
Proceedings Paper:
Venters, CC, Jay, C, Lau, LMS et al. (6 more authors) (2014) Software Sustainability: The Modern Tower of Babel. In: Penzenstadler, B, Mahaux, M and Salinesi, C, (eds.) CEUR Workshop Proceedings. RE4SuSy: Third International Workshop on Requirements Engineering for Sustainable Systems, 26 Aug 2014, Karlskrona, Sweden. CEUR , 7 - 12.

Position Paper: RE4SuSy: Third International Workshop on Requirements Engineering for Sustainable Systems

Software Sustainability: The Modern Tower of Babel

C. C. Venters, [2]C. Jay, [3]L. M. S, Lau, [4]M. K. Griffiths, [1]V. Holmes, [1]R. R. Ward, [5]J. Austin, [6]C. E. Dibsdale, and [3]J. Xu

[1]University of Huddersfield
School of Computing & Engineering
Huddersfield, UK
{v.holmes; r.r.ward}@hud.ac.uk

[2] University of Manchester
School of Computer Science
Manchester, UK
caroline.jay@cs.manchester.ac.uk

[3]University of Leeds
School of Computing & Informatics
Leeds, UK
{l.m.s.lau; j.xu}@leeds.ac.uk

[4]University of Sheffield
Sheffield, UK
m.griffiths@sheffield.ac.uk

[5]University of York
Department of Computer Science
York, UK
austin@cs.york.ac.uk

[6]Optimized Systems and Solutions Ltd,
Derby, UK
charlie.e.dibsdale@o-sys.com

***Abstract*— The development of sustainable software has been identified as one of the key challenges in the field of computational science and engineering. However, there is currently no agreed definition of the concept. Current definitions range from a composite, non-functional requirement to simply an emergent property. This lack of clarity leads to confusion, and potentially to ineffective and inefficient efforts to develop sustainable software systems. The aim of this paper is to explore the emerging definitions of software sustainability from the field of software engineering in order to contribute to the question, what is software sustainability? The preliminary analysis suggests that the concept of software sustainability is complex and multifaceted with any consensus towards a shared definition within the field of software engineering yet to be achieved.**

***Index Terms*— Non-functional requirements, quality attributes, software engineering, software sustainability, sustainability**

I. Introduction

The concept of sustainability has principally been associated with ecology, and the relationship between humans and planet Earth [1]. In recent years, software sustainability has emerged as an area of research in the field of software engineering and has been identified as an important future topic as new approaches to research become increasingly dependent on complex software systems, which operate in evolving, distributed e-infrastructure eco-systems [2].

Its importance has been further underlined by recent funding initiatives from the National Science Foundation [3] in the US and the Engineering and Physical Sciences Research Council [4] in the UK, combined with the establishment of the Software Sustainability Institute [5]. In addition, a number of workshops have emerged which are dedicated to exploring the topic of sustainable software and systems from a range of different perspectives [6-7].

Fundamental to the advancement of software sustainability as a field of research requires an understanding of the concept [8]. However, there is no agreed definition of software sustainability. While there have been a number of contributions to formalize a definition of software sustainability, the concept remains an elusive and ambiguous term with individual's, groups and organizations holding diametrically opposed views [9]. However, this is not a problem unique to the field of software engineering [10-11].

The narrative of the 'Tower of Babel' provides a useful analogy to describe the current understanding of the concept of software sustainability both within and outside the community. While there is no divine intervention at its source, there is a considerable amount of confusion and divergence regarding what software sustainability means, how it can be measured or demonstrated, or how to train and educate the broad spectrum of domain scientists and advance the skills of software engineers to develop sustainable software [2, 12-13]. The principal aim of this paper is to explore the definitions that have emerged from the field of software engineering in order to address the question, *what is software sustainability*? Section 2 examines the concept of software sustainability from a software artifact perspective. Section 3, examines definitions which focus on the software development process. Section 4 examines software sustainability as a non-functional requirement. Section 5 examines the use of software sustainability frameworks for exploring sustainability. Section 6 considers whether software sustainability is an emergent property. In Section 7, conclusions are drawn and future directions are outlined.

II. Software Sustainability

The word sustainability is derived from the Latin *sustinere*. The Oxford English Dictionary [14] defines sustainability as '*the quality of being sustained*', where sustained can be defined as '*capable of being endured*' and '*capable of being 'maintained*'. Endured is defined as '*continuing to exist*' and maintained as '*being supported*' [14]. This suggests that time or longevity and maintenance are important factors in understanding sustainability. The most widely adopted definition of sustainability is that proposed by the Brundtland commission which defined sustainability as '*meeting the needs of the present without compromising the ability of future generations to meet their needs*' [15]. However, this definition is rather broad and difficult to understand and apply in any meaningful way. In recent years, a triple bottom line perspective of sustainability has been adopted which considers sustainability to include three components: environment, society and economy [10]. It is argued that by incorporating the three dimensions it leads to more sustainable outcomes [16].

A number of definitions have emerged from the field of software engineering, which focuses on the sustainability of the

software artifact. Seacord et. al., [17] define software sustainability as the '*ability to modify a software system based on customer needs and deploy these modifications*'. However, they state that the terms '*software sustainment*' and '*software maintenance*' are often used interchangeably. They differentiate between the terms based on the IEEE standard definitions where software maintenance refers to '*the process of modifying a software system or component after delivery to correct faults, improve performance, or other attributes or adapt to a changed environment*' [18]. This suggests that primary difference between sustainability and maintainability is the evolution of software based on stakeholder requirements. However, they argue that there is a strong dependency on a range of other factors including the organization, stakeholders, the operational domain 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*' [5]. Despite the ambiguity of the definition, it implicitly suggests that sustainability is concerned with the qualities of availability (*available*), extensibility (*improved*), and the maintainability (*supported*) of the software where the attributes can be aligned in accord with the IEEE definitions [18].

Koziolek [20] defines the term sustainability in the context of software architectures and proposes two definitions of sustainable software. In the first definition sustainable software is defined as '*a software-intensive system that operates for more than 15 years*'. This is a position supported by Tamai and Torimitsu [21] who suggest that the average software lifetime is 10 years, with a minimum of two years and a maximum of thirty. In the second definition, sustainable software is defined as '*a long-living software system which can be cost-efficiently maintained and evolved over its entire life-cycle*'. While the former definition offers no real insight into software sustainability it provides a benchmark against which to investigate formal methods for assessing software longevity. Similarly, the latter definition suggests that maintainability and extensibility are key features of sustainability, which are tightly coupled with the economical dimension in determining the sustainability of software.

Penzenstadler [22] defines sustainability as '*preserving the function of a system over a defined time span*'. This implicitly suggests that software sustainability is primarily concerned with maintainability with the addition of time as a factor. However, does preservation in the strictest sense of the word lead to or ensure sustainability? In addition, it is argued that within this definition there are three core variables that need to be defined for setting the scope for discussions on sustainability:

- System: the humanity in its ecosystem;
- Function: a satisfaction of need;
- Time: spans various generations.

Based on this a distinction is drawn between an *absolute versus* a *relative definition* of software sustainability where the difference is that the former has fixed variables while the later requires that the variables are chosen based on the context. In addition, four aspects are proposed which can be used for serving as a structure for discussing and supporting software sustainability rather than as an '*apodictic differentiation*':

- Development process: use of ecological, human and financial resources;
- Maintenance process: continuous monitoring of quality and knowledge management;
- System production: focused on the use of resources for production to be achieved;
- System usage: takes into account responsibility for the environmental impact.

The principal distinction here is that the first two aspects focus on the organization and its processes, and the latter two focuses on the system being developed. As a result, it is suggested that a distinction can be made between '*software for sustainability*' which is related to the absolute definition and '*sustainable software*' which is related to the relative definition.

III. Sustainable Software Development

An increasing number of definitions of *sustainable software* focus on sustainability from a *software development* perspective. Amsel et. al., [23] consider sustainable software engineering as a process which '*aims to create reliable, long-lasting software that meets the needs of users while reducing environmental impacts*'. Without specific reference to a definition of reliability, this suggests that there is a link between reliability, stakeholder requirements and the environment. In order to explore the environmental impact of software usage they developed GreenTracker, which measures the energy consumption of software CPU consumption. This is related to the concept of reliability, which plays a key role in determining the cost-effectiveness of systems. The data generated by the tool can then be analyzed to create more energy efficient software.

Naumann et. al., [24] make a distinction between sustainable software and sustainable development. In the first definition, sustainable software is defined as '*software, whose direct and indirect negative impacts on economy, society, human beings, and environment that result from development, deployment, and usage of the software are minimal and/or which has a positive effect on sustainable development*'. This relates the triple bottom line perspective of sustainability to the software development lifecycle. However, they suggest that a sustainable software product can only be achieved if the organization is aware of the negative and positive impacts on sustainable development that results from usage. In the second definition sustainable software development is defined as '*the art of developing sustainable software with a sustainable software engineering process so that negative and positive impacts result in and/or are expected to result from the software product over its whole life cycle are continuously assessed, documented, and used for further optimization of the software product*'. To achieve sustainable software through a sustainable software development process they proposed the GREENSOFT model; a conceptual reference model, which includes a cradle-to-grave product life cycle model for software

products, sustainability metrics and criteria for software, extensions for software engineering.

Calero, Moraga, and Bertoa [8] attempt to make a similar distinction between '*sustainable software development*' and the '*sustainability of a software product*'. Sustainable software development is defined 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*'. In contrast, sustainability of a software product is defined as '*the capacity of developing a software product in a sustainable manner*'. However, it is not clear what the real distinction is between the two definitions since sustainable software development should in essence lead to a sustainable software product.

Tate [25] argues that developing software is a complex task that is performed in an environment of constant change and uncertainty, which results in software products that are unsustainable. He defines sustainability as '*developing the capability to deliver customer value today and tomorrow*'. However, it is also linked to *agility* and *context* where agility is concerned with the balance between the short term versus long term, anticipation versus adaptation, ceremony versus informality, and context is the specific context of each project, which must be understood in order to adapt development practices. He proposes that the solution to this problem is sustainable development; a mindset and culture which can be accompanied by a set of practices that include continual refinement of the product and project practices; a working product at all times; continual investment in and emphasis on design; and valuing defect prevention over defect detection. However, Fenner et. al., [26] argue that for sustainable engineering to be successful it requires a paradigm shift in thinking to embrace a holistic approach founded in complexity science.

IV. Software Sustainability: Non-Functional Requirement?

A number of commentators argue that sustainability should be classified as a first-class, non-functional requirement [23, 27-28]. In the field of 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*' [18]. Non-functional requirements express desired qualities of the system to be developed and refer to both observable qualities and also to internal characteristics. In addition, they specify criteria that can be used to judge the operation of a system, rather than its specific functional behavior. As a result, a number of contributions have focused on defining software sustainability as a non-functional requirement.

Without explicit reference to specific non-functional requirements, the GREENSOFT model proposed by Naumann, Dick, Kern and Johann [24] is designed to incorporate a range of non-functional requirements within the three categories of the sustainability criteria and metrics section of the reference model. This separation allows the examination of first-, second- and third- order impacts on the environment that result from effects of supply, effects of usage and systemic effects. However, they suggest that the fundamental question at the heart of the model is not, in which phase are metrics applied or in which phases are they taken in order to improve the quality attributes? The principal question is, *in which life cycle phase can the related effects be observed*?

Taina [29] argues that from the software system perspective, software is an indivisible component. As a result, sustainability can be defined only within a software system. This suggests that sustainability is a relative factor as two software artifacts cannot be compared unless they are in similar software systems. He suggests that sustainable software has the following properties:

- Fit for purpose: defines how software helps its system reach its goal;
- Reduction: defines how software supports its system in waste reduction;
- Beauty: defines the value of the system in sustainable development.

It is suggested that all these factors can be measured similarly inside a software system. An important caveat is that the problem domain should be defined where the software is executed prior to the definition of sustainability being defined. This strongly suggests that software sustainability is highly context dependent and a relative concept.

Venters et. al., [13] defined software sustainability as a composite, non-functional requirement which is *'a measure of a systems extensibility, interoperability, maintainability, portability, reusability, scalability, and usability'* where the attributes can be 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.

Several of the quality attributes specify the '*effort required*' to achieve a particular outcome. This suggests that the concept of sustainability is strongly coupled to other quality attributes such as energy and cost efficiency, and resource utilization over its entire lifetime. They argue that by defining software sustainability as a non-functional requirement it allows us to move away from the focus of thinking about how we sustain existing software, to understanding how we can develop sustainable software in the future. This is a position supported

by Koziolek [20] who proposes that sustainability comprises the core attributes of *maintainability, modifiability, portability and evolvability*. However, it is clear that such a definition needs to embrace other dimensions of sustainability.

Defining software sustainability as a non-functional requirement is a position supported by Calero, Bertoa, and Moraga [8] who suggest that software sustainability is related to a number of the main quality attributes and their sub-characteristics defined in the ISO/IEC 25010 quality model; the standard has eight product quality characteristics and thirty one sub-characteristics. However, they suggest that sustainability can be considered from two perspectives: *energy efficiency* and *perdurability*. In terms of energy efficiency they propose the sub-characteristics of:

- Energy consumption: the degree to which the amount of energy used by a software product when performing its functions meet requirements;
- Resource optimization: the degree to which the amount and types of resources used by a product when performing its functions meets sustainability requirements.

Based on the sub-characteristics of reusability, modifiability, and adaptability they define '*perdurability*' as the '*degree to which a software product can be modified, adapted and reused in order to perform specified functions under specified conditions for a long period of time*'. However, this significantly narrows the view of software sustainability as a non-functional requirement and potentially eliminates important software quality attributes. Similarly, it is not clear why defining software sustainability in terms of its perdurability i.e. very durable, is different from the overall aim of making software sustainable, at least in terms of the artifact, as the definition of sustainability is underpinned by the idea of enduring.

One of the principal challenges in defining sustainability as a non-functional requirement is how to demonstrate that the quality factors have been addressed in a quantifiable way. Software architectures provide a potential mechanism for reasoning about software sustainability at an architectural level, which is achieved through adherence to design principles such as modularity, separation of concerns and conceptual integrity throughout the entire life cycle [13, 20, 30].

V. Software Sustainability Frameworks

A number of frameworks have also been proposed for exploring sustainability. Cabot et. al., [31] focus on natural sustainability which they define as '*the exploitation of an (eco)system that does not degrade or adversely change the system beyond what is acceptable*'. To address sustainability they propose the *i* framework* as a sustainability taxonomy for modeling and integrating stakeholders' sustainability issues. This can be used for exploring alternative design options during the development of a software system where decisions may have a potential impact on sustainability. However, the extent to which this approach can be utilized beyond the case study used to develop the taxonomy is unclear but provides a useful basis to explore its limits and generalizability.

Jansen, Wall and Weiss [32] focus on sustainability from an economic perspective and consider how a system can remain economically viable over its entire lifetime. To address this they propose TechSuRe as a method for reasoning about sustainability in assessing software evolution and technology integration from three perspectives: time, risk and cost benefit. Sustainability is defined in terms of '*sustainability risk*' which is an estimated value based on nine high-level indicators: lifetime in production, lifetime, competence risk, technology evolution risk, risk of changing business model, market risk, lifetime certainty, complexity risk and technology evolution fitness. The output of the assessment is an indication of the expected lifetime of the technology's [economic] sustainability.

Penzenstadler and Femmer [33] propose a reference model for sustainability that decomposes sustainability into five dimensions:

- Environmental: improving human welfare by protecting natural resources;
- Individual: the maintenance of the private good of individual human capital;
- Social: maintaining social capital and preserving the societal communities in their solidarity;
- Economic: maintaining assets;
- Technical: long-time usage of systems and their adequate evolution with changing surrounding conditions and respective requirements.

The method comprises a generic sustainability reference model, a meta-model, and instances derived for specific processes and software systems, and is primarily designed to aid as a reference model for software developers. Their approach demonstrates how environmental sustainability can be aligned with the other dimensions of sustainability. How the reference model potentially integrates with non-functional requirements presents an opportunity to explore its robustness.

Rodriguez and Penzenstadler [34] propose utilizing the IMAGINE approach developed by Bell and Morse [35] which applies the principles of systems thinking for analyzing and assessing sustainability of a software-intensive system that exhibits a significant impact on the sustainability of city mobility. The overall aim of the research was to investigate the applicability and usefulness of the approach from within classical sustainability research to requirements engineering for software-intensive systems. Rather than adopt an absolute definition of sustainability they utilize a set of sustainability indicators previously defined by Rodriguez and Penzenstadler [36]. While a formal evaluation was not possible, the results suggest that the IMAGINE approach could be successfully applied with a focus on sustainability at its roots. Similarly, the existing sustainability indicators can be utilized in other assessments in related application areas.

In addition, a number of frameworks have been proposed for defining sustainability without specific reference to the field of software engineering [37-39]. However, it is suggested that the primary challenge of developing frameworks for sustainability requires defining the boundaries to include the context within which the problem domain is situated [39].

VI. Software Sustainability: Emergent Property

A diametrically opposed position of an absolute definition of software sustainability is that it is simply an emergent property of a software system. An emergent property cannot be attributed to any specific part of the system but emerges once the components of the system have been integrated into a whole [40]. Two types of emergent properties can be identified:

- Functional emergent properties: the purpose of the system emerges after its components are integrated;
- Non-functional emergent properties: related to the behavior of the system in its operational environment.

This suggests that sustainability cannot be designed or engineered and quantified until after the software system is operational. This issue was explored at WSSSPE'1 [7] where the following question was raised, *can sustainability be designed for or is it an emergent property that a market determines* [41]? Using MS Word as an example, it was suggested that it had sustained i.e. endured. As a result, this product could be described as sustainable software since by a dictionary definition it had endured. However, it was argued that its sustainment was driven by demand not by any software engineering quality. Nevertheless, since its release in 1983 it would be difficult to argue that its sustainment could be solely attributed to market forces since the software has been maintained, evolved, matured and ported on to different platforms.

VII. Summary and Conclusions

The aim of this paper was to explore the emerging definitions of the concept of software sustainability from the field of software engineering in order to contribute to the ongoing discussion of *what is software sustainability*?

This preliminary analysis suggests that software sustainability is a complex and multifaceted concept that can be viewed from a variety of perspectives and can include a range of different dimensions and factors. Despite the numerous definitions that exist for software sustainability most are either too vague or limited in their scope. Such definitions prove inadequate because of their reliance on abstract constructs that provide limited guidance in developing quantitative indicators for measuring performance or representing the complexities involved. As a result the term software sustainability is frequently used to embody vague, diverse and contradictory ideas that are neither sound nor novel. This lack of a shared definition can lead to incompatible practices. The quote regarding Big Data and teenage sex could aptly be applied to software sustainability:

> "Everyone is talking about it, nobody really knows how to do it, everyone thinks everyone else is doing it." Dan Ariely

While not everyone is talking about it the rise in research output would suggest that there is a growing interest in software sustainability as an active area of research. However, the significance of not having a shared and common definition of software sustainability cannot be underestimated. Without a clear and commonly accepted definition of what software sustainability means, contributions will continue to remain insular and isolated, which will ultimately lead to ineffective and inefficient efforts to address the concept or result in its complete omission from the software system [22]. Any consensus within the field of software engineering has yet to be achieved.

What is required is a definition that is tailored to quantitative sustainability objectives that encompasses its complexity and multi-dimensional nature. This would result in a clear indication as to whether objectives of software sustainability have been met; provide a holistic view of the ecosystem in which it operates; have a quantitative character; contain parameters whose relevance will be slow to degrade. Importantly, any definition of software sustainability must be understandable not only to domain scientists and software engineers but to the layperson. In the interests of avoiding future inconsistencies it is necessary to develop an integrative framework that allows the characterization and classification of existing definitions and perspectives and their relevance to software sustainability that meet the above criteria. Future work will focus on the development of a framework that aims to disambiguate the term software sustainability and express it in terms of quantifiable metrics rather than conceptual constructs. How to make software sustainable both in terms of the software artifact, the development process, and how these relate to the wider concerns of environmental, economic, social, individual, and technical sustainability remains an open area of research.

References

[1] A. Woodruff and J. Mankoff. "Environmental sustainability," IEEE Pervasive Computing, 8(1), pp: 18-21, 2009.

[2] A. Geist, and R. Lucas. "Major computer science challenges at Exascale." International Journal of High Performance Computing Applications, 23(4): pp: 427-436, 2009.

[3] NSF A Vision and Strategy for Software for Science, Engineering, and Education. Available: http://www.nsf.gov/publications/pub_summ.jsp?ods_key=nsf12113

[4] EPSRC Software for the future. Available: http://www.epsrc.ac.uk/funding/calls/2014/Pages/softwarefuture.aspx

[5] Software Sustainability Institute. Available: http://www.software.ac.uk/about.

[6] Re4Susy: Third International Workshop on Requirements Engineering for Sustainable Systems. Available: http://www.ics.uci.edu/~bpenzens/2014re4susy/

[7] WSSSPE'1: First Workshop on Sustainable Software for Science: Practice and Experiences. Available: http://wssspe.researchcomputing.org.uk/wssspe1/

[8] C. Calero, M. A. Moraga, and M. F. Bertoa. "Towards a software product sustainability model," WSSSPE'1: First workshop on sustainable software for science: practice and experiences, SC'13, 17 November 2013, Denver, CO, USA.

[9] D. S. Katz et. al., "Summary of the first workshop on sustainable software for science: Practice and experiences (WSSSPE1)," Journal of Open Research Software, 2 (1):e6, pp: 1-21, 2014.

[10] H. Svendrup and M. G. E. Svenson. "Defining the concept of sustainability – a matter of systems thinking and applied systems

analysis," Systems Approaches and their Application, pp: 143-164, 2004.

[11] P. Glavič and R. Lukman. "Review of sustainability terms and their definitions," Journal of Cleaner Production, 15(18), pp: 1875–1885, 2007.

[12] B. Penzenstadler, and A. Fleischmann. "Teach sustainability in software engineering?" Proceedings of the 24th IEEE-CS Conference on Software Engineering Education and Training, IEEE Computer Society, pp: 454-458, 2011.

[13] C. C. Venters et. al., "The blind men and the elephant: Towards an empirical evaluation framework for software sustainability," WSSSPE'1: workshop on sustainable software for science: practice and experiences, SC'13, 17 November 2013, Denver, CO, USA.

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

[15] United Nations: World Commission on Environment and Development: Our Common Future. Oxford Univ. Press, 1987

[16] J. Elkington. "Enter the triple bottom line," In: R. Henriques and A. Richardson (eds.). The triple bottom line: Does it all add up? pp: 1-16, 2004.

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

[18] IEEE. 1990. "IEEE standard glossary of software engineering terminology", IEEE Std. 610.12-1990.

[19] Software Sustainability Institute. Available: http://www.software.ac.uk/about

[20] H. Koziolek. "Sustainability evaluation of software architectures: A systematic review," Proceedings of the joint ACM SIGSOFT conference on quality of software architectures. Boulder, Colorado, USA, pp: 3-12, 2011.

[21] T. Tamai and Y. Torimitsu. "Software lifetime and its evolution process over generations," Proceedings of the Conference on Software Maintenance, pp: 63-69, 1992.

[22] B. Penzenstadler. "Towards a definition of sustainability in and for software engineering," SAC'13: Proceedings of the 28th Annual ACM Symposium on Applied Computing, Coimbra, Portugal, March 18-22, pp: 1183-1185, 2013.

[23] N. Amsel, Z. Ibrahim, A. Malik, B. Tomlinson. "Toward sustainable software engineering," ICSE: Proceedings of the 33rd International Conference on Software Engineering. Waikiki, Honolulu, HI, USA, pp: 976-979, 2011.

[24] S. Naumann, M. Dick, E. Kern, T. Johann. "The GREENSOFT model: A reference model for green and sustainable software and its engineering," Sustainable Computing: Informatics and Systems, 1, pp: 294–304, 2011.

[25] K. Tate. "Sustainable software development: An agile perspective", Addison-Wesley, 2006.

[26] R. A. Fenner, C. M. Ainger, H. J. Cruickshank, P. M. Guthrie. "Widening engineering horizons: Addressing the complexity of sustainable development," Proceedings of the ICE: Engineering Sustainability, 159 (4), pp: 145-154. 2006.

[27] M. Mahaux, P. Heymans and G. Saval. "Requirements engineering: Foundation for software quality," Lecture Notes in Computer Science Volume 6606, pp: 19-33, 2011.

[28] T. Johann and W. Maalej. "The social dimension of sustainability in requirements engineering," RE4SuSy: Second International Workshop on Requirements Engineering for Sustainable Systems Monday, Rio, Brazil, July 15th, 2013.

[29] J. Taina. "Good, bad, and beautiful software - In search of green software quality factors," CEPIS UPGRADE, volume XII, pp. 22–27, 2011.

[30] S. Sehestedt, C-H. Cheng, and E. Bouwers. "Towards quantitative metrics for architecture models," Proceedings of the WICSA 2014 Companion Volume. Sydney, Australia, ACM: pp: 1-4, 2014.

[31] J. Cabot, S. Easterbrook, J. Horkoff, L. Lessard, S. Liaskos, J. Mazón. "Integrating sustainability in decision-making processes: A modelling strategy," ICSE: 31st International Conference on Software Engineering. pp: 207-210, 2009.

[32] A. Jansen, A. Wall and R. Weiss. "TechSuRe: A method for assessing technology sustainability in long lived software intensive systems," SEAA 2011: Proceedings of the 37th EUROMICRO Conference on software engineering and advanced applications, Oulu, Finland, August 30 - September 2, 2011.

[33] B. Penzenstadler and H. Femmer. "A generic model for sustainability with process- and product-specific instances," In International Workshop on Green In Software Engineering and Green By Software Engineering at AOSD, 2013.

[34] A. Rodriguez and B. Penzenstadler. "An assessment technique for sustainability: Applying the imagine approach to software systems," RE4SuSy: Second International Workshop on Requirements Engineering for Sustainable Systems Monday, Rio, Brazil, July 15th, 2013 at RE'13, July 15th-19th, 2013.

[35] S. Bell and S. Morse. "Sustainability indicators: Measuring the immeasurable?" Earthscan Publications, 1999.

[36] A. Rodriguez and B. Penzenstadler. "An assessment technique for sustainability: Applying the IMAGINE approach to software systems," Technical Report TUM-I1218, Technische Universitat Munchen, 2012.

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

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

[39] A. Mukherjee and H. Muga. "An integrative framework for studying sustainable practices and its adoption in the AEC industry: A case study," Journal of Engineering and Technology Management, 27, pp: 197-241, 2010.

[40] P. Checkland, "Systems thinking, systems practice: Includes a 30 year retrospective," John Wiley & Sons, 1981.

[41] "WSSSPE Collaborative notes," Available bit.ly/wssspe13