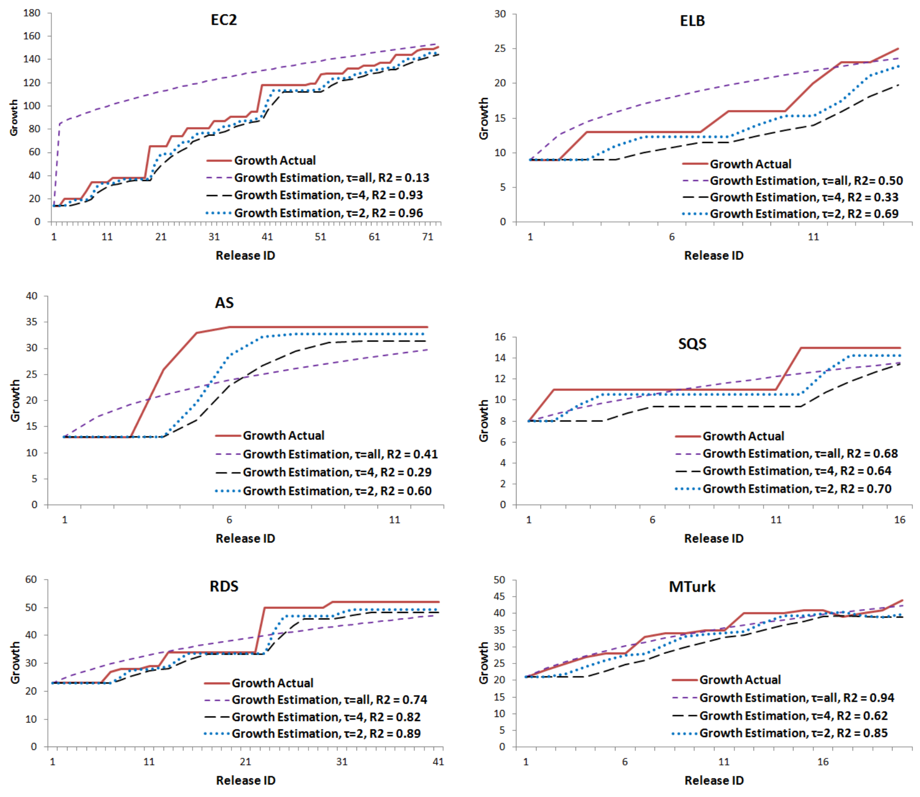

Fig. 9. Growth –number of operations– per release ID, and inverse square model predictions.