Send Orders for Reprints to reprints@benthamscience.ae

Open Computer Science Journal, **2015,** *2,* **1-10**

Open Access

No Silver Bullet – The Evolutionary Model

Daniel Galin*

28 Yehuda Halevi st., Raanana, 43556, Israel

Abstract: This paper's objective is to present an evolutionary model that better explains the behavior of the gap between software engineering (SE) essence difficulties and SE capabilities, and adds an evolutionary dimension to Brooks's approach regarding the gap. In his 1987 landmark paper, Brooks argues that software projects incorporate inherent essence-conceptual difficulties that are unchanging and irreducible. He concludes that SE developments hardly contribute to SE capabilities to handle SE essence difficulties. Thus, there is no solution to SE essence difficulties. In Brooks' words, "*there is no silver bullet* (SB)", as the gap between SE essence difficulties and SE capabilities, (hereafter named "the SB gap"), is unchangeable and irreducible. We argue that the SB gap is the result of two evolutionary processes, that affect SE simultaneously: (a) continuously growing SE essence difficulties and (b) continuously growing SE capabilities. Thus, the size of the gap is not irreducible and unchangeable as in Brooks's argument. Periods of rapid SE development, rapid growth of SE capabilities may cause a reduced size of the gap. However, in periods where economic developments cause a higher growth rate of SE essence difficulties, a larger gap may result. It is noteworthy that the smaller the gap, the better the software development performance is in terms of reliability, productivity and simplicity.

Keywords: Software engineering, software engineering capabilities, design tools and techniques, requirement specifications, change of requirements, project management.

1. INTRODUCTION

In his 1987 paper, Fredrick Brooks [1] describes the familiar software project with its typical surprises of *missed schedules flawed products* as being analogous to the mystical behavior of werewolves, whom hunters have no success in killing:

Of all the monsters that fill the nightmares of our folklore, none terrifies more than werewolves, because they transform unexpectedly from familiar into horror. One seeks a bullet of silver that can magically lay them to rest.

Now, more than twenty years later, we can add to Brooks's legendary analogy as follows:

Later in our dreams, the hunters who want to lay the werewolves to rest continue to improve their rifles, making them more accurate and their ammunition more lethal. They also develop better optics for their field glasses. At the same time, the werewolves have changed, too. The new generations seem to be smaller in size, quicker to escape, and even better at detecting the hunters. In addition, they have developed new ways to surprise us, with horror shows combined with running maneuvers of changing directions. All these changes help them to continue escaping the hunters' bullets. During this period, equipped with more advanced weapons, the hunters try over and over again to hit the werewolves, but with no success. So, unfortunately, up until now, there have been no lethal hits. Now, as before, one seeks a bullet of silver that can magically lay them to rest.

This paper's objective is to present an evolutionary model that describes and explains the behavior of the gap between software engineering (SE) essence difficulties and SE capabilities. This gap is "responsible" for the unsolved essence difficulties of SE projects, This model presented here, supported by theoretical and empirical literature, tries to add an evolutionary dimension to Brooks' approach regarding this gap.

In his landmark 1987 paper, Brooks [1] presents his conclusion that the nature of software development projects incorporates difficulties of two types: (1) inherent essence-conceptual abstraction difficulties ("SE essence difficulties") and (2) difficulties characterized by syntax errors and other technical non-essential errors ("SE accidental difficulties"). Brooks claims that SE essence difficulties, which mainly affect the specifications, design and testing stages of the software development process, are unchanging and irreducible. The other type of SE difficulties, the SE accidental difficulties are typical to the programming stage, and are comparatively easy to solve. Brooks argues that SE essence difficulties are caused by "*inherent properties of this irreducible essence of modern software systems: complexity, conformity, changeability and invisibility*". In order to examine his construct, Brooks analyzed a wide range of SE technologies and methodologies developments. He found that, while these improvements, contribute almost entirely to better handling of SE accidental difficulties, they hardly

*Address correspondence to this author at the 28 Yehuda Halevi st., Raanana, 43556, Israel; Tel: 972-9-7713750; Fax: 972-9-7745802;
E-mail: dgalin@bezeqint.net

2352-6270/15

contribute to the solving SE essence difficulties, Accordingly SE essence difficulties are unchanged, and so there is no solution for SE essence difficulties. In Brooks's words, *there* is *no silver bullet*, as the gap between SE essence difficulties and SE capabilities, hereafter named "the SB gap", is unchanged and irreducible. Twenty years later [2], in a 2007 retrospective examination of SE developments since the publication of his 1987 paper, Brooks concludes that his approach regarding SE essence difficulties is still valid. Furthermore he finds that "*most of the proposed radical improvements continue to address only accidental difficulties*". In other words, SE essence difficulties continue to be irreducible and unchanged.

Brooks's analytical axiomatic article has incited many reactions by both supporters and opponents. Berry [3-5], Jeffrey [6] and Spinellis [7] support Brooks's conclusions. Berry [3, 4] refers to the phenomenon of SE essence difficulties as *the inevitable pain of software development*. He argues there are still no solutions for SE essence difficulties. He also claims that the never-ending requirement changes during the software development process and later, add tremendously to SE essence difficulties. In another paper, Berry [5] argues that SE developments create SB situations that *quickly solve all the formerly too-tough problems that the silver bullet lets us solve. Doing this brings to a new frontier of not-easily-solved problems.* In other words, growing SE capabilities drives the adoption of growing SE essence difficulties. Jeffrey [6] presents the human factor, including cultural differences between developers, clients and users, as an additional source of SE essence difficulties. Spinellis [7] and Bell [8] found that new methodologies and technologies, although often considered SBs by the developers, were not successful and SE still suffer from SE essence difficulties.

Several authors, among them Blaha [9], Glott *et al.* [10] and Sharma *et al.* [11] agree with Brooks's conclusions but offer a variety of SE methodologies and technologies, which are expected to mitigate the effects of SE essence difficulties. Blaha presents the application of reverse engineering as a tool for analyzing the database quality of proposed software products, as an additional source for better handling of SE essence difficulties. This may serve, according to Blaha as "copper bullets", which could improve SE capabilities significantly, yet not become the much needed SB. Glott *et al.* like Blaha, address the reliability aspects of SE essence difficulties, and claim that the application of models of second-generation quality metrics will significantly improve the quality of selected software. Based on their professional experience in India, Sharma et al [11], suggest experienced management as a tool with which to achieve meaningful improvement of SE capabilities.

The 20th anniversary of Fred Brooks' 1987 paper occasioned a discussion panel, whose subject was whether the findings presented in the 1987 paper are still valid after 20 years [2, 12, 13]. Fred Brooks, who himself participated in the panel, expressed his belief that no change had occurred during the past two decades. Moreover, he claimed that SE advancements had contributed to reducing SE accidental errors in the development process, but had no significant effect on SE essence difficulties. All of the other anniversary panel participants agreed that the SB had not been created or discovered over the two-decade period, since Brooks first published the paper. Almost all of the participants mentioned object-oriented methodology as the major tool that could bring us closer to finding the SB. Some of the participants stressed the importance of craftsmanship and education as the method by which to achieve simplicity in software structure and its implementation, thus getting closer to the SB.

Among the better known authors who disagree with Brooks are Harel [14], Cox [15] and Dromey [16]. They believe their suggestions for new SE technologies or methodologies, when fully implemented, will substantially reduce - or even close - the current SB gap. Harel analyzes the history of software development since the 1950's and concludes that the continuous growth of SE capabilities, contributed to solve SE accidental difficulties, but also SE essence difficulties He sees *a brighter future for system development* by implementation of a framework composed of modular development, visual representation and the use of computerized tools. Cox claims that a SB could be achieved by applying the principles of object-oriented development processes and creating an advanced trade of software modules. Dromey expects a major reduction in SE essence difficulties as a result of reducing, or possibly eliminating, the complexity of software development processes. This, he argues, can be accomplished by splitting project requirements, dealing with them "one by one", and finally integrating the resulting designs.

To sum up, both supporters and opponents of Brooks' ideas agree that: (a) The inherent software development essence difficulties are the cause of reliability, productivity and simplicity problems related to SE projects. (b) During the two decades since the publication of Brooks's paper, the numerous SE developments during this period have significantly contributed to enhancing SE capabilities, mainly in regard to solving SE accidental / random difficulties, but not enough to solve SE essence difficulties.

Based on the above review, if we follow Brooks' approach, one may question, why haven't the cumulative result of the many developments of SE methodologies and technologies, during the last two decades, finally solved the problem of SE essence difficulties, and provide us with the much needed SB

The answer to the above question lies in the fact that both, the SE essence difficulties and SE capabilities are not irreducible and unchanging but changing over time. Their simultaneously evolutionary changes explain the fact of substantial possible changes in the size of the SB gap over time periods

In the next section, I will propose and describe an evolutionary SB gap model. This model expands Brooks's approach regarding the SB gap by adding a time dimension to the SB gap discussion. The two sections that follow discuss the various factors and processes that generate and affect the evolution of SE essence difficulties and the evolutionary growth of SE capabilities to handle SE essence difficulties.

2. THE EVOLUTIONARY GAP MODEL

According to the evolutionary SB gap model, the SB gap is the result of two independent simultaneous evolutionary processes:

(a) The continuously growing SE essence difficulties, becoming more rigorous over the years, and (b) The continuously enhancing SE capabilities, trying to handle the growing SE essence difficulties.

According to the model, the gap between these two evolutionary processes, at any point of time, the SB gap, is not of constant size. In periods of rapid SE development, rapid growth of SE capabilities is observed, causing a reduction in the size of the SB gap. However, in periods where economic developments cause a higher growth rate of SE essence difficulties, a larger SB gap may result. It should be mentioned that the smaller the SB gap, the better the software development performance is in terms of reliability, productivity and simplicity.

3. THE EVOLUTIONARY CHARACTERISTICS OF SE ESSENCE DIFFICULTIES

Four simultaneous processes create the evolutionary growth of SE essence difficulties: (1) The enhancement of projects' requirement specifications; (2) The effect of shorter time schedules of software development projects; (3) The increased frequency of customers demands for changes of the requirement specifications, while the project is being carried out and after its completion. (4) The higher dependence on IT systems for operation, control and management of organization, results in a lower tolerance to SE errors. These four processes are discussed in further detail below.

Each of these processes cause growing essence difficulties which result in reduced achievements in one or more of the areas of productivity, reliability and simplicity.

3.1. The Enhancement of Projects' Requirement Specifications

The phenomenon of enhancement of projects' requirement specifications has four main characteristics, each of which increases SE essence difficulties. Newer software products are characterized by:

- Increased algorithm complexity, a greater number of parameters and variables, more complex relationships between variables, and more complicated calculations.
- Growing interface requirements – namely, software-software, software-firmware (software embedded in electronic equipment) and firmware-firmware. We refer here to standard requirements from new software packages to provide a wide range of interface capabilities according to international standards, required interface with leading software packages, and required interface with firmware embedded in the equipment by principal manufacturers.
- More requirements for intra-organizational integration of individual systems into one integrated system. Representative examples of such integrations are the ERP (Enterprise Resources Planning) and CRM (Customer Relationship Management) software systems, which combine the functions of several intra-organizational software systems.
- Increased requirements for inter-organizational integration abilities of software systems. Typical higher SE essence difficulties of this type evolve in the development of software systems for supply chain management (SCM) services. Another area of inter-organizational integration is initiated by governmental agencies for the operation of systems in the area of taxation, which serve both the public and government agencies.

Linda Northrop [2] describes the enhancement of requirements: "current trends are leading us to systems of unthinkable scale in not only line of code, but in the amount of data stored, accessed, manipulated and refined, the number of connections and interdependencies, the number of hardware and computational elements, the number of system purposes and user perception of these purposes, the number of routine processes, the number of interactions and emergent behaviors, and the sheer number of people involved in some way". She explains the increase in projects' requirements as follows: "our global appetite for complex software and software-intensive systems continues to increase at a rate comparable to the increases in the computational capacity of hardware".

The evolution of software systems, in both size and complexity, can be seen by analyzing the growth of the number of software lines of code from a software system's release to its subsequent release. Sue and Neamtiu [17] examine the size growth of a sample of 7 software applications over several years. The results are shown in Table **1**.

Similar findings were reported by Robles *et al.* [18]. They analyzed reasonably large and representative samples of stable software systems, large in the number of lines of code, with an active community and user base, and also studied the software systems' evolution. Most of these software systems show a clear linear growth pattern.

An alternative way to study the evolution of software size and complexity is by examining the number of modules in subsequent software system releases. Turski [19, 20] discusses Lehman's laws of software evolution. In [19], Turski presents a typical example of software system growth over 21 releases.

Release No.	Size (number of modules)
1	977
6	1492
11	1832
16	2091
21	2315

In the first release, there were 977 modules, while in the 21st release there were 2,315 modules; this means an average growth of 70 modules in the number of software

Table 1. Application size evolution.

Program	Time Frame (Years)	First release		Last release size (LOC)	Total size growth	Annual size growth
		Year	Size (LOC)			
Samba	15	1993	5,514	1,045,928	X189.7	X1.42
Sendmail	15	1993	25,912	87,842	X3.4	X1.085
Bird	9	2000	169,306	321,689	X1.9	X1.074
OpenSSH	9	1999	12,819	52,284	X4.1	X1.17
SQLite	8	2000	17,273	65,108	X3.65	X1.18
Vsftpd	8	2001	6,774	15,711	X2.32	X1.11
Quagga	5	2003	41,623	47,511	X1.14	X1.03

modules, compared to its former release, and about a 137% growth rate over 20 releases. Similar results of steady growth were found for software systems based on open source software, specifically related to the Linux kernel and its many releases over 14 years ([21, 22]). The evolution of software complexity is also discussed by Arthur [23]. He proposes three causes for software systems' growing complexity: an increase in the diversity of the "species" served by the system, an increase of structural sophistication, and additional functionality to overcome performance limitations.

3.2. The Effect of Shorter time Schedules of Software Projects

The continuous trend of decrease of project time schedules has been typical of all types of software projects over recent decades. It is the general belief that project schedules of the same magnitude are cut by 50% every 2-4 years. The implications of this trend on SE capabilities to handle SE essence issues are severe:

- In general, shorter project time schedules means less time available for reviewing and testing. Thus, the ability to fully solve SE essence difficulties is reduced.
- The requirement to complete projects within ever shorter time constraints leads to a need to employ a larger development team. Larger development teams, in turn, eventually causes a need for more cooperation and coordination efforts, which is harder to achieve.
- Furthermore, shorter time schedules, in many cases, forces the developer to use partners or subcontractors or outsourcing services for parts of the project. These types of project organization require higher control efforts and cause coordination difficulties. Thus, this situation is involved with higher risks for software essence errors.

3.3. The Effects of More Requirement Changes

The increased frequency of customer demands for changes of requirements specification during the development process and after the project has been completed, typical to the last decade, yields lesser ability to cope with SE essence difficulties:

- The handling of more changes of requirements specifications during software development, adds to the project's SE essence difficulties. It increases the work burden and time pressures on the project teams, leading to inability to perform all the required reviewing and testing of the software system. As a result, a project that was involved with many changes is expected to be characterized by high rate of unsolved SE essence difficulties.
- Demands for changes after the project was completed usually create SE essence difficulties that are harder to solve, as they are usually performed by less experienced team.

The severe negative effect of a higher frequency of changes of the requirementד is discussed by Berry [3], Williams *et al.* [24] and Rahman *et al.* [25]. Berry argues that "a typical software development method is effective in its first application development problem. However, once developers have built and developed a version with its method, the requirements begin to change, whether from E-type system pressures or client and user demand. When an inevitable change comes along, modifying the method, documenting artifacts is so painful that developers avoid doing it the right way, by carefully tracking the change's effects. Instead, they create a quick patch that increases the system's brittleness". Williams *et al.* [24] and Rahman *et al.* [25] refer to the risks involved in late changes of requirements of schedule delay and additional costs, which are harder to maintain in the future. On top of these impacts, a severe risk discussed in these studies is that of the "snowball effect", which relates to additional changes in other parts of the project needed on top of the original initiated change. The original change task is completed only once the entire series of additional changes has been satisfactorily completed. They suggest a risk assessment study prior to a decision to perform a requested change of requirements.

3.4. Lower Tolerance to SE Errors

The IT advancements, especially the introduction of integrated software systems, inevitably create growing reliance and dependence of users on these systems, for the

Table 2. US average software development productivity of software projects from 1960 to 2010 (FP per staff month, months needed for a10-FP project).

Year	Function Point (FP) per Staff Month	Staff Months Needed to Complete a 10-FP Software Project
1960	1.25	8.0
1965	1.93	5.2
1970	2.63	2.6
1975	4.83	2.07
1980	5.96	1.67
1985	7.75	1.29
1990	11.75	0.85
1995	14.00	0.71
2000	17.43	0.57
2005	19.25	0.52
2010	20.93	0.48

operation, control and management of organizations. Thus, organizations became less and less tolerant to SE essence errors, even to minor ones. Organizations that once could tolerate an error that paralyzed the operation of a minor application for a short time, are no more able to tolerate it. Due to systems' integration, any error affects now the whole integrated software system. This trend raises the need for higher perfection of the software development process, which naturally adds to the difficulty of solving the SE essence difficulties.

Mens *et al.* [26] describes the dependence on software as general: this dependence "takes place in all sectors of society, including government, industry, transportation, commerce, manufacturing and the private sector". They claim that the low quality of software and specifically software maintenance are the challenges of SE, and list relevant areas of research and development for SE. Supported by other researchers' findings, Duggan [27] discusses the increased reliance of business on IT and lists several reasons: (a) "To support missions and priorities, either for strategic enhancement or competitive necessity. This accentuates the challenges that face IS developers"; (b) "Organizational drives to apply more sophisticated technologies and establish flexible communication networks to accommodate a variety of data and processing distribution strategies, multimedia operations and integrated systems"; and (c) "Increased security concern that now attends the greater movement of data".

4. THE EVOLUTIONARY PROCESS OF GROWING SE CAPABILITIES

During the last four decades, SE has been characterized by a continuous flow of developments of methodologies and techniques. This resulted in growing capabilities and was expressed by rapid growth in productivity. Studies on SE productivity growth trends by Jones [28] show about 150% growth in average productivity over the 20-year period of 1985-2005. While in 1985, 1.29 staff months were required to complete a project of an estimated 10 function points, only 0.52 staff months were needed to complete the same project in 2005. The US average software development productivity of software projects from 1960 to 2010 are presented in Table **2**.

The advancement of software development tools explains a great part of this productivity change. Based on results by Jones [29], we find that completing the above-mentionesd10-FP project employing Assembler language would require 10.5 staff months, while employing Cobol would require 4 staff months. However, only 1 staff month was needed when a spreadsheet was used. Even if we assumed that most productivity growth was gained by eliminating SE accidental difficulties, a substantial improvement in handling SE essence difficulties was still achieved. As mentioned above in the Introduction, this improvement in handling SE essence difficulties is noted by Berry [5], accompanied by new SE essence difficulties.

As shown above, SE has experienced a continuous flow of new SE technology and methodology developments over the last decades. Will this SE capability improvement continue? Besides the expected new SE methodologies and technologies, one can expect the additional contribution of already-existing SE tools. A great part of the known SE developments has still not contributed their full potential to improved SE capabilities, but are expected to do so in the future. We may assume that the current contribution of any of these developments to SE capabilities in handling SE essence difficulties is minimal. However, some of these SE developments will succeed in fulfilling their potential, and contribute much more to further extend SE capabilities to handle SE essence difficulties.

The four examples (out of many SE developments) presented below, demonstrate current SE developments that have not yet realized their full potential. These four SE developments are:

Table 3. Software Reuse - 1955-2015.

Year	% of Software Reuse Level
1955	0-5%
1965	0-5%
1975	0-10%
1985	0 -15%
1995	0-25%
2005	0-40%
2015 (predicted)	0.85%

1. COTS software and software reuse
2. CASE tools
3. Automated testing
4. Open source software

Common to the above-mentioned four software engineering areas is their long presence in the SE market, and their slow rate of implementation, much below the predicted expectations. Still the continuous growth in the use of these SE developments, together with extensive research efforts the near future, are expected to yield extended SE capabilities or even a breakthrough, at least in some, if not all, of these four areas.

4.1. COTS Software and Software Reuse

The use of purchased or reused software, in its variety of forms, is probably a very effective, if not the most effective, response to growing SE essence difficulties. These SE practices include purchase of software packages (COTS software), purchase of software components for their integration in required applications, as well as software reuse. Common to all these courses of action is their reliance on software that has already been tested and corrected, according to defects identified in previous tests and by earlier users. It should be noted that applying COTS software and software reuse do not completely eliminate SE essence difficulties for the development teams. These teams still have to cope with SE essence difficulties related to the analysis of the requirement specifications and their fit with regard to the proposed software package. Still, the major part of the analysis, design and testing of SE essence specific difficulties is saved. Applications of these practices can cause: (a) Substantial reductions of the required development resources; (b) A lower rate of SE essence errors; (c) A shorter project schedules and better schedule keeping; and (d) A smoother development process, due to increased standardization, resulting from repeated use of the same software package components.

The application rate of COTS software, purchased software components, and software reuse is still relatively low compared to their evaluated potential. The level of software reuse has been continuously growing over the last decades, as presented by Jones [30] for the period 1955-2005 and as predicted for 2015. See Table **3** for reuse percentages.

In their 1995 paper, Garlen *et al.* [31], while believing that the future of software development productivity depends on software reuse, discuss the difficulties in realizing the reuse potential. They claim that the main reason for difficulties in matching reused software in a new software project is the incompatibility in programming language, operating platform and database scheme. More than ten years later, in 2009, Garlen *et al.* [32] and Andersen [33], facing the still low implementation of software reuse, once again discuss the mismatch issues of software reuse as the reason why the potential of software reuse is still far from being realized. A variety of methodologies to extend software reuse, including the building of software reuse infrastructure, prioritizing of candidate reuse projects, and establishing software reuse libraries, are offered by Sutcliffe *et al.* [34] and Sherif *et al.* [35]. Another way to promote reusable software discussed by Augustwamy and Frakes [36] is through the designing and building of reusable components that will be easier to apply. In an empirical study, he examines design principles that could guide the developers of reusable software components.

CBSD (COTS Based Software Development) is considered to be the next revolution in software development, with the advantages of lower costs, shorter timetables and higher quality. It is also expected to contribute to easier maintenance and be based on replacement with new enhanced COTS components [37]. Much research has been dedicated to analyzing the current challenges of CBSD in the finding and selection of appropriate COTS product implementation. The difficulties of selection and systematic identification of the needed COTS products are considered the main barrier to wider implementation of CBSD. Methods for improving the selection process are discussed by many authors, to mention just few [37-43]. The documentation quality of COTS components, namely incomplete and inaccurate documentation, as well as communication with vendors, are mentioned as difficulties in the integration and testing stages of implementing CBSD projects [38], and [44]. Another reason preventing wider implementation is the expected high efforts needed to select and identify the appropriate COTS components that cause developers to prefer self-development. The application of COTS implementation efforts' estimation method can promote COTS use [45].

Table 4. Average defect rates in US software projects, 1955-2015.

Year	Average Defect Removal %	Delivered Defects per Function Point
1955	80%	1.40
1965	83%	1.02
1975	85%	0.75
1985	87%	0.58
1995	90%	0.40
2005	92%	0.28
2015 (prediction)	96%	0.10

4.2. CASE Tools

It is expected that over the next few years, applications of new or improved computer-aided SE (CASE) tools will improve general software development performance. Today's CASE tools enable teams to automate segments of the software development process with no defects; specifically, to automate the design generation out of requirement specifications, and the automated code generation out of design. Other CASE tools provide updated and accurate documentation and support coordination among members of large development teams. Thus, while the current CASE tools mainly contribute to solving accidental SE difficulties, they also contribute, to some extent, to reducing SE essence difficulties by reducing rates of design-generated errors, improved documentation and coordination. The success rates of CASE applications in the nineties were disappointing. Research results identified the main causes for low implementation as follows: they were not user-friendly, offered low management support, and a lack of voluntariness [46]. This low implementation rate is still discussed a decade later [47]. Much research effort has been dedicated to the development of methods to overcome the difficulties of wider implementation of CASE tools (some typical examples, [48-51].

4.3. Automated Testing

The improvement in testing techniques, especially by automated testing, as well as improved development tools and design reviews, have improved SE capabilities by reducing the defect rates and saving correction efforts. Based on Jones's [30] results, Table **4** presents the defect rates in the US for the years 1955-2005 and a prediction for 2015.

The higher percentages of defects removed and the lower numbers of defects delivered helped SE cope with users' growing dependence on the regular operation of software systems and their lower tolerance to SE errors.

Automated testing provides for a better error identification rate compared with manual testing, as related to SE accidental errors as well as SE essence errors. Currently, automated testing tools do not serve all types of software. Moreover, the planning and defining of the automated testing plan (the scenarios) still requires numerous manual resources. Thus, surveys of testing methods application find that, although the contribution of automated testing is already substantial, it's still far below its full potential [52-54]. Research efforts into the development of tools for automating software testing include the development of tools for specific programming languages, the analysis of organizational aspects, as well as economical aspects [55-59].

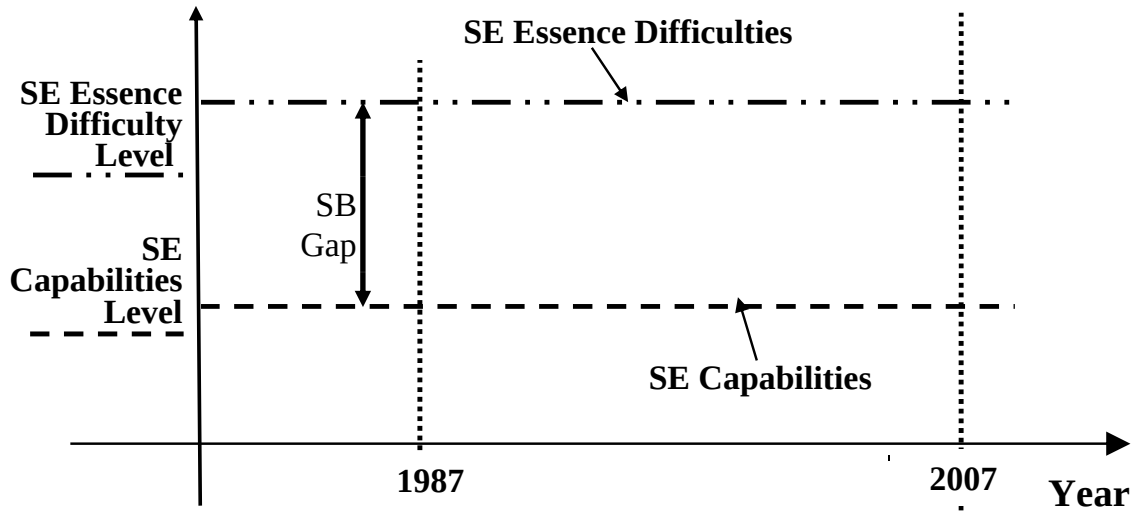
4.4. Open Source Software

Open source software (OSS) provides software developers with free access to a great variety of ready-to-use software program codes, while allowing one to make changes and adaptations to suit individual needs. Similarly to the application of COTS software, the application of OSS - which others have already developed, tested and corrected - presents a sizeable potential for reducing the required development resources. It should be noted that applying OSS still requires the developers to handle SE essence difficulties involved in analyzing the fit of the proposed OSS package to the customer's requirement specifications and in performing the necessary adaptations, Still, the major part of the analysis, design and testing of SE essence difficulties are saved. It is expected that software development, based on OSS, will become one of the major answers to the challenges of growing SE essence difficulties. A technical and economical evaluation should lead to the adoption of OSS [60].

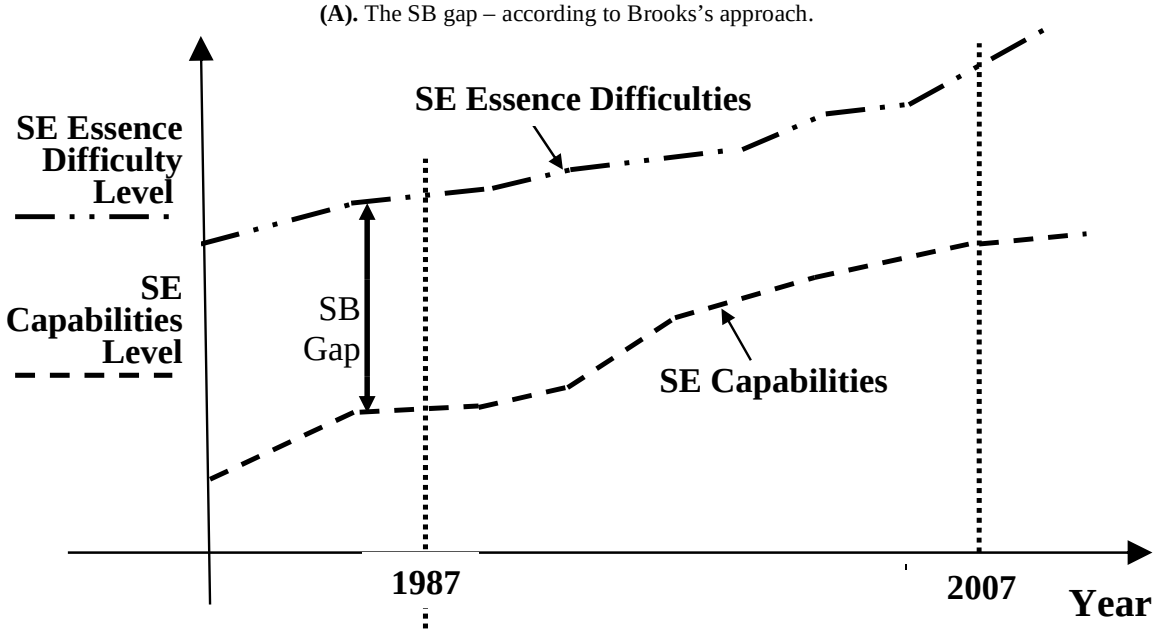
Growing application rates of open source software OSS, due to the development of culture and institutions dedicated to OSS implementation [61-63] have been noted in the last few years. Case studies of OSS development applying open source tools are encouraging, despite the identified interoperability and adaptability problems and other limitations of these tools [64]. The accumulation of open source components leads to the growing difficulty of maintaining open source libraries and repositories, especially the categorization of components. The development of tools for maintaining these libraries, including the automatic categorization of components, are the focus of current research projects [65].

5. DISCUSSION

The SB gap evolutionary model provides a comprehensive description and explanation to the changing behavior of the SB gap. The SB gap is defined, as a product



(A). The SB gap – according to Brooks's approach.



(B). The SB gap - according to the SB gap evolutionary model.

Fig. (1). The SB Gap – A Comparison.

of two independent processes of evolution that govern SE essence difficulties and SE capabilities. While a few parts of the evolutionary model have been discussed in several previous publications our model is a comprehensive one. The model explains the changes in the size of the SB gap. It increases and it decreases along the time as a result of development in the SE essence difficulties and SE capabilities. Thus, it enhances and in a way changes the approach of irreducible and unchanging SE essence difficulties presented by Brooks and others. Fig. (**1**) illustrates the comparison between Brooks' approach and the SB gap evolutionary model.

Are there ways to reduce the SB gap at the global level and at the software developer level? Some global initiatives to develop new technological tools and methodologies, already happened in the past through international professional organizations and may be expected in the future. Examining the software developer level, we find that the software developer has some ways to reduce the SB gap in his organization. Some reduction of the relevant SE essence difficulties may be achieved by ruling out unreasonable requirements listed in new project specification. The software developer could in many cases, adopt the project schedule to an applicable time table and avoid unbearable time pressures during the development. Additional route to avoid increase of SE essence difficulties is by controlling the change requirements board that will act as an effective filter for filtering out requirements that are not urgent or those that are expected to add negligibly to the project's contributions. The software developer may also affect the size of the SB gap by increasing the SE capabilities by adopting new and advanced SE tools.

6. CONCLUSIONS AND FUTURE RESEARCH

A theoretical-qualitative approach served as the basis for the irreducible and unchanging perception of SB gap, as identified by Brooks's1987 ideas [1], and the perception of

my evolutionary SB model. Empirical-quantitative research has not yet been conducted to support these perceptions. Empirical longitudinal studies will enable a better estimation of the extent of SE essence difficulties compared to SE accidental errors and improve the evolutionary model.

Some ideas for future empirical research:

1. Evaluation of the various stages of the software development life cycle as sources of SE essence difficulties. In other words, estimating the relative contribution of the requirement specification, the design stage, the programming and testing stages to SE essence difficulties.

2. The study the evolutionary behavior of SE essence difficulties and SE capabilities and the measurement of the average rate of enhancements over the last decades. Such empirical studies could contribute to SE theory and can, practically, direct the developers of SE technology to solve the important issues of SE essence difficulties, and to intensify the process of reducing the SB gap.

CONFLICT OF INTEREST

The authors confirm that this article content has no conflict of interest.

ACKNOWLEDGEMENTS

Declared none.

REFERENCES

[1] F. P. Brooks, "No silver bullet- essence and accidents of software engineering", *Computer*, vol. 20, no. 4, pp. 10-19, 1987.

[2] S. D. Fraser, F. P. Brooks, M. Fowler, R. Lopez, A. Namioka, L. Northrop, D. L. Parnas and D. Thomas, "No silver bullet reloaded – a retrospective on essence and accidents of software engineering", in *Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion (OOPSLA '07*), *New York, NY*, 2007, pp. 1,026-1,030.

[3] D. M. Berry, "The inevitable pain of software development, including extreme programming caused by requirements volatility", in *International Workshop on Time-Constrained Requirements Engineering (T-CRE), Essen, Germany*, 2002, pp. 9-19.

[4] D. M. Berry, "The inevitable pain of software development: why there is no silver bullet", in *Radical Innovations in Software and Systems Engineering in the Future, Proceedings of the 2002 Monterey Conference, Selected Papers*, M. Wirsing, S. Balsamo and A. Knapp, Eds., LNCS, Springer, 2004, pp. 50-74.

[5] D. M. Berry, "The software engineering silver bullet conundrum", *IEEE Softw.*, vol. 24, no. 2, pp. 18-19, 2008.

[6] H. J. Jeffrey, "Addressing the essential difficulties of software engineering", *J. Syst. Softw.*, vol. 32, no. 2, pp. 157-179, 1996.

[7] D. Spinellis, "Silver bullets and other mysteries", *IEEE Softw.*, vol. 23, no. 3, pp. 22-23, 2007.

[8] A. E. Bell, "Software development amidst the whiz of silver bullets", *Commun. ACM*, vol. 51, no. 8, pp. 20-24, 2008.

[9] M.Blaha, "Copper bullet for software quality improvement", *Computer*, vol. 37, no. 2, pp. 21-25, 2004.

[10] R. Glott, A.-K. Groven, K. Haaland and A. Tannenberg, "Quality Models for Free/Open Source Software – Towards the Silver Bullet?", in *The 36 EUROMICRO Conference on Software Engineering and Advanced Applications*, 2010, pp. 438-446.

[11] N. Sharma, K. Singh and D. P. Goyal, "Experience-based Software Process Improvement", in *Information Intelligence, Systems Technology and Management - 5th International Conference (ICISTM), Gurgaon, India*, 2011, pp. 71-80.

[12] S. D. Fraser and D. Manci, "No silver bullet: software engineering reloaded", *IEEE Softw.*, vol. 24, no. 1, pp. 91-95, 2008.

[13] D. Manci, S. D. Fraser and W. Opdyke, "No Silver Bullet: A Retrospective on the Essence and Accidents of Software Engineering", in *Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion (OOPSLA '07). New York, NY*, 2007, pp. 758-759.

[14] D. J. Harel, "Biting the silver bullet – toward a brighter future for system development", *Computer*, vol. 25, no. 1, pp. 8-20, 1992.

[15] B. Cox, "No silver bullet revisited", *Am. Prog. J.*, pp. 1-8, 1995.

[16] R. Dromey, "Climbing over the "No Silver Bullet" Brick Wall", *IEEE Softw.*, vol. 17, no. 2, pp. 118-120, 2000.

[17] S. D. Sue and I. Neamtiu, "Studying Software Evolution for Taming Software Complexity", in *Proceedings of the 21st Software Engineering Conference (ANSWEC)* Auckland, 6-9 April, 2010, pp. 3-12.

[18] G. Robles, J. J. Amor, G. Barahona and I. Herraiz, "Evolution and Growth in Large Libre Projects", in *Proceedings of the 2005 International Workshop on Principles of Software Evolution (IWPSE'05)*, 2005, pp. 1-10.

[19] W. M. Turski, "The reference model for smooth growth of software systems revisited", *IEEE Trans. Softw. Eng.*, vol. 22, no. 8, pp. 814-815, 2002.

[20] W. M. Turski, "Reference model for smooth growth of software systems", *IEEE Trans. Softw. Eng.*, vol. 28, no. 8, pp. 599-600, 1996.

[21] A. Israeli and D. G. Feitelson, "The linux kernel as a case study in software evolution", *J. Syst. Softw.*, vol. 83, no. 3, pp. 485-501,2010.

[22] M. W. Godfrey and Q. Tu, "Evolution in Open Source Software: A Case Study", in *The 14th IEEE International Conference on Software Maintenance ICSM"00)*, 2000, pp. 131-142.

[23] W. B. Arthur, "On the Evolution of Complexity", 1993, http//ideas.repec/p/wop/safiwp/93-11-070.html

[24] B. J. Williams, J. Carver and R. Vauglin, "Change Risk Assessment: Understanding Risks Involved in Changing Software Requirements", *The 2006 International Conference on Software Engineering Research and Practice*, ACM: New York 2006.

[25] M. A. Rahman, R. Razali and D. Singh, "A risk model of requirements change impact analysis", *J. Softw.*, vol. 9, no. 1, pp. 76-81, 2014.

[26] T. Mens, M. Wermelinger, S Ducasse, S. Demeyer, R. Hirshfeld and M. Jezayeri, "Challenges in Software Evolution", in *Principles of Software Evolution, Eight International Conferences on Software Evolution*, 5-6 September 2005, pp. 13-22.

[27] E. W. Duggan, "Silver pellets for improving software quality", *Inform. Res. Manage. J.*, vol. 17, no. 2, pp. 1-23, 2004.

[28] C. Jones, "*Applies Software Measurement, Global Analysis of Productivity and Quality*", 3rd ed., McGraw Hill: New York, 2008, pp. 265-268.

[29] C. Jones, "*Programming productivity*", McGraw Hill: New York, 1986, pp.150-152.

[30] C. Jones, "*The Technical and Social History of Software Engineering*", Addison-Wesley: Upper Saddle River, NJ, 2014.

[31] D. Garlan, R. Allen and J. Ockerbloom, "Architectural mismatch:why reuse is so hard", *IEEE Softw.*, vol. 12, no. 6, pp. 17-26, 1995.

[32] D. Garlan R. Allen and J, M, Ockerbloom, "Architectural mismatch: why reuse is still so hard", *IEEE Softw.*, vol. 26, no. 4, pp. 66-69, 2009.

[33] M. S. Andersen, "Architectural Mismstch Issues in Identity Management Deployment", in *ECSA'10 Proceedings of the Fourth European Conference on Software Architecture, New York*, 2010, pp. 31-33.

[34] A. Sutcliffe, G. Papamargaritis and L. Zhao, "Comparing requirements analysis methods for developing reusable components llibraries", *J. Syst. Softw.*, vol. 79, no. 2, pp. 273-289, 2006.

[35] K. Sherif, R. Appan and Z. Lin, "Resources and incentives for the adoption of systematic software reuse", *Int. J. Inform. Manage.*, vol. 26, no.1, pp. 70-80, 2006.

[36] R Anguswamy and W B Frakes "A Study of Reusability Complexity and Reuse Design Principles", in *Proceedings of ESEM'12 Proceedings of The ACM-IEEE international Symposium on Empirical software Engineering and Measurement*, 2012, pp. 161-164.

[37] P. Vitharana, "Risks and challenges of component-based software development", *Commun. ACM*, vol. 46, no. 8, pp. 67-73, 2003.

[38] M. Morisio, C. E. Seaman, A. T. Parra, V. R. Basili, S. E. Kraft, and S. E. Condon, "Investigating and ImprovIng a COTS-Based Software Development Process", in *International Conference of Software Engineering (ICSE, '00)*, 2000, pp. 31-40.

[39] A. Cechich and M. Piattini, "Early detection of cots component functional suitability", *Inform. Softw. Tech*., vol. 49, no. 2, pp. 108-121, 2007.

[40] M. Torchiano and M. Morisio, "Overlooked facts on COTS-based development", *IEEE Softw*., vol. 21, no. 2, pp. 89-93, 2004.

[41] K. R. P. H. Leung and H. K. N. Leung, "On the efficiency of domain-based cots product selection method", *Inform. Softw. Tech*., vol. 44, no. 12, pp. 703-715, 2002.

[42] L Mariani, "Dynamic detection of cots components incompatibility", *IEEE Softw*., vol. 24, no. 5, pp. 76-85, 2007.

[43] M Tivoli and P Inveradi, "Failure-free coordination synthesis for computer -based architecture", *Sci. Comput. Prog*., vol. 71, no. 3, pp. 181-212, 2006.

[44] S. Mahmood and A. Khan, "An industrial study on the importance of software component documentation:a system integrator's perspective", *Inform. Process. Lett*., vol. 111, no. 12, pp. 583-590, 2011.

[45] T. Wijsyassiriwardhane, R. Lai and K. C. Kang, "Effort estimation of component –based software development – a survey", *IETE Softw*., vol. 5, no. 2, pp. 216-228, 2011.

[46] J. Livari, "Why are case tools not used?", *Commun. ACM*, vol. 39, no. 10, pp. 94-103, 1996.

[47] L. C. Norman and D. Lending, "Case tools: understanding the reasons for non-use", *ACM SIGCPR Comput. Personnel*, vol. 19, no. 2, pp. 13-26, 1998.

[48] P. Campos and N. Numes, "Towards useful and usable interaction design tools: canon sketch", *Interact. Comput*., vol. 19, no. 5-6, pp. 597-613, 2007.

[49] B. Lundell, "Changing perceptions of case technology" *J. Syst. Softw*., vol. 72, no. 2, pp. 271-280, 2004.

[50] M. E. McMurtrey, V. Grover, J. T. C. Teng and N. J. Lightner, "Job satisfaction of information technology workers: the impact of career orientation and task automation in a case environment of management information systems" *J. Manage. Inform. Syst.*, vol. 19, no. 2, pp. 273-302, 2002.

[51] E. J. Barry, C. F. Kemermer and S. A. staughter, "How software automation affects software evolution: a longitudinal empirical analysis". *J. Softw. Maint. Evol-R.*, vol. 19, no. 1, pp. 1-31, 2007.

[52] J. Lee, S Kang an D Lee, "Survey of software testing practices", *IET Softw*., vol. 6, no. 2, pp. 275-282, 2012.

[53] J. Kasurinen, O. Taipale and K. Smolander. "Software test automation practice: empirical observations", *Adv. Softw. Auto*., Special Issue on Test Automation, pp. 1-18, 2010.

[54] A. Bertolino, "Software Testing Research: Achievements, Challenges and Dreams", in *Proceeding of FOSE'07 2007 Future of Software Engineering Conference*, 2007, pp. 85-103.

[55] S. Anzi, B. Dolby. J. Jensen and A. Moller, " A Framework for Automated Testing of Javascript Web Applications", in *Software Engineering (ICSE) 2011 33rd International Conference, Honolulu, HI*, pp. 571-580.

[56] R. Ramler and K. Wolfmaier, "Economic Perspectives in Test Automation: Balancing Automated and Manual Testing with Opportunity Cost", in *AST'06 Proceeding of the 2006 International workshop on automation of Software test*, 2006, pp. 85-91.

[57] D. Amaltifano, A. R. Pasolino, P Tramoltana, S. De Carmine and A. Memon,"Using GUI Ripping for Automated Testing of Android Applications", in *ASE'2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, 2012, pp. 258-261.

[58] W. K. Kista, E. S. Sundmark and D. K. Lundkvist,."Technical Debt in Test Automation", in *2012 IEEE Fifth International Conference on Software Testing ,Verification and Validation (ICST), Montreal, Canada*, 2012, pp. 887-892.

[59] P. A. da Mota Silveira Neto, I. do Carmo Machado, J. D. McGregor E. S. de Almeida and S. R. de Lemos Meira, "A systematic mapping study of software product lines testing", *Inform. Softw. Tech.* , vol. 53, no. 5, pp. 407-423, 2011.

[60] C. Neumann and C. Breidert, "The challenges of using open source as reuse strategy, upgrade", *Eur. J. Inform. Prof*., vol. 6, no. 3, pp. 38-42, 2005.

[61] S. V. Engelhard and A. Freytag, "Institutions, culture and open source", *J. Behav. Organ*., vol. 95, no. C, pp. 90-110, 2013.

[62] M. Zschoch, "The success of open source", *Can. J. Polit. Sci.*, vol. 40, no. 1, pp. 250-252, 2007.

[63] B. Fitzgerald, "The transformation of open source software", *MIS Quarterly*, vol. 30. no. 3, pp. 587-596, 2006.

[64] R. van Wendel de Joode and T. M. Egyedi, "Handling variety: the tension between adaptability and interoperability of open source software", *Comput. Stand. Int.*, vol. 24, no. 1, pp. 109-121, 2005.

[65] S. Kavaguchi, P. K. Gang, M. Matsushita and K. Inoue, "An automatic categorization system for open source repositories", *J. Syst. Softw*., vol. 79, no. 7, pp. 939-95, 2006.

Received: June 02, 2014 Revised: November 12, 2014 Accepted: December 12, 2014