Defining and Measuring Software Sustainability: Towards An Empirical Framework for Evaluation at the Architectural Level

Abstract

Software sustainability has been identified as one of the key challenges in the development of scientific and engineering software as we move towards new paradigms of research and computing infrastructures. However, it is suggested that sustainability is not well understood within the software engineering community, which can led to ineffective and inefficient efforts to address the concept or result in its complete omission from the software system. This paper proposes a definition of software sustainability and considers how it can be measured empirically in the design and engineering process of software systems.

Keywords

Architectural-reasoning; software architectures; software engineering; software quality; software sustainability; non-functional requirements.

Introduction

> "It was six men of Indostan
> To learning much inclined,
> Who went to see the Elephant
> (Though all of them were blind),
> That each by observation
> Might satisfy his mind."
>
> Blind Men and the Elephant, John Godfrey Saxe

The parable of the Blind Men and the Elephant[1] tells the tale of a group of six blind men who touch only one part of an elephant in order to learn what it is like. Based on their individual experience they suggest that the elephant is like a wall, spear, snake, tree, fan or rope. They then compare their experience and learn that they are in complete disagreement. This paper argues that the current understanding of software sustainability is similar to the parable of the Blind Men and the Elephant in that there is no agreed definition of what software sustainability means and how it can be measured or demonstrated. Software sustainability is a vague term and is not well understood within the software engineering community with individuals, groups and organization holding diametrically opposed views. However, Penzenstadler[2] states that without a clear and commonly accepted definition of what sustainability means, contributions remain somewhat insular and isolated, which can lead to ineffective and inefficient efforts to address the concept or result in its complete omission from the software system. This paper proposes that software sustainability can be considered as a composite, non-functional

requirement that can be analyzed and evaluated at the architectural-level to allow architectural-level reasoning about the software system(s). Section 2 of the paper sets the context for the discussion by examining the concept of sustainability. This lays the foundation for Section 3, which considers how sustainability can be defined as a composite, non-functional requirement. Section 4 considers the role of architectural evaluation methods as an empirical framework in reasoning about sustainability at an architectural level in the software development process, outlines how scenarios could be used to develop a set of measures, and positions this in terms of the increasing use of agile methods in scientific computing. In Section 5, conclusions are drawn and future directions are outlined.

Software Sustainability?

In recent years, the traditional approaches of research of experimentation and theory have been joined by large-scale computational simulation and data-intensive science, commonly referred to as the third and fourth paradigms of science[3]. For example, domain scientists are utilizing state of the art computational methods combined with atomistic molecular dynamic simulations of DNA circles to understand the role of DNA topology and supercoiling in genetic control[4]. Other examples include civil engineers utilizing artificial intelligence approaches with sensor networks to monitor water systems[5]. However, these new approaches to research are highly dependent on software systems, which are increasingly complex in nature and operate in evolving, distributed e-infrastructure eco-systems. In addition, the emergence of service-oriented computing where software is composed of loosely coupled services with the ability to bind to these services dynamically at runtime i.e. ultra late binding, which allows for a system to respond to changing requirements represents a significant paradigm shift in the way that software and hardware are not only developed but are also utilized by end-users[6]. A major challenge in developing sustainable computational science and engineering software within such environments is how to integrate it with existing components, services and systems that were not originally designed to interact with each other; this includes both software and hardware. Similarly, the 'as a Service' paradigm such as Software as a Service (SaaS), Infrastructure as a Service (IaaS) etc., as part of the nomenclature of cloud computing presents new challenges.

As an area of research, software sustainability is receiving increasing attention with a significant increase in research output in the last few years[7]. Its importance has been underlined by recent funding initiatives from the National Science Foundation and the Engineering and Physical Sciences Research Council (EPSRC) in the UK combined with the establishment of the Software Sustainability Institute and the emergence of a number of workshops dedicated to the topic of sustainable software and systems. This leads us to the question of what is software sustainability? Software sustainability is a rather ambiguous concept and a number of definitions have been proposed. The Oxford English Dictionary[8] defines sustainability as 'the quality of being sustained', which in turn is defined as 'capable of being endured' and 'capable of being 'maintained'. Endured being defined within the context of this paper as 'continuing to exist' and maintained as 'being supported'

[8]. Seacord et. al.,[9] view sustainability in relation to 'all activities related to software evolution and the ability to modify a software system based on stakeholders changing requirements'. This perspective accords with the OED definition of maintainability. However, they argue that there is a strong dependency on a range of other factors including the organization, developers, end-users, the operational domain in which the software operates as well as other software artifacts including the architecture, design documentation, and test scripts. The Software Sustainability Institute define sustainability as 'software you use today will be available - and continue to be improved and supported - in the future' which implicitly suggests that sustainability is concerned with concepts of availability, extensibility, and the maintainability of the software[10]. This aligns with the previous definitions, which emphasize the concepts of maintainability, extendibility and evolvability as being core underpinnings of sustainability. Calero, Bertoa and Moraga[11] define sustainable software as 'a mode of software development in which resource use aims to meet product software needs while ensuring the sustainability of natural systems and the environment'. This definition goes beyond the focus of the software artifact and highlights how software development practices themselves can affect the sustainability of society, economy, and the environment. This suggests that a broader definition may be required to encompass how software is developed in a sustainable manner. A number of frameworks have also been proposed for defining sustainability without specific reference to the field of software engineering[12-13]. Despite the numerous definitions that exist for sustainability most are either too vague or limited in their scope and any consensus within the field of software engineering has yet to be achieved. When we consider software sustainability in terms of how it is loosely defined by the Software Sustainability Institute with implicit references to the concepts of availability, extensibility, and maintainability, it naturally leads us to consider it as a prime candidate to be classified as a composite, non-functional requirement. Such a classification allows us to move from the focus from thinking about how we sustain existing software, to understanding how we can develop sustainable software in the future.

Software Sustainability as a Non-Functional Requirement

In software engineering, non-functional requirements or software quality attributes can be defined as 'the degree to which a system, component or process meets a stakeholders needs or expectations'[14]. Non-functional requirements express desired qualities of the system to be developed and refer to both observable qualities and also to internal characteristics. McCall et. al.,[15] proposed a taxonomy which distinguished between two levels of quality attributes: quality factors and quality criteria. Quality factors are external attributes and can be only measured indirectly. Examples of quality factors include correctness, efficiency, flexibility, integrity, interoperability, maintainability, portability, reliability, reusability, testability, and usability. Quality criteria can be measured either subjectively or objectively by combining the rating for the individual quality criteria that affects a given quality factor, and then a measure can be obtained to assess the extent to

which that quality factor can be satisfied. Quality factors can be broadly categorized into three classes (Fig. 1):

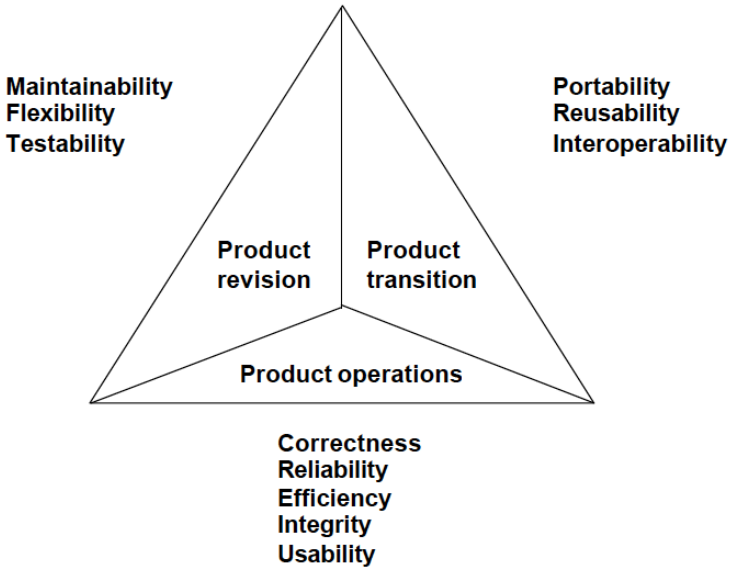

Figure 1: McCall's Software Quality Factors

We propose that software sustainability should be considered in a similar manner to the concept of dependability[16]; a measure of a system's availability, integrity, maintainability, reliability, and safety where the attributes of dependability are defined as:

- Availability: readiness for correct service;
- Integrity: the absence of improper system alteration;
- Maintainability: undergo modifications and repairs;
- Reliability: continuity of correct service;
- Safety: the absence of catastrophic consequences on the user(s) and the environment.

These attributes are then combined with the concepts of threats and failures to create the composite, non-functional requirement of dependability. How might a similar approach work for software sustainability? We propose that software sustainability can be defined as 'a measure of a systems extensibility, interoperability, maintainability, portability, reusability, scalability, and usability' where the attributes are defined as:

- Extensibility: a measure of the software's ability to be extended and the level of effort required to implement the extension;
- Interoperability: the effort required to couple software systems together.

- Maintainability: the effort required to locate and fix an error in operational software;
- Portability: the effort required to port software from one hardware platform or software environment to another;
- Reusability: the extent to which software can be reused in other applications;
- Scalability: the extent to which software can accommodate horizontal or vertical growth.
- Usability: the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use.

If we accept that the concept of sustainability goes beyond the software artifact itself then other quality attributes such as efficiency may be appropriate candidates:

- Efficiency: the amount of computing resources and code required to execute a function.

Defining sustainability as a non-functional requirement is a position supported by Calero, Bertoa, and Moraga[11]. However, they do not explicitly state which attributes contribute directly to the concept of sustainability, rather they believe that it must be related in someway to the ISO/IEC 25010 quality model[17]. One of the principal challenges in defining sustainability as a composite, non-functional requirement is how to develop appropriate measures and metrics for the identified quality factors to demonstrate that they have been addressed in a quantifiable way. For example, the reliability of a software system cannot be directly measured. To achieve a measure of software reliability requires directly measuring the number of defects encountered. Here we can distinguish between the terms measures and metrics where metrics are a system of measurement through which the merits of an entity can be assessed and measures may contribute to a metric as a set of quantitative values within the system. McCall's[15] approach to this problem was to define a set of metrics and develop expressions for each quality factor according to the following formula:

$$F_q = c_1 m_1 + c_2 m_2 + \ldots + c_n m_n$$

Equation 1: McCall's Software Quality Formula

where F_q is a quality factor, c_n are regression coefficients and m_n are the metrics that affect that quality factor. Unfortunately, many of the metrics McCall defined can only be measured subjectively. However, the metrics may be used as a form of checklist to grade subjective specific aspects of the software. Whether existing measures and metrics of quality attributes are appropriate or new measures and metrics are required for evaluating software sustainability is unclear. As a result, what measures and metrics are suitable to demonstrate software sustainability is an open research question. However, software quality attributes are the most

neglected elements of software projects due to a perfect storm of influences including the primary focus on functional capabilities, the effort associated with the process of extracting and refinement, and a lack of appropriate methods and tools[18]. For software to be sustainable it needs to be both useful and adaptable as stakeholders requirements, technology and environments evolve and change. Solutions to these issues must be planned and it is appropriate that these issues be addressed at the architectural-level. How this might be achieved is discussed in the following section.

Software Sustainability & Software Architectures

As a basis for discussion we propose that software architectures and architectural evaluation methods provide a potential mechanism for reasoning about software sustainability at an architectural level. This leads us to the question of what are software architectures and why are they useful for developing sustainable software? The architecture is the foundation of any system. It expresses the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its conceptual design and evolution[19]. It is a design artifact or blueprint of how the system will be built. However, software architectures are often the by-product of the software development process rather than providing a solid foundation upon which the software is built. This is particularly true in academic research-intensive environments where the focus is on proof of concept or is highly experimental in nature, and the primary driver is implementing functional requirements rather than engineering the best solution. As a result, software created in academic research institutions often exhibit a number of characteristic flaws and are generally not sustainable beyond the lifetime of a given project or usable by other researchers. Clements, Kazman, and Klien[20] argue that successful software system development and evolution is highly dependent on making informed decisions at the architectural level. This is a position supported by Koziolek[21] who argues that the quality of software architectures determines sustainability. To achieve this they suggest that scenario-based software architecture evaluation methods can support the analysis of sustainability. However, their own analysis highlights the limitations of existing methods and they suggest that methods and metrics should be integrated into a new method.

A number of architectural evaluation methods exist which provide a structured approach to evaluating how well an architecture meets stakeholder's requirements in terms of the quality attributes that the architecture exhibits. Their purpose is to analyze a candidate architecture in order to identify potential risks and to verify that the non-functional requirements have been addressed in the design. However, they differ significantly in their focus and the number of attributes employed. For example, while the Software Architecture Assessment Method[22] and its variants focus on singular quality attributes, other methods such as the Architecture Tradeoff Analysis Method[23] include multiple attributes, which is suggested contributes to a better understanding of the strengths and weaknesses of the overall architecture and its constituent parts. The key concept underpinning

these methods is a set of scenarios that are important to stakeholders' and allow the systems properties to be estimated. Scenarios are primarily used in requirements engineering to provide a personalized, fictional story with characters, events, products and environments in which stakeholders can engage during the design and evaluation process[24]. They also provide system designers with a method to explore ideas and identify the potential ramifications of specific design decisions. To address the limitations of the existing architectural evaluation methods, Venters et. al.,[25] proposed an architectural evaluation framework (AEF) that utilizes scenarios in order to understand how to achieve interoperability for military capability. While the use of scenarios is not inherently different from existing methods the framework was built upon two measures derived from NATO's Measures of Merit (MoM):

- Measures of Performance: verify an individual system against its service specification and are independent of the scenario allowing the results to be compared with services that provide the same or similar functionality.
- Measures of Effectiveness: are dependent on the scenario and provide a measurement of how well a system accomplishes its assigned tasks within a specific context and provides a level of confidence.

Key quality attributes included agility, dependability, availability, and interoperability. Specific military quality attributes included survivability and lethality where survivability in this context was concerned with the ability to remain mission capable after a single engagement, and lethality was the effectiveness of a weapon's system in all environments against the full spectrum of battlefield threats. While we do not advocate that these measures are necessarily applicable to the topic of software sustainability it illustrates that scenarios can be used to derive a set of meaningful measures that allow reasoning at an architectural-level. This is a view supported by Sehestedt, Cheng, and Bouwers[26] who state that software architectures and their representations in models are instrumental in achieving sustainability and the fulfillment of requirements. They propose seven metrics, which characterize the completeness, consistency, correctness and clarity of the documentation within views of architecture models and architectural decisions:

- Decomposition quality;
- Best practices adherence;
- View consistency;
- Rationalization completeness;
- Requirement fulfillment;
- Change scenario robustness;
- Decision traceability.

However they acknowledge that the challenge in designing such metrics is that architecture models are generally not formal models and that expert knowledge is required for computing the metrics. One of the biggest hurdles to this approach is

that the use of software architectures and evaluation approaches are still reluctantly used in practice and are not integrated with architecture-level metrics when evaluating implemented systems, which limits their capabilities[21]. Similarly, the selection of the most appropriate methods is highly dependent on the context in which the architecture is being evaluated and the quality attributes being addressed[24].

Anticipation of software evolution indicates that an agile approach to development may be desirable, or indeed necessary. The most significant implication of this for the proposed framework is that architecture design – and its evaluation – is not a one-off event that happens early in the design phase, but a process that happens continuously. Ensuring quality when using agile methods relies considerably on automation of development processes such as testing. Capturing software sustainability within quantifiable measures and metrics is therefore potentially very powerful, as it opens up the possibility of monitoring certain aspects of it automatically.

Conclusions

> "And so these men of Indostan
> Disputed loud and long,
> Each in his own opinion
> Exceeding stiff and strong,
> Though each was partly in the right,
> And all were in the wrong!"
>
> Blind Men and the Elephant, John Godfrey Saxe

In this paper we present our ideas as a basis for discussion in order to consider how we can address the challenges associated with developing sustainable software. We propose that software sustainability is a composite, non-functional requirement that is a measure of a number of core quality attributes including extensibility, interoperability, maintainability, portability, reusability and scalability. In addition, we suggest that software architectures and architectural evaluation methods are integral to the development of sustainable software. Complex software systems can only be built when we architect them using existing as well as newly engineered parts that provide the required overall capabilities. During the development and evolution of such software, the architecture plays a crucial role in defining the relations between these parts. It permits the decomposition of software into manageable parts and to compose the software from existing or adapted parts and enables the cost-effective engineering of software by multiple teams. Architecting sustainable science and engineering systems essentially means finding the right trade-off between the attributes and the various other requirements imposed on the system. Architectural representations of systems can be effective in understanding broader system concerns by abstracting away from system details, hence the trend for reasoning about quality attributes at the architectural level. A key task in this activity will be the derivation of suitable measures and metrics to be used for

evaluating the architectures, addressing both functional and non-functional concerns. It is important to define measures and metrics that truly quantify the characteristics of the architectures they intend to assess. This should include formalizing intuitive ideas of measures and metrics, which follow a rigorous process of validation. The parable of the Blind men and the Elephant[1] has been used to illustrate a range of different purposes including the need for improved communication and respect for different perspectives. Each of the six blind men has their own perspectives as to what they have observed. We propose that the development of a software sustainability architectural evaluation framework would assist in facilitating a greater holistic view of software sustainability.

References

[1]. Saxe, J. G. 1963. The blind men and the elephant. McGraw-Hill.

[2]. Penzenstadler, B., and Femmer, H. 2013. Towards a Definition of Sustainability in and for Software Engineering. In: SAC'13: Proceedings of the 28th Annual Symposium on Applied Computing.

[3]. Hey, T., Tansley, S., and Tolle, K. 2009. The Fourth Paradigm: Data-Intensive Scientific Discovery. Microsoft Research.

[4]. Mitchell J. S., Laughton C. A., and Harris S. A. 2011. Atomistic simulations reveal kinks, bubbles and wrinkles in supercoiled DNA. Nucleic Acids Research. 39, p: 3928-3938.

[5]. Mounce, S. R., Mounce, R. B., and Boxall, J. B. 2011. Novelty detection for time series data analysis in water distribution systems using Support Vector Machines. Journal of Hydroiformatics, 13 (4), pp 672-686.

[6]. Tsai, W. T. 2005 Service-oriented system engineering: A new paradigm. In SOSE'05: Proceedings of the 2005 IEEE International Workshop on Service-Oriented System Engineering. IEEE Computer Society. p. 3-6.

[7]. Penzentadler, B., Bauer, V., Calero, C., and Franch, X. 2012. Sustainability in Software Engineering: A Systematic Literature Review.

[8]. Oxford English Dictionary. 2012. Oxford Dictionaries.

[9]. Seacord, R.C., Elm, J., Goethert, W., Lewis, G. A., Plakosh, D., Robert, J., Wrage, L., and Lindvall, M. 2003. Measuring software sustainability. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA.

[10]. Software Sustainability Institue: http://www.software.ac.uk/about.

[11]. Calero, C., Bertoa, M. F., and Moraga, M. A. 2013. Sustainability and quality: Icing on the cake. In RE4SuSy: Second International Workshop on Requirements Engineering for Sustainable Systems

[12]. Burger, P., and Christen, M. 2011. Towards a capability approach of sustainability. Journal of Cleaner Production, 19 (8), pp: 787–795.

[13]. Seghezzo, L. 2009. The five dimensions of sustainability. Environmental Politics. 18(4), pp: 539-556.

[14]. IEEE. 1990. IEEE Standard Glossary of Software Engineering Terminology, IEEE Std. 610.12-1990.
[15]. McCall, J. A., Richards, P. K. and Walters, G. F. 1977. Factors in software quality: concepts and definitions of software quality. RDAC-TR-77-369.
[16]. Avizienis, A., Laprie, J-C., Randell, B., and Landwehr, C. 2004. Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing, 1, pp: 11-33.
[17]. ISO/IEC 25010. 2010. Systems and software engineering - Software product Quality Requirements and Evaluation (SQuaRE) - Software product quality and system quality in use models.
[18]. Brosseau, J. 2010. Software quality attributes: Following all the steps. Clarrus Consulting Group Inc.
[19]. ISO/IEC 42010:2007, 2007. Systems and Software Engineering: Recommended Practice for Architectural Description of Software-Intensive Systems, International Organization for Standardization.
[20]. Clements, P., Kazman, R., and Klien, M. 2002. Evaluating Software Architectures: Methods and Case Studies. Addison-Wesley Professional.
[21]. Koziolek, H. 2011. Sustainability evaluation of software architectures: a systematic review. In: QoSA-ISARCS'11: Proceedings of the joint ACM SIGSOFT conference on quality of software architecture and architecting critical system, New York, NY, USA, pp: 3-12.
[22]. Kazman, R., Abowd, G., Bass, L., and Clements, P. 1996. Scenario-based analysis of software architecture. IEEE Software, vol. Nov, pp. 47-55.
[23]. Kazman, R., Klein, M., Barbacci, M., Lipson, H., Longstaff, T., and Carrieare, S. J. 1998. The architecture tradeoff analysis method, In: ICECCS'98: Proceedings of the Fourth International Conference on Engineering of Complex Computer Systems Monterey, CA, USA: IEEE Computer Society, pp. 68-78.
[24]. Dobrica, L., and Niemelae, E. 2002. A survey on software architecture analysis methods. IEEE Transactions on Software Engineering, vol. 28 (7), pp. 638-653.
[25]. Venters, C. C., Russell, D. J., Liu, L., Luo, Z., Webster, D. E., and Xu, J. 2009. A scenario-based architecture evaluation framework for Network Enabled Capability, In: COMPSAC'09: Proceedings of the. 33rd Annual IEEE International, pp: 9-12.
[26]. Sehestedt, S., Cheng, C-H., and Bouwers, E. 2014. Towards Quantitative Metrics for Architecture Models. First International Workshop on Software Architecture Metrics, WICSA 2014: 11th Working IEEE/IFIP Conference on Software Architecture, Sydney, Australia, April 7-11 2014.