JISBD 2017 (La Laguna)

Ruiz, F. (Ed.), Actas de las XXII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2017). La Laguna (Tenerife), septiembre de 2017.

Las XX Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2015) se han celebrado en La Laguna del 19 al 21 de julio de 2017, como parte de las Jornadas SISTEDES.
El programa de JISBD 2017 se ha organizado en torno a sesiones temáticas o *tracks*. A continuación se detalla el contenido de las actas:

- Track RCP – Requisitos, Calidad y Pruebas

Ecological Debt: outlining a measure to evaluate software *greenability*

Ignacio García-Rodríguez de Guzmán, Félix O. García, Mª Ángeles Moraga, Mario Piattini

Instituto de Tecnologías y Sistemas de la Información, University of Castilla-La Mancha, Ciudad Real, Spain
ignacio.grodriguez, felix.garcia, mariaangeles.moraga, mario.piattini@uclm.es

Abstract— Developing low quality software (with design flaws, poor quality code, etc) lead to a product with an inner cost that could be measured by using technical debt, that could be considered as the economical effort to solve all the existing design problems of a given software. As time goes on, software quality is acquiring new dimensions, and one of the most important one in the recent years (required by our society) is Software Sustainability, that could be understood as the degree of environmental-friendliness of a software system. So, following the idea of technical debt, we propose the concept of *Ecological Debt* which purpose is to measure the economical effort to develop a sustainable software following the *Green-in* principles.

1 Introduction

Technical debt can be defined as *"writing immature or not quite right code in order to ship a new product to market faster"*; we can find it in many forms (process, scope, testing and design) [1]. Technical debt could also be understood as the *"invisible result of past decisions about software that affect the future*" [2]. Eq.1 presents a very simple equation that summarizes how technical debt could be calculated. In this formula, the "*technological flaw"* concept represents any kind of bad smell, anti-pattern or lack (of documentation or of test cases, for example).

$$Technical\_Debt = \Sigma Refactor(Technological\_Flaw_j) \text{ [Eq.1]}$$

Technical debt may be considered to be a result of decisions to trade-off competing concerns during development; but the problem is that these decisions are the result of short-term thinking due to many reasons [1].

On one hand, the existence of technical debt is not a good indicator; but on the other hand assuming a certain degree gives us the possibility of releasing a product to take full advantage of a market opportunity, test a first version of a product with a stakeholder who has no clear idea of the requirements, or put off implementing requirements that are not essential for a first version of the system.

Technical debt is a "measure" related, in turn, with software development and maintenance. However, in the last years there is an interest, necessity and efforts on creating green IT solutions [3] in order to save IT's increasingly energy demands due to several factors [4]. This new perspective when developing green IT solutions, and

adfa, p. 1, 2011.
© Springer-Verlag Berlin Heidelberg 2011

particularly, green IT software solutions lead us to a new question: *which is the sustainability cost of not adopting sustainable practices in software development?*

Following the metaphor of technical debt where, any kind of software flaw or software design is compromising a value in the future, we propose the concept of *ecological debt* as measures (which we firstly outline in [5]) to evaluate the costs of not adopting good practices in developing sustainable software following the *Green-in* practices.

2 Software Sustainability and Green in Software Engineering

In broad terms, software sustainability consists on several practices that could be undertaken from several dimensions in order to follow an environmental friendly software development process to produce environmental friendly software product. However, the *Green in* software engineering could be considered as practices which apply engineering principles to software by taking into consideration environmental aspects. The development, the operation and maintenance of software are therefore carried out in a green manner and produce a green software product, process or service.

3 Ecological Debt: Hidden costs of non-sustainable software

3.1 Outlining the concept

While technical debt is in fact the lack of required functional or nonfunctional requirements (on purpose or unintentionally), it is possible to outline a similar situation with respect to green software development: the concept of *ecological debt*. ***Greenability*** requirements (as nonfunctional requirement) would not be an absolute value (for example, the response time for queries), but would be allowed to move within a given range. Obviously, the greener the system is, the less its consumption of resource is. But once again, the trade-off issues should be analyzed. If the long term cost of increasing the greenness in a software system is smaller than making a system highly sustainable (is less than planned but with a given and acceptable degree of greenability), then it may be reasonable to include a certain value of *ecological debt* in the system. It is important to track this debt, since some legal stipulation or stakeholder requirement might change. If that should happen, the systems would have to be refactored, with the subsequent waste of resources (extra costs). When *ecological debt* is incurred, it is very important to consider the feasibility of reengineering (and in turn, refactoring) source code, since this is considered a very time and resource-consuming task.

Nevertheless, not all *ecological debt* is due to decisions taken because of greenability requirements (or failing to complete requirements). These include data transfer, processing and hardware infrastructure, all of which is required for the delivery of updates. All these issues, which may cause further consumption of power and resource [6], imply a decrease in software greenability. If we submit our systems to such policies or strategies we are assuming an *ecological debt* that must be recognized and quantified.

3.2 Providing a first definition

Now that the concept is clear, a possible definition can be outlined. *Ecological debt* may be considered as "*the cost (in terms of resource usage) of delivering a software system with a greenability degree under the level of the non-functional requirements established by stakeholders, plus the incurring cost required to refactor the system in the future*" (Eq.2). Such proposed definition differs from other similar ones which consider all the sustainability dimensions [7], but it is important to consider that *ecological debt* is a measure oriented to *Green-in* vision of software sustainability.

$$Ecological\_Debt = \Sigma Cost(resource_i) + \Sigma Refactor(Ecological\_Flaw_j) \text{ [Eq.2]}$$

According to Eq.1 and Eq.2, ecological and technological debts have a common factor; this is the need to fix flaws. Until now, it has not been clear whether there are specific flaws that are associated with low values of software greenability, or even if the existing ones (applied in classical software maintenance) also influence greenability. As proposed in the previous section, a possible starting point for classifying which flaws affect greenability would be the examination of those flaws related to classical maintenance, validating their impact on software greenability.

Eq. 2 points to a very important consideration when talking about ecological debt; this is the fixed (and unrecoverable) cost of the overused resources. An overused resource is seen as a software or hardware resource which has been oversized for the actual software need. For example, a very common oversized resource related to greenability is power consumption. Power consumption can be expressed as cost ($, €, £, etc.), carbon footprint (CO_2) or electrical power (w/h). The cost of these resources is a sort of investment that cannot be recovered or repaired (as can be done, on the other hand, with technical debt). It is possible to match such wasted power consumption to an economic concept- the irrecoverable expense: it is an outlay that cannot be recovered, and thus, must not influence the future decisions of the organization, since it cannot be recovered. It is nevertheless important to point out that in ecological debt, this irrecoverable expense must be taken into account in the context of the maintenance of the software maintenance strategy for the software portfolio. Technical debt exists for the whole period during which the organization is not decide to solve it, but the irrecoverable expense factor of the ecological debt (Eq.2) is a continuous expense that will never be recovered. This is the most important reason for reducing the ecological debt.

3.3 Weaving economical- and technical-debt

Both forms of debt-the ecological and the technological- have in common the need to fix flaws. We can in fact state that a software system has a set of flaws which are responsible for its debt and that set is made up of the ecological and technical flaws. Now, the issue is: what happens if a technical flaw affects greenability, or vice versa?

For any given system, if there were not a set of (technical and ecological) flaws that clearly affects both the system maintainability and greenability positively and in the same way, it would be necessary to undertake a trade-off analysis, since refactoring technological flaws would negatively affect ecological debt, and vice versa. Studies such as [8] reveals that removing certain design flaws increases power consumption.

On the other hand, if there were common flaws, it would be possible to apply a set of refactoring transformations that benefits both debt and technological debt (while in turn improving greenability and maintainability). Finally, (though very improbably), if ecological and technical flaws were the same, then refactoring all of them would reduce both kinds of debt and improve greenability and maintainability.

4. Conclusions

In this paper, we present the concept of *ecological debt*, which aims to measure the cost of not considering good sustainable practices or taking bad design decisions in software development, producing software products with a low greenability (as a quality dimension) level. Ecological debt follows the same philosophy than technical debt, since the second one considers the cost of solving design flaws in software systems.

In further steps, authors will reinforce the definition of ecological debt to be able to determine which factors directly affect to such measure. In addition, trade off analysis should be carried out in order to identify in what extent software sustainable good practices affects (negatively) to software design good practices, and vice versa.

Acknowledgements

This work has been partially supported by the projects GINSENG (TIN2015-70259) and SEQUOIA (TIN2015-63502-C3-1-R), founded by the Ministerio de Economía y Competitividad y Fondo Europeo de Desarrollo Regional FEDER.

References

1. Lim, E., Taksande, N., Seaman, C.: A Balancing Act: What Software Practitioners Have to Say about Technical Debt. IEEE Softw. 29, 22–27 (2012)
2. Kruchten, P., Nord, R.L., Ozkaya, I., Falessi, D.: Technical debt: towards a crisper definition report on the 4th international workshop on managing technical debt. Softw. Eng. Notes. 38, 51–54 (2013)
3. Murugesan, S.: Harnessing Green IT: Principles and Practices. IT Prof. 10, 24–33 (2008)
4. Murugesan, S., Gangadharan, G.R., Harmon, R.R., Godbole, N.: Fostering Green IT - Guest Editors' Introduction. IT Prof. 15, 16–18 (2013)
5. Calero, C., Piattini, M. eds: Green in Software Engineering. Springer International Publishing, Cham (2015)
6. Naumann, S., Dick, M., Kern, E., Johann, T.: The GREENSOFT Model: A reference model for green and sustainable software and its engineering. Sustain. Comput. Inform. Syst. 1, 294–304 (2011). doi:http://dx.doi.org/10.1016/j.suscom.2011.06.004
7. Betz, S., Becker, C., Chitchyan, R., Duboc, L., Easterbrook, S.M., Penzenstadler, B., Seyff, N., Venters, C.C.: Sustainability Debt: a Metaphor to Support Sustainability Design Decisions Original. In: 4th International Workshop on Requirements Engineering for Sustainable Systems (2015)
8. Perez-Castillo, R., Piattini, M.: Analyzing the Harmful Effect of God Class Refactoring on Power Consumption. IEEE Softw. 31, 48–54 (2014). doi:http://doi.ieeecomputersociety.org/10.1109/MS.2014.23