Exploring Initial Challenges for Green Software Engineering

[Summary of the First GREENS workshop, at ICSE 2012]

Patricia Lago
VU University Amsterdam
The Netherlands
p.lago@vu.nl

Rick Kazman
University of Hawaii
USA
kazman@hawaii.edu

Niklaus Meyer
Green IT SIG
Swiss Informatics Society,
Switzerland
niklaus.meyer@acm.org

Maurizio Morisio
Politecnico di Torino, Italy
maurizio.morisio@polito.it

Hausi A. Müller
University of Victoria, Canada
hausi@cs.uvic.ca

Frances Paulisch
Siemens, Germany
frances.paulisch@siemens.com

ABSTRACT
The GREENS workshop provides a forum for practitioners and academics to share knowledge, ideas, practices and current results related to green and sustainable software engineering. This first workshop was held at ICSE 2012 in Zurich, Switzerland. It featured a keynote talk, twelve research position statements and two breakout sessions that discussed topics that ranged from bringing sustainability and energy efficiency into all software lifecycle stages, to green measures and estimations, practices, notations, and tools to both greening the software engineering process, and greening the resulting Information and Communication Technology systems. This report presents the themes of the workshop, summarizes the results of the discussions held in the breakout sessions, as well as the identified research challenges.

Categories and Subject Descriptors
D.2 [**Software Engineering**]: Miscellaneous; D.2.1 [**Software Engineering**]: Requirements; D.2.8 [**Software Engineering**]: Metrics—*complexity, product, process*; D.2.10 [**Software Engineering**]: Design

Keywords
Energy efficiency, sustainability, software engineering, green IT

1. INTRODUCTION
Information and Communication Technology (ICT) accounts for approximately 2% of world CO2 emissions [2], a figure equivalent to aviation, according to Gartner estimates [4]. In fact, this 2% includes only the *in-use phase* of hardware: in the remaining 98% software both operationalizes the private sector in doing its business and the public sector in supporting society, as well as delivering end-user applications that permeate our personal lives. Software can contribute to decrease power consumption (i.e. become greener) in at least two ways. First, by being more energy efficient, hence using less resources and causing fewer CO2 emissions. Second, by making its supported processes more sustainable, i.e. decreasing the emissions of governments, companies and individuals. To this end, enterprise software must be completely rethought to address sustainability issues and support sustainable & innovative business models and processes.

Creating greener software has been the focus of GREENS 2012 [1] with the special theme "Green Knowledge for Sustainable Software Engineering." GREENS aims at bringing together software engineering researchers and practitioners to discuss the state-of-the-art and state-of-the-practice in green software, as well as research challenges, novel ideas, methods, experiences, and tools to support the engineering of sustainable and energy efficient software systems.

2. THE KEYNOTE TALK
The keynote was given by Pia Stoll (principal scientist at ABB Corporate Research, Sweden) on Sustainable development of a Residential Demand Response System [3]. This inspiring keynote stressed the need for reliable *sustainability indicators* required to assess the level of sustainability in system development. To that end, a sustainable development framework has been explained (yielding four dimensions, these being Scope, Business, Systems and Technology). Pia illustrated with one large case study (the Stockholm Royal Seaport) the relation between sustainable development (defined as the development that meets the needs of the present without compromising the ability of future generations to meet the needs of the future) and two key concepts: the concept of *needs*, and in particular the essential needs of the world's poor, to which overriding priority should be given; and the idea of *limitations* imposed by the state of technology and social organization on the environment's ability to meet present and future needs.

3. BREAKOUT SESSIONS
The workshop accepted twelve papers[1] for inclusion in the proceedings and two posters. The papers can be divided into two distinct categories: those focusing on **energy efficiency** (EE) and discussing how EE of software can be estimated, reported upon, and monitored; and how software engineering can be blended with **green thinking** at various levels of abstraction (from design patterns to architecting and economic estimations). The authors of accepted papers were invited to present their ideas to the workshop in the form of a position statement. The topics identified as most interesting by attendees have been: (1) Best practices for sustainable and energy efficient software (engineering); (2) Green software measures, including metrics and Key Process Indicators (KPIs); and (3) Green software life cycle. The latter two topics have been selected for further discussion in the breakout sessions, and the following describes the discussion outcomes.

3.1 Green Software Measuress
The starting point of discussion in the session is to consider measures for Green IT as a non-functional property of software (or of systems including software).

The traditional list of non-functional properties is (using standard

[1]Papers accepted for GREENS 2012 are available from the ACM Digital Library.

ISO 9126 and later ISO 25010) efficiency, compatibility, usability, reliability, security, maintainability and portability. The standard does not explicitly mention green properties. Any property related to greenness could be either added as a decomposition of efficiency, or added as a new high level property. Influenced by the opening keynote, the breakout session reaches consensus on proposing a new high level, non-functional property of software systems, called *sustainability*. Among others, sustainability includes resource consumption (such as energy), greenhouse gas emissions, social sustainability, and recycling.

The discussion further focused on resource- and energy consumption, in the context of software systems per se. In other words software for greening (i.e. any software that controls, directly or indirectly systems consuming energy, such as cars, buildings, factories) is left out of the discussion. In this context the session identifies two categories of measures: basic, and derived. The basic measures are energy [Joule], power [Watt], gas emissions, cost. All these measures are already established. Derived measures are, at high level, efficiency and productivity. *Productivity*, in this context can be defined as *useful work per Joule*. This is still a high level measure, since it requires defining useful work. The session pushes the community in defining useful work in different contexts. For instance the business community has proposed *business transaction per Joule*, the algorithm community has proposed *sorted records per Joule*. Another derived measure is *greenhouse gas emission per useful work*. The session promotes, also at this regard, work to detail further this basic measure. The session agrees that the interplay of software design and hardware design, and their effect on derived measures should be studied further. Besides, the interplay of hardware and software lifetimes should be analyzed too. In many cases new, more advanced software programs require new hardware, and similarly new hardware requires new software.

The session proceeds in discussing *methodological issues* on measuring sustainability. The session pushes the community to agree on procedures to achieve controllability and repeatability of measures in this context. It is clear that measuring all combinations of hardware and software is unfeasible. Accordingly, a key problem is the definition of standard (or at least meaningful and repeatable) configurations of hardware and software to be measured.

Linked problems are the definition of:

- Standard layers and interfaces (a call to a kernel mode function from an application and the related consumption should be allocated to the application or to the OS?)
- Standardized ways of accessing energy measures from hardware
- Standardized layers on hardware components and drivers (disk, memory, wi fi card etc)
- Standardized usage scenarios, until the definition of meaningful benchmark scenarios. The starting point can be approaches like TPCEnergy, JouleSort, SpecPower, SpecJVM. However, they are not specific to sustainability, use often assumptions on the context that should be verified, and usually consider server-side software only.
- Procedures to support repeatability and controllability
- What hardware/software configurations (“standard”)
  - Standardized way to access energy measures from hardware (e.g. virtual machine)
  - Limits of application (call to kernel mode function is application or OS?)
  - Devices / components (besides CPU)
- Standardized scenarios and benchmarks: there exist already a significant number of software packages that estimate the power consumption of computers. Examples include TPC-Energy, JouleSort, SpecPower, SpecJVM. Most of them, however, rely on gross approximations and assume contextual configurations that are embedded, implicit and not specific to energy consumption. While these software packages are a step forward toward estimating the level of greenness of software, they do not yet allow to univocally link a certain software (functionality) to its accountable measure. The next generation benchmarks need to be configurable in both context and observed usage/execution scenarios. They also need to clearly identify which software component is responsible for which resource allocation and hence which rate of the total energy consumption.
- Absolute measures, what they mean? Better to use relative (and reliable) measures.

NOTE: The above can change hugely depending on the node considered (mobile phone vs. data center).

3.2 The Green Software Life Cycle

The discussion naturally converged to a focus of *software as object of the optimisation* with regard to impacts on sustainability. The optimisation potential can be analysed in terms of either energy savings (green IT), or business processes and non-functional requirements that are aligned with sustainability principles (green by IT). The current understanding of green and sustainable software includes both:

- a relative understanding of sustainability – a function shall sustain over a specific time; a completely neutral definition without relations to “higher” values; and
- an absolute understanding of sustainability – the software system or service shall contribute to preserving environmental and human well being.

These two understandings have to be clearly distinguished.

The major challenge in kickstarting sound research in green software is to investigate how to break down the *definition of sustainability* so that it can be applied to software engineering. Accordingly, we need to find out how to break these two abstract definitions down to software characteristics, how to determine the added value, and how to trace it back to software.

After having agreed on a common, reference definition of what sustainability means in and for software engineering, the discussion led to identifying further challenges ahead:

On quality: How does sustainability differ from other –ilities? So far, we know that we have to look at longer time horizons to draw conclusions, and that it is a complex characteristic that can be analysed for internal quality, external quality, and quality of service.

On requirements: What are the types of requirements that lead to sustainable software solutions? How do they differ from "traditional" Non-Functional Requirements (NFRs)? How does a "sustainable" business model translate into software requirements? Can we imagine NFRs that work against planned obsolescence? For example, can services be offered so that a lower level of quality of service can both use old hardware at a more economic rate, and in doing so lead to less e-waste?

On design: Can we find principles for engineering sustainable software as, for example, available for dependable systems? Can we learn from principles used for and in performance, safety, or usability? What are the green design patterns that offer recurring and generally reusable solutions for sustainable and/or energy efficient software?

For any of these challenges and questions, it is crucial to distinguish between the code and its functionality in order to not mix sustainable software and software for sustainability.

4. ACKNOWLEDGMENTS

GREENS is part of the dissemination activities of the Dutch Knowledge Network Green Software. We extend our thanks to those who have participated in the organization of the workshop, and particularly the program committee members. We further thank Qing Gu and Giuseppe Procaccianti for their feedback on this report.

5. ADDITIONAL AUTHORS

Additional authors: Giuseppe Scanniello (Universitá della Basilicata, Italy, email: `giuseppe.scanniello@unibas.it`), Birgit Penzenstadler (Technische Universität München, Germany, email: `penzenst@in.tum.de`) and Olaf Zimmermann (ABB Corporate Research, Switzerland, email: `olaf.zimmermann@ch.abb.com`).

6. REFERENCES

[1] Workshop on Green and Sustainable Software (GREENS). http://greens.cs.vu.nl.

[2] C. Pettey. Gartner estimates ict industry accounts for 2 percent of global co2 emissions. http://www.gartner.com/it/page.jsp?id=503867, 2007.

[3] P. Stoll. Blog on: Sustainable development of industrial software systems. http://piastoll.com.

[4] UNEP/GRID-Arendal Maps and Graphics Library. World greenhouse gas emissions by sector. http://maps.grida.no, 2009.