11-2014

An ecological model for digital platforms maintenance and evolution

Paolo ROCCHI
Guido Carli University

Paolo SPAGNOLETTI
Guido Carli University

Subhajit DATTA
Singapore Management University, subhajitd@smu.edu.sg

An Ecological Model for Digital Platforms Maintenance and Evolution

Paolo Rocchi, Paolo Spagnoletti and Subhajit Datta

P. Rocchi & P. Spagnoletti, CeRSI-LUISS Guido Carli University, Rome, Italy

P. Spagnoletti, e-mail: pspagnoletti@luiss.it

P. Rocchi, IBM, Rome, Italy e-mail: procchi@luiss.it

S. Datta, Singapore University of Technology and Design, Singapore, Singapore e-mail: subhajit_datta@sutd.edu.sg

Published in Organizational Innovation and Change, 2016, pp. 263-280, Lecture Notes in Information Systems and Organisation 13, DOI: 10.1007/978-3-319-22921-8_21

Abstract:

The maintenance of software products has been studied extensively in both software engineering and management information systems. Such studies are mainly focused on the activities that take place prior to starting the maintenance phase. Their contribution is either related to the improvement of software quality or to validating contingency models for reducing maintenance efforts. The continuous maintenance philosophy suggests to shift the attention within the maintenance phase for better coping with the evolutionary trajectories of digital platforms. In this paper, we examine the maintenance process of a digital platform from the perspective of the software vendor. Based on our empirical observations, we derive an interesting statistical relationship that has strong theoretical and practical implications in the study of software defects.

Keywords: Software maintenance, Wakeby, Digital platform, Complex systems

This paper has been awarded the “Special Award Sandro D’Atri” at the XI Conference of the Italian Chapter of AIS held in Genova (IT) on November 21st–22nd 2014.

1 Introduction

The dynamics of organizational emergence together with the evolutionary trajectories of digital infrastructures are challenging the traditional practices for managing innovation and blurring the boundaries between strategic, structural and technological choices [1]. This is particularly true when digital platforms are in place for supporting interactions across multiple sets of actors and, among them, of software developers that contribute to platform evolution [2]. This has been the case for instance of the Internet [3, 4] but also of applications, platforms and information infrastructures owned by private companies that strategically exploit the generativity of digital technologies [5–7].

Many companies (i.e. SAP, Google, Facebook, Apple, etc.) have implemented “third-party developer” strategies and encouraged their business partners, customers or independent developers to come on board their computer platforms [8]. This is also the case of those subjects who develop a software product (i.e. digital platform) and make it available to the community of users together with its source code, application development interfaces (API), software development kits (SDK) and technical documentation. Such new forms of online collaboration increase the speed of improvement and evolution of software products and challenge existing methods for software design and maintenance.

The aim of this paper is to investigate digital platform evolution processes in order to identify new methods for guiding the emergence of complex socio-technical systems. Instead of considering software maintenance as a recovery activity whose costs must be reduced adopting sophisticated methods and techniques, we propose a shift towards a continuous maintenance philosophy. An exploratory case study on the evolution of four versions of a large-scale middleware product, shows that patterns of bugs and fixes fit into an ecological model.

The paper is organized as follows: in the next section we present a literature review on digital platform evolution. Then we discuss the case study data collection and analysis. In the following section we highlight our observations and results, followed by the derivation of a statistical relationship based on our results. We conclude with a discussion and a summary of the results.

2 Related Works

In order to better position our contribution in the existing literature it is worth to illustrate how digital platforms and their maintenance processes have been studied so far. First of all we clarify the distinction between evolutionary and static software systems. Second, we introduce digital platforms as a particularly relevant form of evolutionary software systems. Third, we summarize how development and maintenance processes and methods have been studied in the software engineering and management information systems literature [9, 10].

Static software systems, are intended as computer programs whose acceptability on completion only depends on satisfying, in the mathematical sense, of formal specifications. On the other hand evolutionary systems, must undergo continual evolution to remain satisfactory and operate or address a problem or activity in the real world [11]. Therefore, to remain satisfactory, these programs must be continually changed and updated. The acceptability of evolutionary software systems depends on the results delivered to users and other stakeholders. They must be continually enhanced, adapted and fixed if they are to remain effective within an evolving application environment. Thus, the evolution of such systems is a complex phenomenon being characterized by multi-level, multi-loop, and multi-agent feedbacks.

In this paper we focus on digital platforms, a particular type of evolutionary software systems. In general, a platform is defined as a building block, providing an essential function to a technological system, which acts as a foundation upon which complementary products, technologies, or services can be developed [2, 12–14]. Therefore, digital platforms differ from other software systems in that their design context is not fixed a priori. They have a heterogeneous and growing user base and allow a constant generification of new IT capabilities [3, 15, 16]. In more practical terms, digital platforms allow extensive recombination and reuse of software programs, subroutines, services, features, and content. This generativity is achieved through the deployment of APIs, documentation, debuggers, source code examples, and integrated development environments [8, 17, 18].

For managing these platforms, the traditional values and goals of information systems development practices are challenged and the notion of continuous change emerges as a new paradigm [19]. This implies that continuous analysis, negotiated requirements, and a large portfolio of continuous maintenance activities must replace lengthy analysis and design, user satisfaction, abstract requirements, complete and unambiguous specifications, and projects in the management of emergent organizations. An attempt to implement these principles is represented by agile requirement engineering practices that have gained an increasing attention in the last decade [20–23]. These methods heavily rely on feedbacks collected from the users during the development phase and their purpose is to improve software quality. However agile methods are still focused on minimizing the maintenance efforts during the operational lifecycle of a software system and hence they do not fully embrace the philosophy of continuous maintenance.

Previous studies on software maintenance processes have looked at the phenomenon from different perspectives [24]. For instance, some authors have analyzed the maintenance processes of an ERP

software package from the perspective of the customers organizations [25] and have compared them with existing standards (IEEE/EIA 12207.0 maintenance-process standard) [26]. Other studies have focused on the dynamics of community maintenance contributions enabled by the Internet and the volunteer workforce [27]. We adopt the perspective of the software vendor for contributing to a better understanding of how to guide the emergence of digital platforms in complex settings.

3 Research Strategy

An exploratory case study is conducted for investigating the evolution of a digital platform from the perspective of the software vendor. The research design is based on a single case with four embedded units of analysis [28]. The single case provides the typical context of a software vendor in charge of the continuous maintenance process of a software product during its operational lifecycle. Large scale empirical studies of maintenance data present several challenges. In fact, defect data are not often diligently recorded, and are seldom published for proprietary systems. Moreover, since the software vendor is a leading multinational company the single case allows us to conduct a revelatory case from a privileged observation point.

The four embedded units of analysis are represented by four different releases of the same software product, a middleware application that is deployed worldwide among more than 5000 large customer organizations. The middleware product (XYZ) provides services for monitoring the performance of IT resources, including disks, CPU, and applications. The XYZ helps to automatically detect bottlenecks, and potential resource problems, and act on them proactively.

The XYZ is particularly relevant with respect to our purposes for two main reasons. First, being an infrastructure level software it operates at an intermediate layer between multiple digital devices configurations and multiple applications depending on the IT infrastructure of each customer organizations. This makes the system subject to a huge variety of external input. Second, XYZ can be considered as a software platform in that it provides an environment for the design of new resource models and gives customers the possibility to develop their own monitoring agents.

3.1 Data Sources and Analysis Methods

Empirical data were collected through direct contact with the head of the maintenance team that gently provided us with archival data on software bugs and fixes, information on the maintenance process, technical documentation, and commercial information. A dataset with more than 2,200 defect reports over a four year period represents the main source of data on which the following analysis is based.

Our study investigates changes to the four releases (or versions) of XYZ: B.1; B.2; B.2.1, B.2.2. Release B.1 derives from a product developed by a company acquired by our focal software vendor, which was delivered without further changes. Later the focal software vendor made significant economical investments and release B.2 derives from an effort to optimize and improve XYZ; B.2.2 represents a second major enhancement, based partially on customer feedback.

Users of XYZ who find unexpected behavior such as adverse incidents and bugs, write requests for change (RFC) thus the acronym RFC will be used interchangeably with “defect” or “error” in this paper. An RFC does not demand functional changes; users raise another type of request, which we shall call “suggestions (SUG)”, in order to vary or add a function. A single SUG proposes new or modified functionalities.

Defects and suggestions are recorded in a special database. The data is captured and grouped according to releases. Each release is maintained as an independent entity; thus some failures can repeat and others are unique to a release.

Age and severity are the attributes of RFCs adopted in our statistical analysis. The severity of a RFC denotes the impact of the corresponding error, which can be in one of the following categories:

• Severity 1: Critical Impact—A software component which is critical for business does not operate; or an absolutely necessary interface has failed; or an operator is unable to use XYZ resulting in a critical impact on operations. This condition requires an immediate solution.

• Severity 2: Significant Impact—A software component is severely restricted in its use, causing significant business impact. This indicates that XYZ is usable but is strongly limited.

• Severity 3: Moderate Impact—A non-critical software component is malfunctioning, causing moderate business impact. This indicates the program is usable with less significant features.

• Severity 4: Minimal Impact—A non-critical software component is malfunctioning, causing minimal impact, or a non-technical request is made.

Age provides the concise and precise account of the efforts expended to implement a change. 'Age' is usually called 'time to repair (TTR)' in current literature and is surveyed in a variety of technical fields. The historical data from four different releases of a XYZ are used to illustrate how each release has evolved over time. Furthermore use time series analysis techniques for identifying patterns in these data. Time series models assume that events are correlated over time and the impact of other factors is progressively captured in historical archives.

4 Observations and Results

In the following subsections, we highlight our observations and results from studying the XYZ data across four releases.

4.1 Characteristics of Releases

The managers of XYZ allowed submission of suggestions for a specific period of time between 10/12/2007 and 01/06/2009; during this time users recorded 143 SUGs. The majority of proposals (around 94 %) was submitted during the year 2008 and contributed to the enhanced version B.2.2 as noted earlier. Most of SUG (=127) have been closed during the time window of submitting recommendations, in this way the suggestions contributed to improve and to add new functions to releases B.2.1 and B.2.2.

Table 1 presents the start dates of the maintenance process for each release, which is taken to be the date of the first defect being raised. The final date is taken to be 30th September 2011, when data collection for this paper was closed. The parameter A in Table 1 indicates the temporal range starting with the first opened RFC and the 30th of September; the parameter B is the distance between the first and the last opened RFC. Thus our study on various releases covers different periods of time: the examination of release B.1 exceeds 4 years while the study of B.2.2 covers about 2 years and half. We decided to close our survey on the 30th of September and in this day the last RFC of B.2.2 was opened due to occasional reason that's why A and B coincide. Obviously, in the interests of consistency, we considered the number of RFC submitted over the first 730 days (=2 years) after each version has been released (Table 2); moreover we report the number of defects which required more than 1 year for resolution, the number of severity-1 defects and the percentage of these defects that have been closed after 30 days. All the releases have some RFCs with zero age (that is, the number of days spent to fix a RFC). This may indicate one of the following situations:

• A false problem was reported;

• The problem was trivial and immediately closed;

• The problem had already been addressed at the time the RFC was raised.

We notice that release B.1 has the highest number of submitted defects in the first couple of years, the highest number of defects with age exceeding 1 year and the highest number of severity-1 defects (Table 2).

Table 3 illustrate the increase in the size of XYZ—executable version—after each upgrade. Generally speaking, the size of a release in megabytes can be taken to mirror the complexity of the release's functionality. Releases B.2 and B.2.2 are much larger than their predecessors.

We note that B.2, B.2.1 and B.2.2 have the lowest number of defects (Table 2). These measures indicate the higher quality of the last releases respect to B.1, and match with the brief history of XYZ outlined in Sect. 2. Actually B.2, B.2.1 and B.2.2 were driven by more organized and focused development efforts, instead B.1 was adopted in a cursory manner. In the present context, one reasonably concludes that this can be a reflection of unsatisfactory development resulting in increased maintenance efforts.

4.2 Structure and Roles in the Maintenance Team

Defects are managed by a complex structure that basically includes four teams as follows:

• First Level Team—This group analyzes the issues and addresses the problems related to user errors or basic configurations when possible, otherwise involves the Second Level Team. The responses of this level are fast but do not go deep into problems.

• Second Level Team—It works with customers face to face to resolve RFCs. Level 2 provides the customer with a solution. If, and only if, customer is satisfied with the solution, Level 2 can close the procedure of fixing.

• Third Level Team—This level is responsible for resolving severe errors, creating fixes and making them available to users. This team assists customers in the diagnosis of reported problems that may be product defects and for making changes to released products in response to a RFC. This process governs the support of the product releases from assistance request by Level 2; it recognizes a valid problem through code changes, testing and then delivering of a fix for the detected error.

• Development Team—This level is responsible for new features that will be included in next product releases. In some cases provides help to Level 3 for hot customers issues or to evaluate possible enhancements request. The overall organization of teams is summed up in Fig. 1. It is possible to identify two areas in the chain of operations. The 'front-end' includes the support teams (Level 1 and Level 2) that have direct contacts with the client; the 'back-end', with the teams working on the problem resolution does not have a direct interaction with users.

The author of a RFC is required to describe the malfunctioning he or she experienced and to summarize the symptoms according to the list in Table 4. However over 80 % of records mention the common symptom 'program defect'. Users appear to provide the most generic description of the problem. The frequency of symptom #19 becomes lower when defects are serious. Users make a certain effort to scrutinize severity-1 failures. However the non-trivial percentage of symptom #19 (77.3 %) points out that this effort is not so great. So the lack of precision in describing the problem by the users has less to do with the effort required, and more influenced by users' attitude towards reporting less than critical errors.

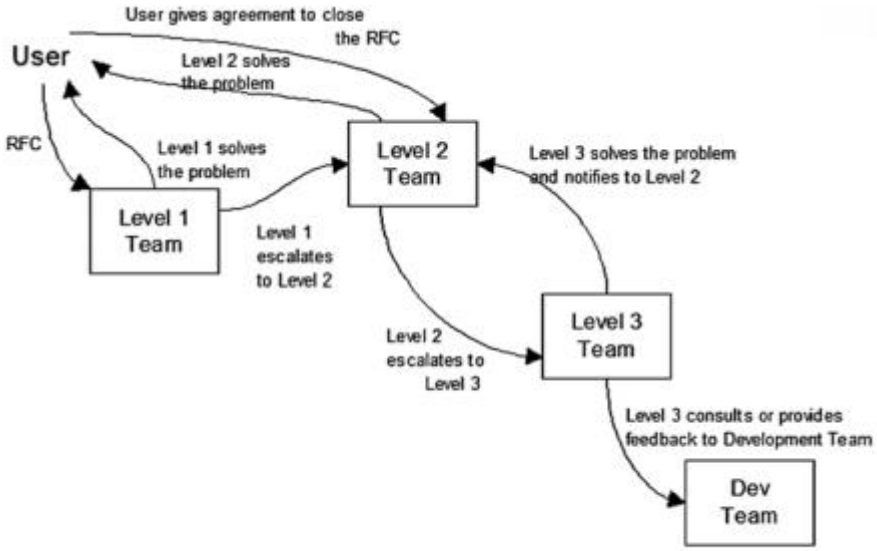

Fig. 1 Flow chart of the maintenance teams

4.3 Statistical Analysis

We have selected four distributions of age by severity and calculated nine statistical parameters of each distribution (Table 5). The kurtosis says that the severity-2 ages and the severity-4 ages have distributions with a lower, wider peak around the mean; on the other hand severity-1 and severity-3 ages show rather leptokurtic shapes. The age mean—named mean time to repair (MTTR) in current literature—diminishes through the groups 2, 3 and 4. Also the 50th percentile, which is the median, decreases from severity 2 to 3. Note that the mean and the median have been computed over the entire populations and not over a sample; their trends indicate that the effort to handle a change reduces as the severity of the defect lessens. The age mirrors the progressively reduced complexity of defects from 2 to 4, but the severity-1 problems have the lowest age mean, the lowest median and even the lowest standard deviation. This surprising result can be explained in the following manner.

An expert usually handles a RFC with severity 2, 3 or 4 but service level agreements warrant that a severity-1 problem must be resolved within 1 month (30 days). Thus the management needs to allocate more skilled personnel to close the most severe errors within this deadline. As we learnt, two, three or more experts work around this kind of errors and the age-mean is the lowest in the leftmost column of Table 4. However 80 % of age in group-1 largely exceeds 30 days (Table 2), which means the teams which handle severity-1 problems usually miss their deadlines. 4.4 Mean Time Between Events In general, it may be said that defect-fixing should make a sequence of independent processes; instead repairs—correlated in a way—reveal systematic flaws in the change management. We verified whether the age distributions of RFCs fit with the statistical Gamma model, typical of the Poisson processes. Gamma is a multiple-parameters family of continuous probability distributions.

As change managers established special procedures to handle each RFC according to its severity, we segregated the age into four homogeneous sets. We used the Kolmogorov-Smirnov test to evaluate the fitness of data with the Gamma distribution and this test was done at a 95 % level of confidence. Table 6 displays the parameters explaining the goodness-of-fit tests, in particular the table includes the goodness-of-fit statistic values (D), and the probability values (P). We note how the ages of severity 2 and 3 perfectly fit with Gamma (see Fig. 6 in Appendices) instead the higher distance D in the first and fourth row show how these processes comply with the Poisson model at lower degree of conformity. At far right Table 6 exhibits the most suitable values α, β and k for each group of data. The Gamma (k, α, β) distribution models the time required for an event to occur, given that the events occur randomly in a Poisson process with a mean time between events of β.

4.5 Defects Distribution

It is generally observed that users normally detect several defects soon after the product is released, and with time the number of opened RFCs comes down. We posited that studying the distribution of defects over time could reveal some pattern and regularity. The discussion in this section outlines our quest for a statistical law of defect-emergence.

We examined the temporal series of defects discovered for B.1, B.2, B.2.1 and B.2.2 releases to find the best description of these series. We performed the Kolmogorov–Smirnov test and observed that the four series of data fit with the Wakeby (WAK) distribution when the Kolmogorov–Smirnov test is accepted at the 99 % significance level. Table 7 shows the fitness parameters of the tests: D (= statistic), P (= probability-value) and R (= rank). At right side Table 7 exhibits the most suitable values of the Wakeby distributions. As R equals to 1 the Wakeby model represents the best fit respect to the other 39 distributions although the temporal series exhibit very different profiles. Figures from 2, 3, 4 and 5 plot the probability density functions regarding releases from B.1 to B.2.2. Each PDF plots the dates when the defects occurred during the range B (see Table 1). The dates have been grouped in order to execute the Kolmogorov–Smirnov test and at the far right of Table 7 one can find the size of the bars plotted in Figs. 2, 3, 4 and 5. This size is expressed in days.

5 Discussion

The analysis of the defects' time series conducted on the four versions of the middleware product offers insights that have implications for both research and practice. The first result is a confirmation of the contingent relationship between software development methods and software maintenance efforts. In fact version B.1 was implemented by a different development team with different methods and this has determined an higher number and higher severity of defects. This evidence confirms the perception that level of engagement in the software development process determines the error-proneness of the software produced. Further investi gation in this direction can lead to a deeper understanding of the contingent factors and their effects. As a practical implication of this finding, software companies can better identify the effective configurations of software development methods with respect to the architectural complexity and degree of openness of the digital artifact to be developed.

As a second result, we observed that eighty percent of severity-1 RFCs requires over 30 days for fixing the errors. This result supports the view that severe errors cannot be resolved in less than a minimum time. This is a reflection on the fact that communication overheads often negatively impact the time required for completing software tasks. The time necessary to repair severe defects cannot be compressed and thus preventive strategies often work better. Proactive maintenance is frequently less expensive as it directs actions to rectify a failure's potential root cause, rather than waiting for the manifestation of errors and then addressing them. Such proactive approach implements the continuous maintenance philosophy advocated when digital platforms are seen as embedded into emerging organizational contexts [19, 29].

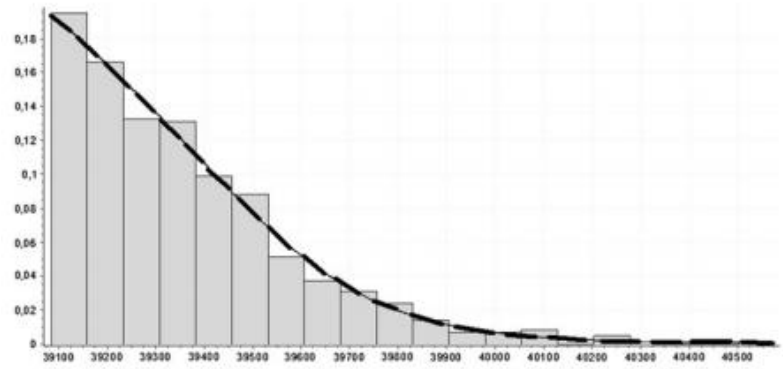

Fig. 2 PDF of 838 RFCs opened in the arch of 1471 days (Release B.1)

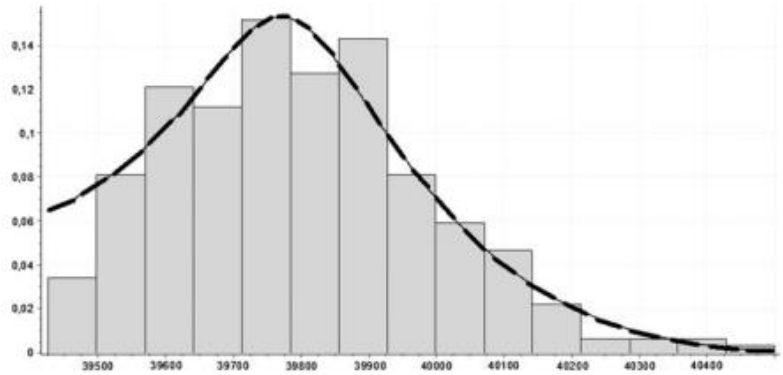

Fig. 3 PDF of 322 RFCs opened in the arch of 1074 days (Release B.2)

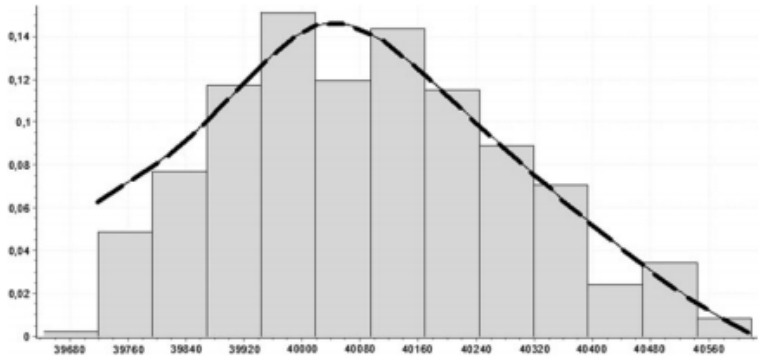

Fig. 4 PDF of 495 RFCs opened in the arch of 975 days (Release B.2.1)

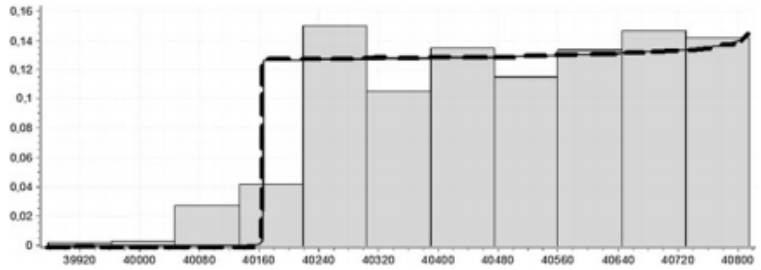

Fig. 5 PDF of 593 RFCs opened in the arch of 939 days (Release B.2.2)

As a third result, this study revealed that software defect time series best fit the Wakeby distribution. We found this distribution to match partial as well as entire time series data from all the releases with high confidence levels. Such regularity deserves a particular attention since it opens new perspectives of further empirical and theoretical studies. At a practical level, the Wakeby distribution can help in supporting proactive maintenance activities by forecasting software defects. From a theoretical perspective, it may be useful to outline some of the basic properties of the Wakeby distribution.

The Wakeby model is one of the more recent statistical distributions. It was defined by Harold Thomas and introduced by Houghton in 1978 [30]. The WAK function is largely adopted to study hydrology and in particular in the area of flood frequency analysis. Thomas defined the Wakeby distribution to account for the 'separation effect'. In order to account for this effect a distribution is needed with thick right-hand tail and left-hand tail. This makes the middle part of the distribution function steeper than traditional skewed curves. In addition WAK separates the calculation of the tails through β and δ that are shape parameter of the left end-tail and of the right-end tail respectively. We remind that ξ and α are location parameters; γ is a non-localized shape parameter.

It may be highlighted that WAK has five parameters, more than most of the common systems of distributions. This allows for a wider variety of shapes and the distribution is well suited to simulation of intricate physical phenomena. Furthermore, the Wakeby distribution exhibits more stability under small perturbations when compared to the Beta distribution and other more common distributions. Thus Wakeby distribution is highly general; it can describe complex events; it is robust against outliers, and it has a closed functional form for deter mining quantiles. To the best of our knowledge this is the first application of the Wakeby distribution in empirical software engineering. A deeper investigation on the meaning of these parameters in the two fields can provide further insights on the dynamics of digital platform evolution. This can lead to identify possible parallels between the complex socio-technical phenomenon of digital platform evolution and the behaviour of some physical, biological or social complex system.

6 Conclusion

This research contributes to the design of new managerial practices for coping with the evolution of digital platforms. These practices, grounded in the continuous maintenance paradigm, can be informed by new explanatory and predictive theories derived from the analysis of empirical data. Further empirical studies on these lines are necessary for strengthening the external validity of our results. For instance the same statistical analysis can be repeated on defects data taken from public sources (i.e. open source projects) or other proprietary software packages.

Appendices

See Tables 1, 2, 3, 4, 5, 6, and 7, Fig. 6.

Table 1 Time ranges of survey

Release	Starting date	*A* (days)	*B* (days)
B.1	02/01/2007	1732	1471
B.2	10/12/2007	1390	1074
B.2.1	15/07/2008	1172	975
B.2.2	05/03/2009	939	939

Table 2 Opened and closed RFCs

Release	RFCs opened in the 1st couple of years	Total opened RFCs	Total closed RFCs	Closed RFCs with Age ≥ 365	Closed RFCs with Age ≥ 30 (severity 1)	Closed RFCs with Age = 0
B.1	798	838	836 (99 %)	42 (5 %)	60/68	8
B.2	309	322	321 (99 %)	7 (2 %)	9/13	5
B.2.1	455	495	489 (98 %)	8 (1 %)	14/19	8
B.2.2	399	593	475 (80 %)	2 (0.4 %)	10/15	11
		2248			**93/115 (80 %)**	

Table 3 Sizes of releases

Release	Release year	Size	Absolute increase	Percentage increase
B.1	2006	637.168 Mb	N.A.	N.A.
B.2	2007	2.377 Gb	1.739 Gb	+272 %
B.2.1	2008	3.214 Gb	837 Gb	+48 %
B.2.2	2009	4.502 Gb	1.288 Gb	+153 %

Table 4 Symptoms of defects

	Symptoms	Severity →	1	2	3	4	Total
1	Pgm suspended	The program XYZ suddenly hangs or freezes	1	–	–	–	**1**
2	Lost data	Data are lost when XYZ is running	–	2	–	–	**2**
3	Reliability	XYZ is not robust enough and breaks easily	–	3	–	–	**3**
4	Test failed	The system test—conducted to validate a patch—failed	–	1	2	–	**3**
5	Not to spec	No precise symptom of the failure can be specified	–	2	1	1	**4**
6	Performance	The performances of one or more functions of XYZ are inadequate	–	1	3	1	**5**
7	Non-standard	The problem rises randomly	1	4	2	–	**7**
8	Obsolete code	A module of XYZ applies an algorithm which is obsolete	2	5	–	–	**7**
9	Core dump	The program XYZ encounters a sudden failure or outages without any error message	1	5	1	–	**7**
10	Design wrong	The failure of XYZ is due to a design error	–	1	6	–	**7**
11	Plans incorrect	There is a discrepancy between the functions planned for XYZ and the functions required by the user	–	9	1	–	**10**
12	Intg. problem	There are conflicts between XYZ and a program running in the system	–	7	4	–	**11**
13	Install failed	The problem rises during the XYZ installation phase.	–	3	11	2	**16**
14	Usability	Operators find difficult to use XYZ	3	9	13	3	**28** (1 %)
15	Docs incorrect	The documentation of XYZ is erroneous	4	23	7	4	**38** (1 %)
16	Function needed	It seems necessary to perfect or to add a new operation to XYZ	5	36	17	1	**59** (2 %)
17	Build failed	An error occurs during the compilation of XYZ and/or the linker phase	–	49	14	2	**65** (3 %)
18	Incorrect I/O	Malfunctions occur during an i/o operation e.g. XYZ displays a panel	9	78	46	1	**134** (6 %)
19	Program defect	A program error occurs	89	912	762	78	**1841** (81 %)

Table 5 Closed defect per severity

Age↓	Severity→					
	1	2	3	4		
Minimum	0	0	0	0		
Maximum	427	561	750	346		
Std deviation	84.85	98.48	111.55	88.48		
Mean	89.83	110.39	106.66	104.45		
Kurtosis	3.71	1.74	4.91	−0.05		
Skewness	1.89	13.945	1.90	0.89		
25th percentile	36	40	24	33		
50th percentile	65	79	75	83		
75th percentile	120	156	144	166		
Number of defects	115	1150	890	93	2248	**Grand Total**

Table 6 The Gamma distribution parameters

Severity	D	P	k	α	β
1	0.064	0.693	0.74806	2.2175	28.791
2	0.030	0.244	1.00000	1.2564	87.863
3	0.042	0.073	1.00000	0.9142	116.67
4	0.071	0.705	1.00000	1.1204	95.272

Table 7 The Wakeby distribution parameters of the entire data sets

Release	R	D	P	α	β	γ	δ	ξ	Segment (days)
B.1	1	0.01986	0.88908	261.35	0.82955	121.3	0.10885	39082.0	73.5
B.2	1	0.02816	0.95414	803.23	4.4198	228.77	−0.21498	39460.0	71.6
B.2.1	1	0.0205	0.98272	742.2	5.5649	333.95	−0.44585	39749.0	75.0
B.2.2	1	0.04052	0.27695	6.7581E+8	16826.0	676.17	−1.0297	0	85.36

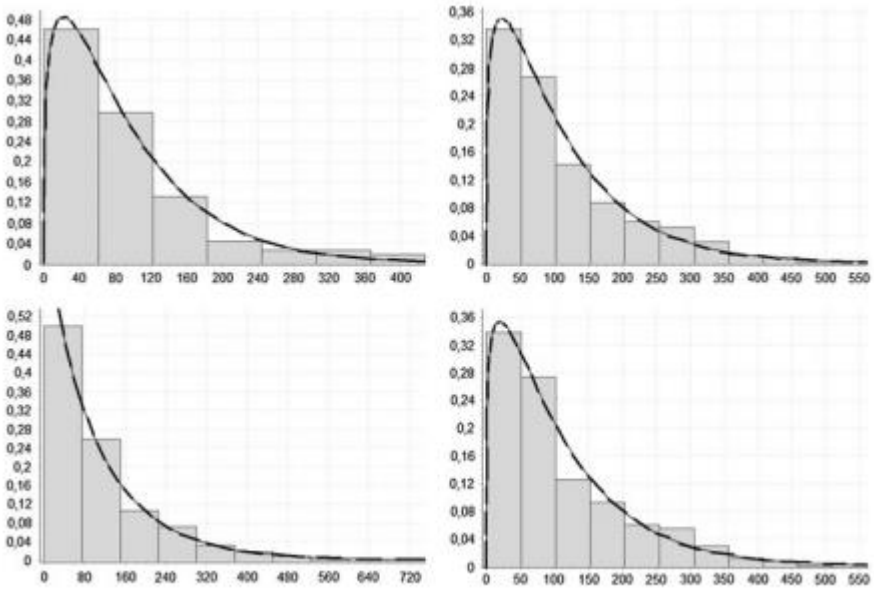

Fig. 6 PDF of the ages (Severity 1, 2, 3 and 4)

References

1. Resca, A., Za, S., Spagnoletti, P.: Digital platforms as sources for organizational and strategic transformation: a case study of the Midblue project. J. Theor. Appl. e-Commerce Res. 8, 71– 84 (2013)

2. Spagnoletti, P., Resca, A., Lee, G.: A design theory for digital platforms supporting online communities: a multiple case study. J. Inf. Technol. 1–17 (2015)

3. Hanseth, O., Lyytinen, K.: Design theory for dynamic complexity in information infrastructures: the case of building internet. J. Inf. Technol. 25, 1–19 (2010)

4. Marsden, C.T.: Net Neutrality: Towards a Co-regulatory Solution. Bloomsbury Academic, London (2013)

5. Zittrain, J.: The generative internet. Harv. Law Rev. 119, 1975–2040 (2006)

6. Rossignoli, C., Zardini, A., Benetollo, P.: The process of digitalisation in radiology as a lever for organisational change: the case of the Academic Integrated Hospital of Verona. DSS 2.0-Supporting Decision Making With New Technologies, p. 261 (2014)

7. Vom Brocke, J., Braccini, A.M., Sonnenberg, C., Spagnoletti, P.: Living IT infrastructures— an ontology-based approach to aligning IT infrastructure capacity and business needs. Int. J. Account. Inf. Syst. 15, 246–274 (2014)

8. Boudreau, K.J.: Let a thousand flowers bloom? An early look at large numbers of software app developers and patterns of innovation. Organ. Sci. 23, 1409–1427 (2011)

9. Vom Brocke, J., Simons, A., Sonnemberg, C., Agostini, P.L., Zardini, A.: Value assessment of enterprise content management systems: a process-oriented approach. In: D'Atri, A., Saccà, D. (eds.) Information Systems: People, Organizations, Institutions, and Technologies, pp. 131– 138. Physica-Verlag, Heidelberg (2010)

10. Magni, M., Provera, B., Proserpio, L.: Individual attitude toward improvisation in information systems development. Behav. Inf. Technol. 29, 245–255 (2010)

11. Lehman, M.M., Ramil, J.F.: Rules and tools for software evolution planning and management. Ann. Softw. Eng. 11, 15–44 (2001)

12. Gawer, A.: Platforms, Markets and Innovation. Edward Elgar Publishing, Cheltenham (2009)

13. Sorrentino, M., Virili, F.: Web services and value generation in the public sector. Electron. Gov. 489–495 (2004)

14. Spagnoletti, P., Resca, A.: A design theory for IT supporting online communities. In: Proceedings of the 45th Hawaii International Conference on System Sciences, pp. 4082–4091 (2012)

15. Williams, R., Pollock, N.: Software and Organisations—The Biography of the Enterprise-Wide System or How SAP Conquered the World. Routledge, London (2008)

16. Vitari, C., Piccoli, G., Mola, L., Rossignoli, C.: Antecedents of IT dynamic capabilities in the context of the digital data genesis. In: ECIS 2012: The 20th European Conference on Information Systems (2012)

17. Spagnoletti, P., Federici, T.: Exploring the interplay between FLOSS adoption and organizational innovation. Commun. Assoc. Inf. Syst. 29, 279–298 (2011)

18. Yoo, Y., Boland, R.J., Lyytinen, K., Majchrzak, A.: Organizing for innovation in the digitized world. Organ. Sci. 23, 1398–1408 (2012)

19. Truex, D., Baskerville, R., Klein, H.: Growing systems in emergent organizations. Commun. ACM 42, 117–123 (1999)

20. Ramesh, B., Cao, L., Baskerville, R.: Agile requirements engineering practices and challenges: an empirical study. Inf. Syst. J. 20, 449–480 (2007)

21. Lee, G., Xia, W.: Toward agile: an integrated analysis of quantitative and qualitative field data on software development agility. MIS Q. 34, 87–114 (2010)

22. Pino, F.J., Ruiz, F., Garcia, F., Piattini, M.: A software maintenance methodology for small organizations : Agile MANTEMA. J. Softw. Maint. Evol. Res. Pract. 24, 851–876 (2012)

23. Subramanyam, R., Ramasubbu, N., Krishnan, M.: In search of efficient flexibility: effects of software component granularity on development effort, defects, and customization effort. Inf. Syst. Res. 23, 787–803 (2012)

24. Hirt, S.G., Swanson, E.B.: Emergent maintenance of ERP: new roles and relationships. J. Softw. Maint. Evol. Res. Pract. 13, 373–387 (2001)

25. Caporarello, L., Viachka, A.: Individual readiness for change in the context of enterprise resource planning system implementation. In: Proceedings of the 6th Conference of the Italian Chapter for the Association for Information Systems, pp. 89–96 (2010)

26. Ng, C., Gable, G.: Maintaining ERP packaged software: a revelatory case study. J. Inf. Technol. 25, 65–90 (2009)

27. Moon, J.Y., Sproull, L.S.: The role of feedback in managing the internet-based volunteer work force. Inf. Syst. Res. 19, 494–515 (2008)

28. Yin, R.K.: Case Study Research: Design and Methods. Sage Publications, Thousand Oaks (2009)

29. Pennarola, F., Caporarello, L.: Enhanced class replay: will this turn into better learning? In: Wankel, C., Blessinger, P. (eds.) Increasing Student Engagement and Retention Using Classroom Technologies: Classroom Response Systems and Mediated Discourse Technologies, pp. 143–162. Emerald Group Publishing Limited, Bradford (2013)

30. Houghton, J.C.: Birth of a parent: the Wakeby distribution for modeling flood flows. Water Resour. Res. 14, 1105–1109 (1978)