# TOWARDS A FRAMEWORK FOR EVALUATING STRENGTHS AND WEAKNESSES OF SOFTWARE PROJECTS

**Sukhoo, A.**[1], **Barnard, A.**[2], **Eloff, M.M.**[2] **and Van der Poll, J.A.**[2]

[1]Central Informatics Bureau
Royal Road, Paillotte, Quatre Bornes, Mauritius. Tel: +230 201 2011.
E-mail: aneeravsukhoo@yahoo.com
[2]UNISA
School of Computing, UNISA, South Africa. Tel: +27 12 429 6817.
E-mail: barnaa@unisa.ac.za, eloffmm@unisa.ac.za or vdpolja@unisa.ac.za

## ABSTRACT

*Currently software projects are considered by some as being no different from traditional engineering projects. Software project management methodologies are developed with a generic concept in mind as is the case with Prince2 and PMBOK. Although these methodologies have a wider scope, there are certain specificities tightly bound to software projects that warrant the need for, in particular, project management methodologies that focus on the development of software projects. This paper presents these specificities in terms of strengths and weaknesses of software projects in contrast to traditional engineering projects. These strengths are factors to be considered seriously and advantage should be taken of them in the management of software projects. We speculate that these strengths may indeed present a plausible framework for the future analysis of different software project management methodologies. At the same time, project managers ought to be cautious about the weaknesses inherent in software projects.*

**Keywords**: Software, project management, strengths and weaknesses of software projects

## INTRODUCTION

Many project management methodologies have been developed with a view to address all types of projects. These methodologies are often said to be generic in nature and are expected to cut across various disciplines (Cockburn, 2000). Software projects are defined in much the same way as traditional engineering projects. Under the term traditional engineering project, we understand those projects associated with the creation of engineering artefacts that are not of a software nature.

Some definitions of the term "project" to be found in the literature include:
- A human endeavour which creates change, is limited in time and scope, has mixed goals and objectives, involves a variety of resources and is unique (Andersen, Grude, Haug and Turner, 1987)
- A complex effort to achieve a specific objective within a schedule and budget target, which typically cuts across organisational lines, is unique and is usually not repetitive within the organisation (Cleland and King, 1983)
- A one time endeavour to do something that has not been done that way before (Smith, 1985)
- A temporary endeavour undertaken to create a unique product or service (PMI, 2000; Schwalbe, 2000).

Taking these definitions into account, and in particular the approach suggested by PMI (2000) and Schwalbe (2000), Marchewka (2003, p. 9) then claims that project management *"is the application of knowledge, skills, tools, and techniques to project activities in order to meet or exceed stakeholder needs and expectations from a project"*.

All these definitions are malleable to be applicable to software as well as traditional engineering projects. In spite of such common definitions, it is our contention that differences between software projects and traditional engineering projects readily show up. These definitions may furthermore impact on the way in which software projects are handled and managed as the influence of the human factor is often ignored. We furthermore speculate that the strengths inherent in software projects should be taken advantage of to steer such projects towards successful completion within the set budget, time and quality standards. However, software projects also have intrinsic weaknesses that may hamper the proper management thereof, if these are overlooked. Project managers must constantly be alert to overcome or mitigate these weaknesses.

The remainder of this paper is structured as follows: Section two discusses typical strengths of software projects as opposed to traditional engineering projects. The third section elaborates on some of the weaknesses normally associated with software projects. We conclude this paper in section 4 by noting that the specificities associated with software projects, should be taken advantage of to steer such projects to successful completion.

## STRENGTHS OF SOFTWARE PROJECTS

Over the past two decades, several problems associated with software project development, contributed to the general failure of these projects to live up to user expectations, they were commonly delivered late, and they mostly ran over the set budget. The Standish Group studied 13,522 projects and reported thereon in the EXTREME CHAOS (2000) report. In particular this study determined that 23 percent of the surveyed projects failed, 49 percent did not meet the requirements and only 28 percent succeeded. Marchewka (2003, p. 6) identified the following as some of the contributing factors to software project failure: *incomplete requirements*, *lack of user involvement*, *lack of resources*, *unrealistic* user *expectations*, *lack of executive support*, and the *lack of* proper *planning*, among others.

Despite the problems related to managing software projects as explicated above, we advocate that software projects have unique strengths when compared, for example, to traditional engineering projects. Project management methodologies should emphasise these strengths to address some of the possible problems associated with software projects and their management. The generic nature of project management methodologies is to a large extent suitable for most projects, however in many cases customisation may be required in software project development.

An insight into the strengths associated with software projects, is important to focus attention on the relevant areas. In this section we investigate some of the more important strengths (Brooks, 1995; Hughes and Cotterell, 2002) associated with software project development in general:

### FLEXIBILITY

The inherent characteristics of software artefacts are often nebulous, and this more often than not imply that software can be modified with greater ease when compared to existing traditional engineering artefacts (Hughes and Cotterell, 2002). Customisation of software is frequently performed in an attempt to shorten the development schedule.

Whenever a software module is developed and presented to a user, certain modifications are normally requested. These requests may range from minor to major modifications, especially if the user requirements were not captured properly during the analysis phase. It is rare occurrence that a software product has to be redesigned and developed in its entirety due to erroneous (in case of user resistance), or badly captured, user requirements. Furthermore, it is often the case that bugs or other types of unanticipated errors are identified only when the software has been in operation for some time already, and consequently has to be fixed. This is normally carried out with an acceptable degree of ease and flexibility to the commensurate level of correctness.

In contrast to the general flexibility of software products discussed above, traditional engineering artefacts do not lend themselves easily to modifications without structural impact. A bridge, building, machinery or an electronic piece of equipment may not be easily modified, if this is even at all possible. A redesign of the products may, therefore, be cost-prohibitive and hence unfeasible to undertake. This reflects an advantage that the flexibility associated with software product alteration has over that of traditional engineering products.

### MOBILITY

Software can be transported at low or no cost at all since it can be rapidly transferred from one locale to another by utilising existing communication options. Trial versions of software may readily be downloaded over the Internet across thousands of kilometres at normal Internet access cost. If the user is satisfied with the trial version of the software, (s)he may decide to procure the registered version. Software can also be transferred using variable sized diskettes, CD ROMS, etc.

Traditional engineering artefacts on the other hand, very often cannot be easily relocated from one physical location to another. This is clearly the case for bridges, dams and concrete buildings. Once built, they stay in place, at least for the foreseeable future (assuming normal circumstances).

## REPLICATION

Replication of the same software product is very common and this feature accounts for reduction in cost and duration of implementation. Examples include the replication of freeware like UNIX and Linux together with commercially available registered software such as Windows XP®, Office XP® and StarOffice® under license agreements. Replication of software products is usually associated with minimum possible effort and can quickly progress round the globe. This is but one of the reasons that explains the prosperity of major software developers such as Microsoft™ and Oracle®. Project managers should bear in mind that the low cost of software replication could offer a competitive advantage.

Traditional engineering projects, however, generally yield products that cannot be replicated as readily as is the case with software projects (accept for some mass produced commercial items). Development of a similar traditional engineering product may require as much effort, time and cost as the original one.

## SCALABILITY

Software is scalable and can be adapted to new and emerging needs. A transaction processing system may be developed to record the day-to-day activities of an organisation. This system may then be used to develop a Management Information System for middle management to produce summarised structured reports. In order to analyse unanticipated situations, middle management may request a Decision Support System, which draws on the data from the underlying database. Higher up in a hierarchy, top management is especially concerned with an Executive Information System capable of presenting information in an executively summarised form (O’Leary and O’Leary, 2000).

Scalability may not be the case for the majority of traditional engineering products. Another floor cannot be readily added to a building unless it has been planned beforehand, in which case supporting iron bars and columns of the necessary material of predetermined diameter must be carefully selected.

## BACKUP

Backup is an effective means to minimise data and software loss. The backup procedure in the software domain is carried out at minimum cost and effort while the backup of a traditional engineering product may prove to be too expensive to maintain. Software backup is, among others, performed for security purposes and to maintain the integrity of data (Pfleeger and Pfleeger, 2003). In the case of software and associated data loss resulting from hardware malfunction, deliberate corruption or natural disasters, recovery procedures can be initiated from the backup version. Clearly, this is not the case for traditional engineering artefacts.

## REUSABILITY OF COMPONENTS

Program modules used in one project can be reused a number of times in other projects, thusly reducing development efforts resulting in time and cost savings. Reusability of components is a common practice in software development. Software developers ought to be well aware of the importance of a library of routinely used modules. It should not be necessary to “reinvent the wheel” whenever the same functionality must be incorporated in other software.

In the case of a traditional engineering product, however, a specific part cannot be removed from the said product and used in another product without usually rendering the original product non-operational.

## POSSIBILITY OF USING PROTOTYPES

Prototyping can be used to construct a final software product. This initiative reduces the effort and eventually the cost associated with the development of a software product. There are various prototype options to be used in software project development. An example includes evolutionary prototypes that are initially presented to the user during early development stages to clarify any misunderstanding between the user and the development teams. The envisioned result of this activity is to enable the capture of highly focused user requirements. This prototype is a simplified version of the final product that are rapidly developed and which will act as a model to show the user what can be expected at the final stage(s) of the project. It is in fact a scaled down version of the final product. An evolutionary prototype is modified in a stage-by-stage manner until it is approved by the user for final implementation (Davis, 1992). Development effort is not necessarily wasted since the evolutionary prototype is not discarded after it has been presented to the user. This may obviously not be the case for throwaway prototypes.

A traditional engineering prototype on the other hand is a representation that usually cannot easily be evolved to constitute the final product. For example, a model of a ship, a building or a car cannot readily be used to build the end product. Users will normally be charged for such development efforts, even if these models are discarded.

## SOFTWARE EVOLUTION

A software project can evolve over time, resulting in new releases that outperform older versions. New releases tend to take advantage of prevailing technology where software evolution is then commonly used for improving software. For example, the Microsoft Windows® operating system and Microsoft Office® products are subject constantly to new releases which incorporate new functionalities, better user interface designs and performance and also take advantage of new technologies like new and faster hardware as well as better communication facilities etc. An example of this is provided by Microsoft's™ Next-Generation Secure Computing Base (NGSCB), code-named *"Longhorn"®*, a new hardware and software design that enables new kinds of secure computing capabilities for providing *"enhanced data protection, privacy and system integrity"* (Microsoft Next-Generation Secure Computing Base – Technical FAQ, 2003). In particular, Microsoft™ describes its Security Support Component (SCC) as *"a new PC hardware component that will be introduced as part of the NGSCB architecture … is a module that can perform certain cryptographic information and securely store cryptographic keys … that provide seal storage and attestation function"* (Microsoft Next-Generation Secure Computing Base – Technical FAQ, 2003). Microsoft™ furthermore claims that the addition of such hardware and software technology to the Microsoft Windows® platform has the associated advantages of offering increased security, privacy and system integrity features.

Traditional engineering artefacts, like bridges and buildings are not necessarily built with an attempt to evolve them or to take advantage of new technologies to improve them. Where the products are evolved to newer models they do not readily replace the original ones. Usually the older and newer versions coexist with each other. Examples include new models of cars and more recent versions of electronic equipment. However, in contrast to software products, traditional engineering products are generally used over a longer period.

## WEAKNESSES OF SOFTWARE PROJECTS

Software projects suffer from certain weaknesses when compared to traditional engineering projects. These weaknesses must be carefully dealt with in order to achieve successful software project completion. Therefore, any proposed software project management methodology should ideally take cognisance of inherent weaknesses in software projects (Olson, 2004).

Examples of weaknesses related to the development of software projects, are:

### INVISIBILITY

In the case of the construction of a physical artefact such as a bridge, the progress of work can actually be seen. In the case of software projects this progress is not immediately visible (Hughes and Cotterell, 2002, p.3). One major weakness is the invisibility or poor visibility of progress during the development of a software project. The user may not be able to conceptualise the software until it is complete. If the user is not satisfied, a great deal of effort may have already been expended, ultimately resulting in project failure. Strategies of software project management must therefore strive to make the invisible visible through the judicious use of appropriate software or project management tools and methods.

In contrast to the invisibility usually associated with the development of software products, traditional engineering product development on the other hand is normally visible as it lends itself to observation of tasks while they are carried out.

### Complexity

Software artefacts are inherently more complex than traditional engineering artefacts (Brooks, 1987). Hughes and Cotterell (2002, p. 3) states that per dollar, the complexity associated with software project development is more than that of a traditional engineering project. At the inception of a project, inaccurate or ambiguous information gathered from the users may constitute a major threat to subsequent stages of the software development process. A logical error may propagate throughout the development process and can go undetected until the last stages. In 1976 already Fagan (1976) stated that errors due to the complexity inherent in software projects, can even go undetected for a long time after the final product has become operational, and that the impact may then be considerable.

In traditional engineering projects, complexity is relatively lower owing to better visibility of progress and usually well-understood requirements from the onset.

## DIFFICULTY REGARDING ESTIMATION

Estimation of the software production process in terms of cost and duration is perceived as a difficult task (Hughes, 1996). An inaccurate estimation may lead to cost overrun, failure to meet deadline or degradation in quality (Yamaura and Kikuno, 1999). Several methods for software estimation include Expert Judgement, Estimation by Analogy, COCOMO, Function Point Analysis, etc. (Schach, 2002). Agarwal, Kumar, Mallick, Bharadwaj and Anantwar (2001) points out that the causes of *"poor and inaccurate estimation are: (a) imprecise and drifting requirements; (b) new software projects are nearly always different form the last; (c) software practitioners don't collect enough information about past projects; and (d) estimates are forced to match the resources available"*. However, during the early stages of the project, it is often difficult to estimate the number of components constituting the product and the time to be spent on each. A start-up software development company will have to test an estimation method for some time before making appropriate estimations. It is often the case that users are not satisfied with high software costs proposed by software developers. This is probably due to invisibility of the amount of work required.

Traditional engineering artefact development has been in operation for eons and consequently the visibility of the amount of work concerned, price of components, and existing well-defined complexity measures, may account for relatively easy negotiations regarding cost and time schedules between the user and the developer.

## DYNAMIC NATURE

The dynamic nature of software becomes a limitation in certain cases where new versions are no longer compatible or suitable to run on older versions of operating systems software or hardware. For instance, a 16-bit software product may not function on a 32-bit operating system or Microsoft Office XP® will not run on a personal computer having an 80486 processor. Software developers are also reluctant to support older versions of software and therefore, users are forced to review hardware and software regularly and upgrade both at the same time.

## INTANGIBILITY

The intangibility feature associated with software makes it difficult for potential clients to purchase a certain product (Olson, 2004). Sometimes free trial versions are offered to assist clients to reach a decision regarding purchase of the registered version. When this is not the case, especially when the requirements of the user cannot be matched with existing software, the software has to be tailor-made. If an evolutionary prototyping method is not used, the user will not be able to appreciate the eventual functionality of the software until it is completed.

On the other hand, the tangibility associated with a traditional engineering product, is evident throughout the construction process. The client may watch the progress of the project on site and even communicate any issues of dissatisfaction to the developer immediately. This may enhance client satisfaction at the closure of the project.

## REGULAR UPGRADES

Regular upgrades improve the quality, performance and efficiency of software, but in the long run becomes expensive. Sometimes older versions of software are no longer maintained or the maintenance service is offered at a higher cost. The user is forced to bear this cost to stay competitive. Often an upgrade regarding software as well as hardware is required yearly, however, this may not always be affordable due to the high cost involved and the frequency of said upgrade. For example, Microsoft Office® is upgraded virtually annually and organisations with a large number of personal computers may find it too costly to upgrade the software every year in exchange for increased productivity.

Traditional engineering products do not require frequent upgrades, as is the case with software products. Once a building is completed, it is planned to be in that state for years. One can argue with a degree of certainty that additional floors in the building will not be added annually.

## BUGS

It is often observed that software suffers from bugs and flaws from the time it is released, requiring the software developer to be called upon to correct them. Sometimes, this affects the confidence of a user towards a software developer. One can speculate that the greater the number of bugs, the greater the loss of confidence. Therefore, a software developer should aim at eliminating these bugs as soon as possible requiring additional effort throughout the development and testing phases. Bugs can often be attributed to the inherent complexity of software, as argued in a previous subsection on complexity.

On the other hand, flaws in traditional engineering products are not so common after release due to a number of factors. These factors include, but are not limited to, the better visibility during the development process, an acceptable level of complexity and existing formal methods to assist the development process that are entrenched in the subject theoretic literature, for example refer to the Whall verification debate (Le Charlier and Flener, 1998; Woodcock and Davies, 1996).

## DIFFICULTY TO ADD PEOPLE TO A DELAYED PROJECT

It is difficult to meet project deadlines by adding new people towards the end of the development cycle of a software project. According to Brooks' Law, *"putting more people on a late job makes it later"* (Brooks, 1995). Generally new people added at late stage during software development, has to spend time learning what has been completed up to that point. Additional effort is also required regarding management, co-ordination and communication.

It is often easier to add new people on a traditional engineering project that is behind schedule. For example in the construction of a bridge or a building, adding more people will generally speed up the construction process. However, the number of additional people must be carefully chosen so as not to impact on management overheads.

### Training

Software projects, most often if not always, need training sessions to be incorporated to use the end product (software). In the case of traditional engineering products, this may often not be the case. Generally, the user does not require extensive training to use a bridge, building or road. Sometimes, following the instructions presented in a user manual, will be sufficient to operate a piece of equipment.

## CONCLUSION AND FUTURE WORK

Project management is not a new concept! Throughout the documented history of the human race there have been projects that attempted to do more than a single person could accomplish alone. Think of examples such as the pyramids in Egypt, the cathedrals in Europe, and the Great Wall of China! Nowadays effective management of projects in the Information Technology environment is seen to be increasingly important as a discipline.

However, project management methodologies, which are generic in nature, have been developed to help organisations cope with projects across various disciplines. Although these methodologies have a wider scope, there are certain specificities tightly bound to software projects. These specificities highlight the need for project management methodologies that focus on the development of software projects. In this paper we investigated and documented the strengths and weaknesses associated with software projects as opposed to traditional engineering projects.

This paper constitutes part of a larger research project focusing on a critical analysis of the situation pertaining to software project management in Mauritius. The use of project management methodologies, tools and techniques faced by software developers in Mauritius in particular, were analysed and reported on in another paper (Sukhoo, Barnard, Eloff and Van der Poll, 2004). Given the assumption of economic rationality and cultural differences and the need to cope with political and community demands on a project's resources (Murithi and Crawford, 2003), a new software project management methodology for Mauritius, is envisaged. With regards to the strengths and weaknesses of software projects explored in this paper, such methodology should take cognisance thereof to exploit the strengths and eliminate the weaknesses where possible.

## REFERENCES

1. Agarwal, R.; Kumar, M.; Mallick, S.; Bharadwaj, R.M. and Anantwar, D. (2001). Estimating software projects. *Software Engineering Notes*; 26(4), 60-67.
2. Andersen, E.S.; Grude, K.V.; Haug, T. and Turner, J.R. (1987). *Goal directed project management*. Kogan Page/Coopers and Lybrand, London, UK. Cited in Turner, J.R. and Müller, R. (2003). On the nature of the project as a temporary organisation. *International Journal of Project Management;* 21(1), 1-8.
3. Brooks, F.P. (1995). *The mythical Man-month: Essays on software engineering,* anniversary edition; Addison-Wesley: Reading, MA, US.
4. Brooks, F.P. (1987). No Silver Bullet: Essence and Accidents of Software Engineering. *IEEE Computer;* 20(4), 10-19.

5. Cleland, D.I. and King W.R. (1983). *Systems analysis and project management*. Mc Graw-Hill: New York, US. Cited in Turner, J.R. and Müller, R. (2003). On the nature of the project as a temporary organisation. *International Journal of Project Management;* 21(1) 1-8.
6. Cockburn, A. (2000). Selecting a project's methodology. *IEEE Software*; 17(4), 64-71.
7. Davis, A. (1992). Operational prototyping: A new development approach. *IEEE Software;* 9(5), 70-78.
8. Extreme Chaos Report. (2000). *Extreme Chaos Report*. Compiled by The Standish Group. Accessed 20.11.2003, online at: http://www.standishgroup.com/.
9. Fagan, M.E. (1976). Design and code inspections to reduce errors in program development. *IBM Systems Journal*; 15, 182-211.
10. Hughes, R.T. (1996). Expert Judgement as an Estimating Method. *Information and Software Technology*. 38, 67-75.
11. Hughes, B. and Cotterell, M. (2002). *Software Project Management,* 3rd edition. McGraw-Hill: London, UK.
12. Le Charlier, B. and Flener, P. (1998). Specifications are necessarily informal or: Some more myths of formal methods. *The Journal of Systems and Software*; 40(3), 275-296.
13. Marchewka, J.T. (2003). *Information technology project management: Providing measurable organizational value*. John Wiley and Sons: Hoboken, NJ, US.
14. Microsoft Next-Generation Secure Computing Base – Technical FAQ (2003). Accessed 20.11.2003, online at: http://www.microsoft.com/technet/security/news/NGSCB.asp?frame=true.
15. Murithi, N. and Crawford, L. (2003). Approaches to project management in Africa: implications for international development projects. *International Journal of Project Management*; 21(5); 309-319.
16. O'Leary, T.J. and O'Leary, L.I. (2000). *Computing Essentials*. McGraw-Hill: New York, US
*17.* Olson, D.L. (2004). *Introduction to information systems project management*, 2nd edition. McGraw-Hill/Irwin series: Boston, MA, US.
18. PMI 2000. *A guide to the project management body of knowledge*, 4th edition. Project Management Institute: Newton Square, PA, US. Cited in Turner, R.J. and Müller, R. (2003). On the nature of the project as a temporary organisation. *International Journal of Project Management;* 21(1) 1-8.
19. Pfleeger, C.P. and Pfleeger, S.L. (2003). *Security in Computing*. Prentice Hall: Upper Saddle River, NJ, US.
20. Schach, S.R. (2002). *Object-oriented and classical software engineering*. 5th edition, McGraw-Hill: New York, US.
21. Schwalbe, K. (2000). *Information technology project management.* Course Technology, Thomson Learning: Cambridge, MA, US.
22. Smith, B. (1985). Chapter in book: Project concepts, in *Effective project administration*. Institution of Mechanical Engineers: London, UK. Cited in Turner J.R. and Müller, R. (2003). On the nature of the project as a temporary organisation. *International Journal of Project Management;* 21(1) 1-8.
23. Sukhoo, A.; Barnard, A.; Eloff, M.M. and Van der Poll, J.A. (2004). A survey of project management tools, techniques and methodologies used in mauritius: the current status. To appear in the *Proceedings of the 2004 PMSA Global Knowledge Conference: "Current Research" Stream.*
24. Woodcock, J. and Davies, J. (1996). *Using Z: Specifications, refinement, and proof.* Prentice-Hall: London, UK.
25. Yamaura, T. and Kikuno, T. (1999). A framework for top-down cost estimation of software development. *Proceedings of the twenty-third annual international Computer Software and Applications conference*, 322-323.