A Green Model for Sustainable Software Engineering

Sara S. Mahmoud and Imtiaz Ahmad

Computer Engineering Department, Kuwait University,
Kuwait P.O. Box 5969 Safat 13060
saramahd87@gmail.com, imtiaz.ahmad@ku.edu.kw

Abstract

Information Communication Technology (ICT) has a strong impact on sustainable development due its rising demands for energy and resources needed when building hardware and software products. Most of the efforts spent on Green ICT/IT have been dedicated to addressing the effects of hardware on the environment but little have been considering the effects of building software products as well. Efficient software will indirectly consume less energy by using up less hardware equipment to run. Our contributions in this paper are devoted to building a two level green software model that covers the sustainable life cycle of a software product and the software tools promoting green and environmentally sustainable software. In the first level we propose a new green software engineering process that is a hybrid process between sequential, iterative, and agile development processes to produce an environmentally sustainable one. Each stage of the software process is then further studied to produce a green and sustainable stage. We propose either green guidelines or green processes for each software stage in the engineering process. We add to the software life cycle the requirements stage and the testing stage. We also include in the first level a complete list of metrics to measure the greenness of each stage in terms of the first order effects of ICT on the environment for a green software engineering process. No effort has been placed before in designing a green software engineering process. The second level explains how software itself can be used as a tool to aid in green computing by monitoring resources in an energy efficient manner. Finally, we show and explain relationships that can be found between the two levels in our proposed model to make the software engineering process and product green and sustainable.

Keywords: Green Computing, Green ICT, Green and Sustainable Software Engineering, Green and Sustainable Software, Sustainable Development, GPIs, ASD, GORE

1. Introduction

With the growing demand of more complex software applications, Information and Communication Technology (ICT) has had a huge negative impact on the environment due to its increasing resource and power consumption. The effect of ICT on sustainable development [1, 12] especially on software is the hot topic now-a-days in Green Computing. Sustainable development refers to resource use for meeting the needs of humans while taking into account the ecological, economic, and societal impacts. Although ICT recently has been trying to find efficient solutions for the environment, it is not clear whether energy and resource savings by ICT will exceed its resource consumption.

A variety of research work on Green ICT has mainly focused on environmental sustainability in terms of computer hardware. But revealing the issues related to energy consumption in software can be a great help in achieving green computing. Software features
are responsible for CO\textsubscript{2} emissions as are hardware components. Software has an indirect effect on the environment by operating and managing the underlying hardware running it. Some software based solutions can monitor and utilize resources efficiently and others can be sustainable enough to limit the need of adding more hardware due to updates. Unfortunately there is a lack of models and work in the area of computer software and software development processes. This paper focuses on achieving green and sustainable software by building a green and sustainable software model that will aid software engineers in the development process of software and include the recent approaches taken by software to ensure the safety of the environment.

Recently many efforts have been done in obtaining green software. Some efforts are focused on building green and sustainable software, some design software processes to aid all stakeholders in building green software products. Others efforts are focused on building software tools that measure the effect of software on the environment and the effect of application development environments on the software in terms of energy efficiency. There are efforts that emphasize on the operating system to help control the power consumption of applications. General software solutions found in [4, 5] include virtualization, closing applications no longer in use, efficient algorithms by writing a compact design of codes and data structures, reduction of parallelism overhead by developing efficient load balancing algorithms, fine grained green computing, and creating energy allocation algorithms for routing data. Naumann \textit{et al.}, [1] came up with a conceptual reference model named GREENSOFT model for sustainable software. Their four part model supports software developers, administrators, and software users in creating, maintaining, and using software in a green manner. The four parts cover a life cycle model, metrics, procedure models, and recommendations and tools for different stakeholders. Shenoy and Earatta in [38] also provide a green development model in which they suggest steps that may lead to lower carbon emissions in the software life cycle stages. Maahux and Canon [2] argue that requirements engineering is critical to the whole software life cycle primarily in the usage phase where customers are delivered the system and expect it to conform to their requirements. They claim that correct requirements engineering can help software last longer thus reducing the energy consumption. Capra \textit{et al.}, [9] focus on developing a measure of energy efficiency for software applications and illustrate how application development environments can have a detrimental effect due to the additional lines of code they add. Gupta and Singh in [8] present a framework for creating an intelligent power profile that implement three methods at the time of login into the system. These methods continuously measure the power consumption of running software in a given period of time and can be incorporated in operating systems. In [10] an approach based on periodic measurements of GPIs and QoS and adoption of Service Oriented architecture is used to optimize energy efficiency at the Software-as-Service layer. Efforts spent on integrating sustainability in Service-Oriented software are found in [29, 31]. In [11, 12] efforts are spent on defining general good practices in green software engineering such as collecting requirements through electronic means and deploying the concept of virtualization. Research that focuses on the design of code and how it may cause bloating are found in [13]. An effort spent on energy efficiency through adding more cores on a single CPU can be found in [14]. Works that focus on the importance of requirements engineering for sustainable software are found in [22, 23, 24, 30, 33]. In [22] the software requirements engineering process focusing on sustainability requirements for a company named The Yellow Project is reported. An approach to greenify service based applications is achieved through integrating eco-aware requirements based on energy goals is found in [23]. In [24] it is argued that green ICT concepts related to software requirement engineering should be added to undergraduate software courses. In [30] a requirements
engineering approach is developed that allows engineers to handle sustainability as a first
class quality objective. Addressing sustainability of software processes is found in [32]. In
[34] efforts are spent in having clear metrics for measuring the carbon footprint of software
development, the amount of resources used by software, and how much damage it does to the
environment. Work found in [33, 39] focus on quality engineering based on the
measurements of software in terms of quality metrics. Works that are dedicated for sustainable
development in computer science are found in [3, 37].

In this paper we design a software model that covers all aspects of software related to
green computing. The model is a two level model in which the first level is a hybrid software
engineering process between sequential, iterative, and agile software development processes
that aims to create a green and sustainable software process. Having an environmentally
sustainable software engineering process will reduce their negative impacts of ICT on the
environment. The second level defines how software itself can be used as a tool to promote
green ICT by identifying all the approaches that have been taken. We categorize the software
approaches and concepts into different categories. No attempt has been made yet before to
provide a green software engineering process. We add in the first level the requirement and
testing stages to the software life cycle unlike other green models, and we further provide
guidelines and processes on how each stage in the software engineering process can be green
and output a green and sustainable software end product. We also define a complete list of
metrics to measure the environmental sustainability of each software engineering phase i.e. in
terms of the first order effects of ICT and show under which stage each metrics lies. Very
little work with no reasonable depth has been spent in considering the testing stage in terms of
environmental sustainability in the software life cycle. We also define relationships found
between the two levels of our software model.

This paper is organized as follows. In Section 2 we define the relationship between ICT's
and the sustainable development of green software and the impact of new software
development methods. The proposed green software model is described in Section 3. In
Section 4 we define how software tools and metrics promote a green and sustainable software
engineering process and Section 5 is the conclusion of our work.

2. The Relationship between ICT and the Sustainable Development of
Software and the Impact of New Software Development Methods

The impact of ICT on sustainable development may be positive or negative. Software is
used today to help optimize the energy usage and consumption in ICTs. Therefore it is not
enough to just focus on building green software application/products, software can also be
looked at in a point of view that can aid in monitoring and utilizing resources efficiently. The
effects of ICT can be summarized into three order impacts in which one impact results as a
consequence of the previous impact. Second order impacts occur only due to first order
impacts and third order impacts occur due to second order impacts. We will follow the
contributions provided by Naumann et al., [1] to identify impacts of ICT on sustainable
development because they are not limited to environmental sustainability but also include
human and social sustainability issues unlike the contributions of Berkhout and Hertin in [16].
First order impacts which are the most obvious are environmental effects that result from
performance requirements, network bandwidth, and product packaging. Second order impacts
are effects that result indirectly from using ICT i.e., effects that appear on the life cycle of
other products or services from the usage of services provided by ICT. ICT now-a-days offers
its services by the means of software. For example, software is used to optimize the design,
production, and disposal of the product being produced. It is therefore obvious that software
takes parts in the software engineering process of many other products or services. Third order impacts, famous for being hard to predict and assess are long term indirect effects on the environment that result from orderly effects of ICT. A well-known impact is rebound effects that may result from efficiency gains that stimulate new demand.

We focus here on the sustainable development of software. There are many meanings provided for green and sustainable development of software and green and sustainable software as there has not been a standard definition for it yet. We will follow the definitions provided by Naumann et al., in [1] for green and sustainable software engineering and green and sustainable software. Green and Sustainable Software Engineering is the art of developing green and sustainable software engineering process. Therefore, it is the art of defining and developing software products in a way, so that the negative and positive impacts on sustainable development that result and/or are expected to result from the software product over its whole life cycle are continuously assessed, documented, and used for a further optimization of the software product. Green and Sustainable Software is the 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. Another definition we are focusing on in this paper since our work is dedicated largely to creating environmentally software processes is related to software processes. That term is green and sustainable software process. A software process as described in [32] and established in the ISO/IE3 12207 standard is a collection of general definitions of the interrelated activities that can be suitably performed during the life cycle of a software product. To obtain a sustainable software product any processes contributing to its life cycle should sustainable themselves. We use the definition provided by Lami et al., in [32] to define the term green and sustainable software process. Green and Sustainable Software Process: Software that meets its (realistic) sustainability objectives, expressed in terms of direct and indirect impacts on economy, society, human beings, and environment that result from its definition and deployment.

Software has not been proving to be energy efficient lately. “The software development life cycle and related development tools and methodologies rarely, if ever, consider energy efficiency as an objective” [9]. Designers and developers usually disregarded the issue of green software due to the existence of efficient and cheap hardware. But the limited efforts spent on software energy efficiency include, the question if application development environments have a positive impact on software application, the issue of performance, and issues in the complier level. Application software written with high level languages and libraries reduce the coding tasks for programmers but may transform the code into a complex one lacking energy efficiency. Application development environments speed up the development process by providing hardware and software tools such as libraries of code modules that can be re-used. We still do not know if application development environments provide any energy efficiency for software applications, but Capra et al. [9] give a hypothesis that they are detrimental to software energy efficiency. They claim that the by concealing the underlying complexities of technologies, application development environments cause extra work on the processor when a program is executed. Re-using libraries of code modules add additional lines of code causing extra CPU-time during execution. Some efforts to increase energy efficiency were improving the performance of software applications fooling developers into thinking that will reduce energy consumption as a result of less CPU usage. However, improving performance may cause an increase in CPU usage because performance is usually measured in terms of response time. Thus performance and energy consumption are two conflicting issues. “The compiler can affect power consumption by applying program
transformations or by driving dynamic voltage scaling and adaptive body biasing” [9]. Further research and investigations must be put in how to obtain green software.

As new technologies arise and advancements in the software engineering topic take place a new approach to the software development life cycle that may assure a more sustainable development process has been bringing great attention. This new approach in [25, 26, 35, 36] to the development process is named Agile Software Development (ASD). The principles brought forth by the 17 software professionals that make up this approach are based on best practices and previous experiences with numerous software development projects. There are different agile methodologies but they all are based one on set of 12 principles. These 12 principles promote mainly close collaboration between software development and business teams, face-to-face communication, early and frequent delivery of increments of the working software, and accepting changing requirements by customers. The agile approach to software development is different from old and traditional ones in that it does not rely on heavy written requirement documents that are unaccepting to any requirement changes, it does not prevent business people and customers to work hand in hand with the software developers, and it's one of its kind that promotes sustainable development by believing that developers, users, and sponsors should maintain working in a constant pace. For example, it encourages lean development through iterative and evolutionary approaches.

3. Proposed Software Model

In this paper we focus on designing a model that covers all aspects of how software can affect the environment. Of the models most popular that refer to the same aim is the GREENSOFT model proposed by Naumann et al., in [1]. The model supports all stakeholders from developers to users to create, maintain, and use software in a green fashion. The model comprises many aspects of a software product and organizes them into four parts. The parts include the life cycle of the product being produced, metrics that should be added as an evaluation criteria, procedure models, and recommended tools for developers, purchasers till users. The model addresses the complex sustainability concept and forces it to be integrated into the complex software engineering process in a smooth way. In each life cycle stage, it states the effects of ICT on the product and the environment. This is one of the hardest tasks and is one of the main challenges today in Green Computing. There is still a lack in the provision of such software processes.

The model proposed by Naumann et al., [1] lacks in including and mentioning the role of software itself in maintaining and optimizing energy usage in ICTs. In our model we argue that to have a full view of how software affects the environment, we should consider along with how the life cycle of a software product effects the environment through ICT, how software itself can aid in keeping the environment green or not. The second level of our model indicates how software itself can monitor and utilize resources efficiently thus aiding in achieving green ICT. We study and review many approaches and concepts based on software to promote green computing and find that we can categorize them into 5 different categories. In terms of the software life cycle we argue that the requirements stage in the general software engineering process is very important in structuring how green a software product will turn out to be and therefore is critical and should be added to the life cycle of a software product. This stage affects all the next stages because if some requirements of the system being built were not included or were misunderstood by the requirement engineer then there is no way to avoid making changes in the design and implementation stages if it were to be accepted by the customer. We also argue that the testing phase should be included in the software product life cycle since this stage also affects our environment. Very little effort with no reasonable depth has been spent in including the testing stage in any works in the
field of green computing as far as we know. No effort has been spent before in designing a
green software engineering process. We created a software engineering process taking the
essence of the sequential, iterative, and agile software development methods to form a green
and sustainable software life cycle. We further put effort into turning each stage into an
environmentally sustainable one by creating green processes or listing green guidelines to
follow. We also mention the metrics that can indicate the greenness of each of the stages in
the software engineering process based upon the green performance indicators accepted for
the EU Project GAMES. Our proposed model is explained next in detail. In section A we
describe in detail level 1 and in section B we describe in detail level 2 of our green model.

3.1. Level 1

The first level represents how to obtain a green and sustainable software product. The
popular and generic software engineering process is made up of seven main stages. From the
beginning to the end the stages are in this order: Requirements, Design, Implementation,
Testing, Usage, Maintenance, and Disposal. There are many models with different variations
of these stages and many companies adopt their own but they all tend to have similar
patterns. The first part of the model proposed by Naumann et al., [1] presents the life cycle of
a software product and how each stage in the product’s life cycle promotes effects on the
environment due to ICT.

But the life cycle process indicated contains only five (if you consider design and
implementation to be in the same stage called development and usage to be usage and
maintenance) stages. We argue that all the life cycle stages of the software engineering
process must be included. Their model is missing the requirement and testing stages. These
two stages play a major role in outputting a green and sustainable software product and can
stimulate effects on the environment. We propose our own software engineering process
composed of nine stages that can aid in outputting a green product and designed to have
environmentally sustainable stages.

In Figure 1 is the first level of our green software model which is composed of the
software engineering process we propose with all the metrics related to each stage. We
designed this process to embed green concerns making it an energy efficient process.
Software engineering processes usually either reflect an iterative approach to the life cycle or
a sequential approach. Our model mainly falls on the iterative approach instead of the
sequential approach for reasons that can lead to it to be green. The model also incorporates
some of the principles found in agile methodologies to help in including environmental
sustainability.

Linear sequential life cycle models have disadvantages that affect the greenness of a
software process. In a sequential life cycle each phase must be completed before the next
phase can begin leaving no space for the possibility of changes in the requirements that can
lead to changes in the design and implementation, and if changes were to happen then
implementing them is very costly and energy inefficient due to the commitment to the
requirements document. There is no way to avoid this concern of changes or
misunderstandings in the requirements between the customer and the requirement engineer,
and sequential life cycle models deny this truth. Sequential life cycle models work well only
when the requirements are very well understood and are unlikely to change. On the contrary,
sequential models can promote energy efficiency practices as well. Sequential models such as
the V-model allow for developing tests early before implementation leading to a higher
chance of success. This can avoid going back to fixing errors and changing requirements
when problems are revealed late after implementation. Finding errors and mistakes earlier is
always a good energy practice because it is less costly.
Iterative life cycles can truly conform to the demands of energy efficiency because their main purpose is to coincide and aggregate the requirements with the development of the software. This means that new requirements, changes in requirements, and misunderstanding requirements are a less severe problem to cope thus making iterative software engineering processes energy efficient. In an iterative life cycle there is no obligation to a long and comprehensive requirements document. This idea is implemented in different ways in different iterative life cycle models such as the incremental delivery model where iteration is found in early delivery of increments of the system and the spiral model where iteration is found in risk analysis and development of prototypes. Iterative models can also lower the risk of overall project failure. On the contrary there are some characteristics about iterative models that do not support energy efficiency. Some iterative models such as the incremental delivery model do not allow changes in requirements of a delivered increment. It is very difficult and costly to try to change old increments since they have been already delivered to the customer.

Agile principles if incorporated in software engineering processes can definitely aid in having environmentally sustainable processes. From the known twelve agile principles, some may have a direct impact on green and sustainable software processes. Changing requirements even late in development is an agile principle that can promote green engineering because it is inevitable that changes will occur and there must be an energy efficient way to deal with this problem. Another agile principle supporting a green process is allowing business people, customers, and developers to work together daily throughout the project. If the customers are closely involved in the development project they can provide new requirements and evaluate the different iterations making it less likely for many changes to occur. Also with the business people they should work hand in hand with the software developers due the changes found in all the processes of each stage.

In our model we combine the good in terms of energy efficiency from both sequential and iterative life cycle models, but it is mainly iterative. The model does not implement the idea in iterative models where small increments of the system are delivered early to the users due to the denial of changes in these increments. Instead it implements the concept of iterative development of the system by developing very small increments but not releasing to the users so that if any changes are need they can be done. Our model also incorporates a second concept of iterative development by benefiting from the incremental early development that leads to easily clarifying the requirements for later increments giving them a higher chance to be unchangeable and correct thus leading to energy efficiency. In these two ways the agility principle related to welcoming changes in the requirements is incorporated. The model also has an essence of sequential life cycles because it incorporates early testing before coding. The model naturally begins with the requirements phase, and then goes on to design, and then the third stage is unit testing. To implement any changes, from there the model allows for going back to the requirements stage again. The fourth stage is implementation with the output of a small increment, and then comes acceptance and system testing on that current increment. After the system/acceptance testing stage faults and errors may arise leading to going back to the requirement and the previous stages for changes and repair. A sixth stage not found in any software engineering process is green analysis. This stage is concerned with using metrics as a measuring criterion to indicate the “greenness” of the current increment. Depending on the values from this stage the model allows for going back to the requirements stage to implement changes that can lead to more energy efficient software. Because the increments are small applying changes is not costly. The system can then be released to the users. The seventh stage is the usage stage, and then maintenance which can go back to the requirements stage for improvements and updates. Finally the last stage is disposal of the software product. We emphasis in the yellow box in the model as seen in Figure 1 that the
customer and the business people should be part of the requirement, design, implementation, and testing stages. We describe below each stage of our proposed process in details.

**Figure 1. Level 1 of our Proposed Green Model**

3.1.1. Requirements Stage: Requirement engineering is the first stage towards building a software product. The requirements are the descriptions of the services that should be provided by the system and any limitations. The requirements engineer's job is knowing what the client wants the system to do. The process in Figure 2 is our proposed energy efficient requirement engineering process. The first stage is the feasibility stage already existing in previous requirement processes. This stage determines if the system to be built is relevant and useful to the business. This stage can aid in energy efficiency because if no advantage will be gained from building the system then it is energy efficient to decide not to process with developing it. The next two stages are based upon the requirements process of iterative development. The customers identify at this stage the outline of the services they wish their system to perform. Next they organize the services in the order they should be developed instead of prioritizing the services in the order they should developed to be delivered to the user as with all iterative processes. The fourth stage is a risk analysis inspired from the iterative spiral model but in terms of energy. Based on the analysis of this stage an alteration in the requirements may be carried out. As a result, from this stage the process goes back to the second stage to implement these changes if any. The final stage is the requirements test stage which we believe should be added to an environmentally sustainable requirements process. This stage is used in extreme programming. It is energy efficient to develop tests along with requirements because it provides a better understanding to the testers and developers of the requirements and mainly satisfies that there will be no changes when system and acceptance testing occurs.

**Figure 2. Green Requirements Engineering Process**

Other approaches to requirements engineering such as goal-oriented requirements engineering (GORE) found in [28] has been an important research topic lately. This approach relies on goals in which each goal may be determined to address different types of abstracts, to specify and modify requirements. If a non-functional requirement or a main soft goal to a
system is environmental sustainability, then the set of general operational or low level soft goals for satisfying environmental sustainability we represent is found below.

1. Use of environmentally approved products for development
2. Reduce transportation means and instead use the internet for communication
3. Use of Cloud Computing in terms of IaaS (Infrastructure as a Service)
4. Use of service oriented software
5. Sacrificing performance above a limit for energy efficiency
6. Running the system on computers with power profiles
7. Reduce the number unnecessary activities in the system
8. Use of a reusable system

3.1.2. Design and Implementation Stage: In the design stage overall system architecture is created based upon the requirements listed in the requirements stage. During this stage fundamental software system abstractions and their relationships are defined. There are a number of design activities that shape the sustainability level of the software component such as architectural design, abstract specification, data structure design, and algorithm design. Below are three design decision guidelines that promote environmental sustainable in the design stage.

1. Programmers should write efficient algorithms via writing a compact design of codes and data structures based upon the application, programming language, and the architecture of the hardware. Optimization comes from experience.
2. The sub-systems should stick to the functions presented in the requirement stage and should be designed to be minimized with smart efficient functionality thus producing an efficient algorithm and a less number of program lines in the implementation stage.
3. The effect of reuse and application development environments specifically frameworks and external libraries has been proven by Capra et al. [9] to be detrimental in terms of energy efficiency for large applications due to the additional software layers that are not needed and are added to the code. These added software layers require extra work from the processor when the program code is in execution.

In Figure 3 is a design process we propose dedicated for component re-use in an energy efficient manner. The first stage of our process is like the known component identification process used in software and that is component searching. Then it goes on to selecting the most appropriate components that will conform to the system to be built. We propose for an energy efficient process that in stage three the components selected should be customized to conform to the requirements to avoid the negative effect on the environment due to extra lines of code not needed and extra work by the processor during execution. Customizing the components is easier, less costly, and faster than as previously suggestions for modifying the requirements instead. Modifying the requirements lets the system lose its uniqueness and requires going back to the requirements stage to re-do the whole process. This modification may affect the functional and non-functional requirements by adding unneeded ones thus
losing the customization aspect of the application and increasing functionality affecting energy efficiency. The validation stage is the last stage.

![Component Search → Component Selection → Component Customization → Component Validation](image)

*Figure 3. Green Design Engineering Process*

Implementation is the stage after design in which the design is implemented into a set of programs and program units. Software developers should choose at this stage the most suitable programming fashion to the application and based on that a programming language. For example, an application with many exceptions during run time should be written in an Aspect Oriented fashion instead of object oriented fashion.

3.1.3. Testing Stage: The software testing process can emphasize on either discovering that the software does not meet its requirements or can emphasize on discovering faults or defects in the software where the behavior of the software is incorrect. To our knowledge the testing stage has been barely considered by any sustainability research work till now. What makes the testing process one that can be detrimental to the environment is that it is an iterative one, meaning that if the customer does not approve that the software is functioning as it should be indicating that the developers do not understand the requirements then as a result the engineers must go back and make changes in the requirement stage and the development stage. This means more working hours, energy for ICT, office lighting etc. For a more green and energy efficient software testing stage, tests should be developed earlier like in the requirement stage to assure the tester and developer that the requirements are correct and conform to the user's expectations.

We present some software engineering metrics or quality attributes that can be specifically useful for measuring the environmental sustainability of the software system in the testing stage. If these metrics are improved this may lead to a more environmentally sustainable testing stage. For each metric, we provide its definition first then describe the environmental benefit that can be achieved when improving this metric.

1. Fault Tolerance: enables a system to continue operation at a reduced level rather than failing completely.

   Environmental Sustainable Benefit: minimizes environmental waste through allowing the system to function even with defect instead of shutting down and requiring repair.

2. Failure Management: involves identifying, responding to, and repairing failures of a software system.

   Environmental Sustainable Benefit: a good failure management can carry out a successful failure process and as a result reduce environmental waste by preventing future failures.

3. Testability: degree to which a software artifact supports testing in a given test context.

   Environmental Sustainable Benefit: reduces e-waste by supporting an easily tested system.

In Figure 4 is the popular testing process used in software engineering. We propose to add to this model a Requirement Stage as the first stage to include energy efficiency into consideration. As mentioned above in the requirement section, having a testing stage before...
development starts allows developers and tests to have a clearer view of the requirements and as a result ensuring no delays or going back to the requirements stage when testing for component and acceptance/system. This ensures that the system functions as intended by the requirements thus confirming a green and sustainable software testing process by decreasing the number of iterations that lead to going back to the previous stages and fixing errors.

3.1.4. Green Analysis Stage: The green analysis stage is a stage we propose to be added to a software engineering process to promote energy efficiency. It is a stage that brings forth new ideas about environmental sustainability that has not been considered before to be added to any software engineering process. The green analysis stage determines the greenness of each increment of the system that is developing. This stage acts like a testing stage but for energy efficiency. Metrics are used in this stage to perform the analysis. We suggest that the IT resources usage metrics such as CPU usage and the Quality metrics such as performance from the Green Performance Indicators accepted by the EU Project GAMES should be used as the measuring criteria. In Figure 5 is the process we propose for this stage. It begins with defining which IT resource and quality metrics should be used for the desired system, and then it goes on to collecting the data from the software and taking the measurements based on the formulas accepted by the EU Project GAMES. Many software tools that we mention in level two of our model can be used in this stage to collect the data from the underlying hardware and map the energy wastage with locations in the code. The third stage analyzes the results from the formulas and determines if changes need to occur in the requirements, design, or implementation stages. There are guidelines defined by eco-related laws and regulations that place a standard on the energy levels systems should abide to help determine this. The final stage implements any changes that need to be taken based on the results.

3.1.5. Usage Stage: This stage focuses on the how the user of the software product should use the product in a green manner. For this stage to be a green the responsibility does not only fall on the user but also on the engineers themselves. Below we provide green guidelines in terms of three stakeholders: software requirement engineer, software developer engineer, and the user.

Software requirement engineer:
• Functions should be written that promote sustainable behaviors by the users.
Software developer engineer:

- Developers can create software that engages in power management features on the computer to allow resources to be used efficiently i.e. shut down when idle.

User of the application:

- Close the application when it is not in use or switching off the computer and turn it on when it is needed again.
- Power off the monitor when not in use instead of using screen savers.
- Enable the standby/sleep mode and power management settings on the computer.

3.1.6. Maintenance Stage: Software maintenance or evolution is a stage involved in changing a system after it has been delivered due to newer versions or enhancements. The maintenance stage is the most costly phase of all the software engineering process. Cost here is proportional to the energy wastage. The changes made to the software application being built may be simple as in just changing errors in the code by going back to the implementation stage, or larger changes to correct design errors, or the most expensive and difficult changes to correct which are changes in requirements or the addition of new ones. We need to carry out the process of change in an energy efficient manner. Below we provide guidelines to help having an environmentally sustainable process of change.

1. What makes the cost of maintenance so high is the amount of effort placed into understanding the system and analyzing the impact of the proposed changes. Usually when a system is developed its maintenance contract is separate from its system development contract and it probably will be given to a different company rather than the original system developer. If the software is not written and saved somewhere for the new company to access it and perform their maintenance job then it will be much harder for them to make the changes and thus costly and definitely not energy efficient.

2. Maintenance is usually seen as a less skilled process than the software development process and thus given to inferior staff to perform who might not have experience and knowledge of old and outdated programming languages. Thus it is the top managers' responsibility to give maintenance staff tutorials and courses to be more familiar with old and new programming languages. That will speed the process of maintenance reducing cost and increasing energy efficiency.

3. It is important that during the development stage a program be written mainly to for well structuring and understandability. If a program is easily understood then it will much faster and easier to apply changes to it and that will reflect upon energy efficiency.

4. System documentation must always be available and consistent to the code.
3.1.7. Disposal Stage: This stage accounts for the replacement of software and hardware that is no longer up to date, no longer used, or has become obsolete. The scope of the disposal stage covers the software and the hardware running the code. Disposal of old hardware causes vast amounts resource and energy consumption. Below are guidelines for energy efficiency in this stage.

1. Software recycling in terms of reusing the software code for future projects, thus keeping in-house software development costs down.
2. Hardware recycling in terms of following reusing and recycling before disposing the equipment and materials.
3. Products that can be used repeatedly should be bought to save natural resources.

3.2. Level 2

Our model defines all the dimensions of software in terms of the development process and software tools that help limit the energy wastage of running applications. The second level of our green model shown in Figure 6, indicates how software tools can play a major role in having energy efficient use of software applications thus promoting green computing. Figure 6 represents the five categories for software tools and concepts that we define with examples. Software has a larger effect on the environmental sustainability of computer systems as opposed to hardware. For example, using monitoring counters and software to quantify power dissipation in parts of the code can help in writing power-efficient software. Applying power scheme features in the new operating systems is another approach using software to manage the use of hardware resources. After reviewing many software concepts such as fine tuning and works dedicated for using software as a tool to promote energy efficiency, we categorized these software approaches taken into five categories with examples: operating systems frameworks, fine grained green computing, performance monitoring counters and metrics, codes written for energy allocation purposes, and virtualization.

Referring to the operating system, frameworks have been proposed for monitoring the energy of running applications. For example, frameworks that create intelligent power profiles are injected into the operating system code to minimize power consumptions of computer systems. They can minimize the average work load of the CPU by shutting or hibernating applications not in use for some time resulting in less heat dissipation and minimized power consumption as defined in [8]. Other frameworks rely on the operating systems to report on the energy consumption of system processes through raw information collected directly from hardware devices or through the operating system at run time. Sensors are used as the source for collecting information such as CPU usage or network usage as defined in [7].

Other approaches such as power analyzers for software are based on performance monitoring counters and frequency information from CPUs to quantify power dissipation on real computer systems. Such approaches like the SPAN in [6] can correlate power estimation with application source codes through a set of API calls. Another approach that falls into this category is a software tool named the GREENTRACKER in [17]. It measures the energy consumption of software and focuses on comparing different systems that serve the same purpose instead of comparing among different versions of the same system. Similar approaches are also found in [18] that map software design to measurements of the power consumption of the CPU, memory, video, audio, and the Ethernet core. One more approach found in [19] runs source code instrumentation on the system. The instrumented application
uses APIs provided by Intel and is connected to a measuring system where a server takes the measured values and converts them to performance counters. All approaches that fall into this category have the aim of providing software designers useful information about the power behavior of the software being developed for better program writing in terms of energy.

![Figure 6. Level 2 of our Proposed Green Model](image)

Approaches related to fine grained green computing are more specific to a running application in terms of not activating devices such as memory banks or I/O peripherals the application is not using unlike coarse grained computing which does not contribute to energy efficient use of resources are found in [5].

Other general approaches are codes written specifically for energy allocation purposes that can successfully route traffic to locations such as data centers with the cheapest energy costs or ones not experiencing warm weather are included in [4].

Another general approach is virtualization which plays a role in green computing and is partly software. Virtual machine software are partitioned based operating systems that allow for multiple applications to exist on a single system as defined in [4]. This can reduce the number of systems needed and the amount of power required contributing to green computing.

4. How Software Tools and Metrics Promote a Green and Sustainable Software Engineering Process

We explain here how energy related metrics and software tools can aid in allowing the software engineering process to be environmentally sustainable thus contributing to green computing. Metrics are an evaluation criteria indicating how green and energy efficient a system, an, application, or in this case a software engineering stage is. They quantize the amount of energy wasted or used efficiently and as a result, indicate how green a computer system is. Software tools can be built to keep track, identify, and limit the amount of energy used by running applications and software engineering concepts such as fine tuning can also definitely aid in having a green software product when incorporated in an application code. These tools can be required to run the application, or used throughout the development process, and such concepts can be implemented in the software application being built.
In our model we focus on indicating the energy consumption of each stage of the software engineering process through metrics instead of considering in which process stage the quality attributes of the application take effect as many previous works. Measuring the amount of energy consumed by ICT is a direct way of quantizing the amount of energy wasted or used efficiently by each stage in the software engineering process. The metrics we refer to here are used to measure the first order impacts of ICT on the environment that affect each stage unlike the metrics referred to in our Green Analysis stage that measure the quality attributes and hardware resource usage of an application. Energy awareness in systems can be obtained and calculated through green metrics such as the Green Performance Indicators (GPIs) found in [15, 20, 21, 27]. The GPIs are classified into four classes: IT Resource Usage GPIs that compute resource usage, the Application Life cycle KPIs that define efforts required to develop or redesign applications and reconfigure IT-infrastructure, the Energy Impact GPIs that represent the environmental impact of data centers, and the Organizational GPIs that describe organizational factors. To measure the environmental sustainability of each stage in the software engineering process we consider metrics from the Application Life cycle KPIs class, Energy Impact GPIs class, and from the Organizational GPIs class. We select the metrics suitable for the measurement of the first order effect of ICT in each stage in our software development process. We referenced our previous work in [20] for the selection of the metrics. In Figure 7 below we list the metrics that measure the greenness and sustainability of the stages of the software engineering process we propose in level 1 and under which stage they lie. We divide the metrics based upon the stages they belong to in terms of measuring the first order impact of ICT on the environment. The metrics found under the two categories energy impact and organizational GPIs may all measure any stage in the software process.

Metrics that can be used to measure the total process life cycle expenses are found Application Life cycle KPIs. Using metrics that measure and describe the total process life cycle expenses give developers a clear view of the greenness of each stage. They are named Life cycle Cost Indicators. These metrics take into account potential parameters such as the developer’s experience, the main service operation complexity, the level of abstraction, the reusability and integration rates, the required stability, and the closeness of the application to the business core. If we consider the cost of development metric, the parameters that are considered are the efforts placed by the developers during the application life cycle. More precisely, if the application runs are observed throughout time it can be determined how an application should be redesigned to be more energy efficient.

Metrics that measure the energy consumption or efficiency of software engineering stage can be listed under Energy Impact GPIs. These metrics describe the impact of ICT in terms of productions practices and usage on the environment. These metrics measure the amount of power supply, consumed materials, CO₂ emissions, and other energy related factors released in the air. The metric IT Hardware Power Overhead Multiplier for example, defines how much power input to hardware is wasted in power supply for fans rather than useful computing components and is given by this equation: AC hardware load at the plug/DC hardware Compute load [20].

Metrics that measure the infrastructural costs, human efforts, material productions, and abideance to eco-related laws in each software engineering stage are found in Organizational GPIs. Measuring the organizational factors related to infrastructural costs or conformation of ICT centers to eco-related guidelines can directly indicate the environmental sustainability of the stages of the software engineering process. For example the Human Resource Indicator metric assesses efforts spent by human resources involved in running and managing an
application, and Return of Green Investment measures the time it takes for green solutions to pay off or recuperate.

**Figure 7. Metrics that Measure First Impact of ICT**

The five software approaches and concepts found in level 2 can be used and referred to in our software engineering process to further ensure its environmental sustainability. Figure 8 presents our complete green model with both levels together.

When one of the non-functional requirements to a new software system is environmental sustainability, requiring that the software should run on an operating system with intelligent power profiles will aid in green computing. As a result there is a linkage between "operating system frameworks" in level 2 and "requirements" in level 1 which represents that software tools such as operating systems with power profiles can be related to the requirement stage because that can be a requirement asked for by the customer. There is also a linkage between the "operating system frameworks" and "usage" in level 1 that shows that user while running the new software can be running it on a computer with an operating system deployed that has power profiles.

Fine grained computing in the design of the software system can be required by the customer or suggested from the engineer when a non-functional requirement such as environmental sustainability is desired. The code for the application can be designed and implemented to fine tune the usage of the underlying hardware running the code. As result there is a linkage between "fine grained green computing" in level 2 and "requirements" in level 1. There is a linkage also between "fine grained green computing" in level 2 and "design" in level 1. This link means that the development engineers are writing the system code in a fine grained manner in which devices are only activated when they are needed.
Software tools for power analyzers based on performance counter and frequency information can be used during the design stage by the engineers to aid them in locating the areas in the code where most energy is dissipated. As a result changes in the program can be made to make the code being developed more energy efficient. As a result there is a link between "performance monitoring counters and metrics" in level 2 and "design" in level 1. These tools and also be used during the green analysis stage when collecting data and altering and analyzing results in terms of energy efficiency. Therefore there is also a link between "performance monitoring counters and metrics" in level 2 and "green analysis" in level 1.

Codes written for energy allocation purposes can be used alongside on another software application that is geographically distributed on another computer with other services in service oriented architectures. Thus there may be a link between "codes written for allocation purposes" in level 2 to "usage" in the software engineering process because such software can be used with another software application through the web to provide its service.

Requiring that the system runs on a virtual machine is a requirement that can be given by the customer to aid in having an environmental sustainable software system. There could also be a link between "virtualization" in level 2 and "requirements" in level 1.

4. Conclusion

Software has a great effect on the environment just as hardware even though the effect is indirect. Not only should software be written efficiently to not over use the underlying hardware, its engineering building process from requirements to disposal should be carried
out in an energy efficient manner and a new green software engineering process should be created to fulfill the constraints of energy efficiency. This will reduce the negative effects of ICT on sustainable development and the environment. It is also important to take advantage of software tools that monitor resources in an energy efficient manner and take part in power scheme features. We developed a green software model with two levels. The first level proposes a new green software engineering process we designed based upon the development processes of sequential, iterative, and agile methods. We further also identify how each software engineering stage can be environmentally sustainable through green processes and/or green guidelines thus ending up with a green software product, and finally we include the metrics we consider relevant to measure the greenness of each software stage. The second level is composed of approaches taken by software itself to contribute to green computing.

We categorize these different approaches and concepts into five main categories. Finally, we relate both levels to each other to indicate how software tools built to promote green computing and software concepts can be used and referred to in the stages of a software engineering process to help output a green and sustainable software product and have a green development process. In the first level concepts such as allowing the possibility of changes in the requirements, doing development tests early before implementation, aggregating the requirements with the development of the software, refusal of a long and comprehensive requirements document, and allowing business people, customers, and developers to work together all aid in having a green and sustainable software engineering process.

References


Authors

Sara S. Mahmoud received her B.Sc. degree in Computer Engineering from Kuwait University, Kuwait, in 2010. Currently, she is pursuing her M.Sc. in Computer Engineering with the Department of Computer Engineering at Kuwait University and also working as a teaching assistant in the same department. Her research interests include green computing and computer networks.

Imtiaz Ahmad received his B.Sc. in Electrical Engineering from University of Engineering and Technology, Lahore, Pakistan, an M.Sc. in Electrical Engineering from King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, and a Ph.D. in Computer Engineering from Syracuse University, Syracuse, New York, in 1984, 1988 and 1992, respectively. Since September 1992, he has been with the Department of Electrical and Computer Engineering at Kuwait University, Kuwait, where he is currently a professor. His research interests include design automation of digital systems, high-level synthesis, and parallel and distributed computing.